# vsTASKER

# User Manual

Reference Manual
2021

# Table of Contents

**Table of Contents**

**Table of Contents**

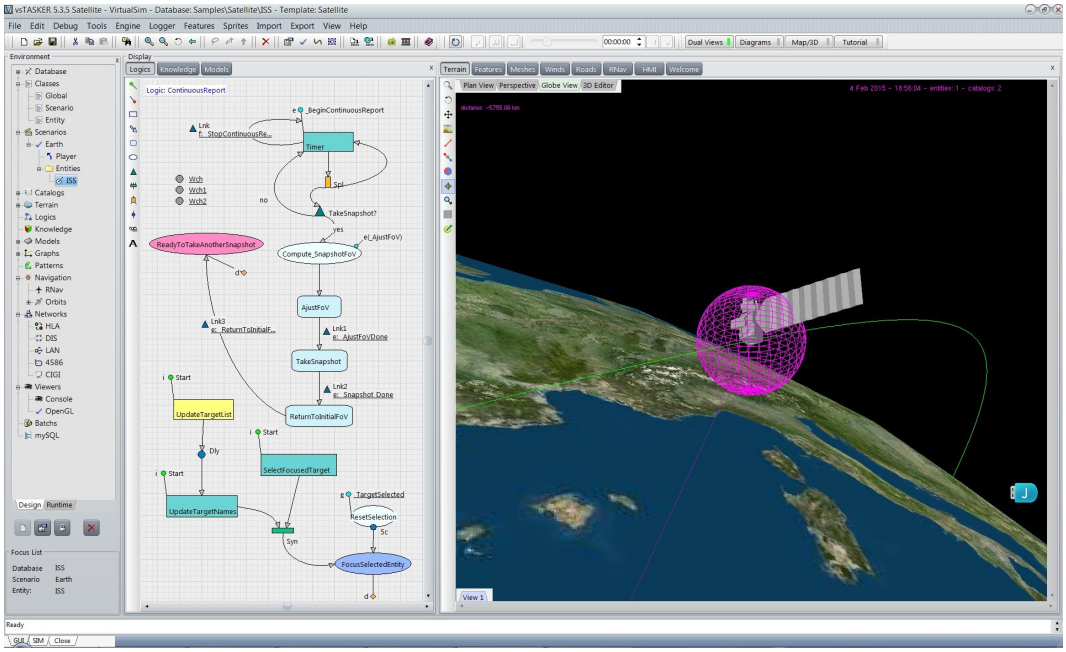
**Table of Contents**

# vsTASKER User Manual

This manual does not replace the training course (basic or advance) and is mainly intended to provide a guidance (or a reminder) for a good understanding of the product. The document covers all the user interface parts of the GUI designer. It is called from any Help button of the GUI interface.

Refer to the Developer Guide for the runtime simulation engine part.

# Before Starting




vsTASKER, the Simulation Development Tool kit for integrators and simulation engineers

# Presentation

vsTASKER is a powerful, innovative ands unique environment for tactical simulation. It is a real-time simulation platform for integrating third party applications or developing and testing all kind of real-time models and concepts by immersing them into a synthetic world, with immediate result and output.

With vsTASKER you can develop and test behaviors and rapidly see the result. Because the tool relies on user entered C/C++ code, engineers will find it very open, powerful and convenient to create all kinds of simulations.
vsTASKER generates C++ code based on diagrams, allowing users to concentrate on their knowledge. With this paradigm, tries and errors as well as what-if procedures are made easy to do and very intuitive.

Real-time simulation is often a tedious work to achieve because it integrates and binds together numerous pieces of software (mathematics, modeling, 3D, physics, sounds, network...). Most of the time, predefined or COTS products cannot be tailored enough to target the needs. Engineers must add numerous user-modules to extend capabilities and at the end, the final result looks like a mix of heterogeneous things nobody wants to touch.

vsTASKER greatly helps designing complex simulation by providing a code generator that combines graphical diagrams with user entered code, to produce an efficient and powerful simulation engine. Moreover, the simulation engine is license free, enabling all kind of deployment. As anything can be modified graphically, on a click of a mouse, the tool regenerate the corresponding code, free of bugs, saving long and tedious coding.

vsTASKER includes several core features that helps simulation designer to achieve complex simulation schemes with less efforts and within shorter time. State machines and Knowledge base, path finding algorithms on grid and networks, DIS or HLA made easy, HMI builder (...) are built-in capabilities amongst many others.

Imagine you want to develop a conduct for human characters being trapped into an office with terrorists or smoke and fire. With vsTASKER, building graphically a typical behavior for hiding, running away, finding exit, etc. can be done in couple of hours by one engineer. Testing behaviors and models will be first done using a convenient and fast 2D OpenGL display, and then conducted using a stealth viewer from any popular 3D engine tool. The tool also provides useful batch mode which, combined with the powerful runtime engine, allows easy data analysis on multiple runs.

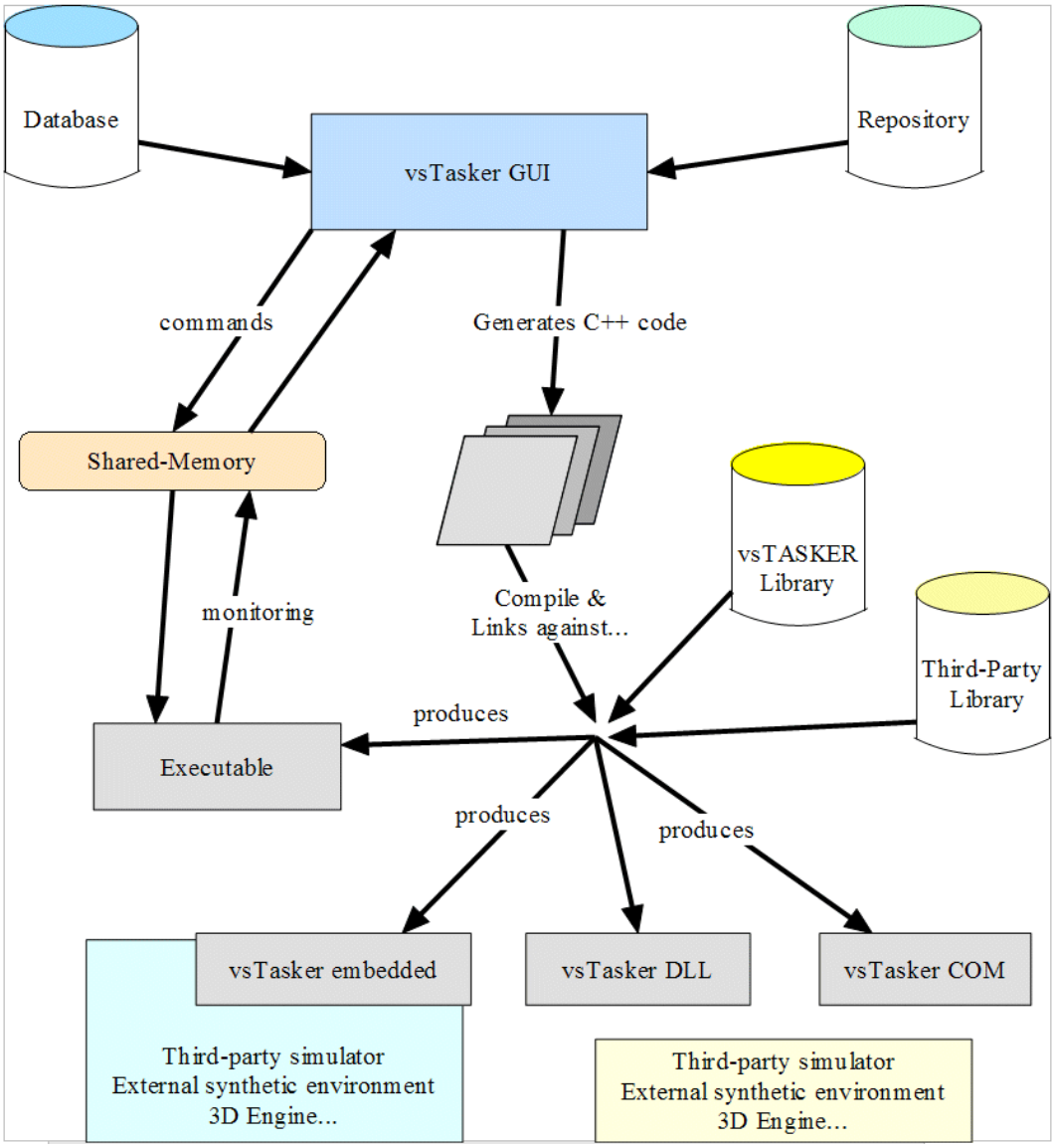
Besides, vsTASKER is an white box and can import or encapsulate any external library, as long as the API is provided and that the Microsoft Studio linker will recognize it.

### Presentation

Using several communication protocols, vsTASKER simulation engine can also connect, share data and control remote process systems. It can then be used as the master (logical) unit of a complex heterogeneous simulation system or just an hidden part, managing a complex logic in real time.

Because user can change behaviors during runtime (manually by direct action on diagrams, reconnection of logics, event triggering or component activation), constructive simulations can be carried out without the need of complex and costly dedicated user interfaces.

With vsTASKER and Visual Studio (Express or Professional), engineers can start building their own simulation application without pain and distribute them free of charge. The tool will quickly become a part of their development process, from small and quick prototypes up to big projects.

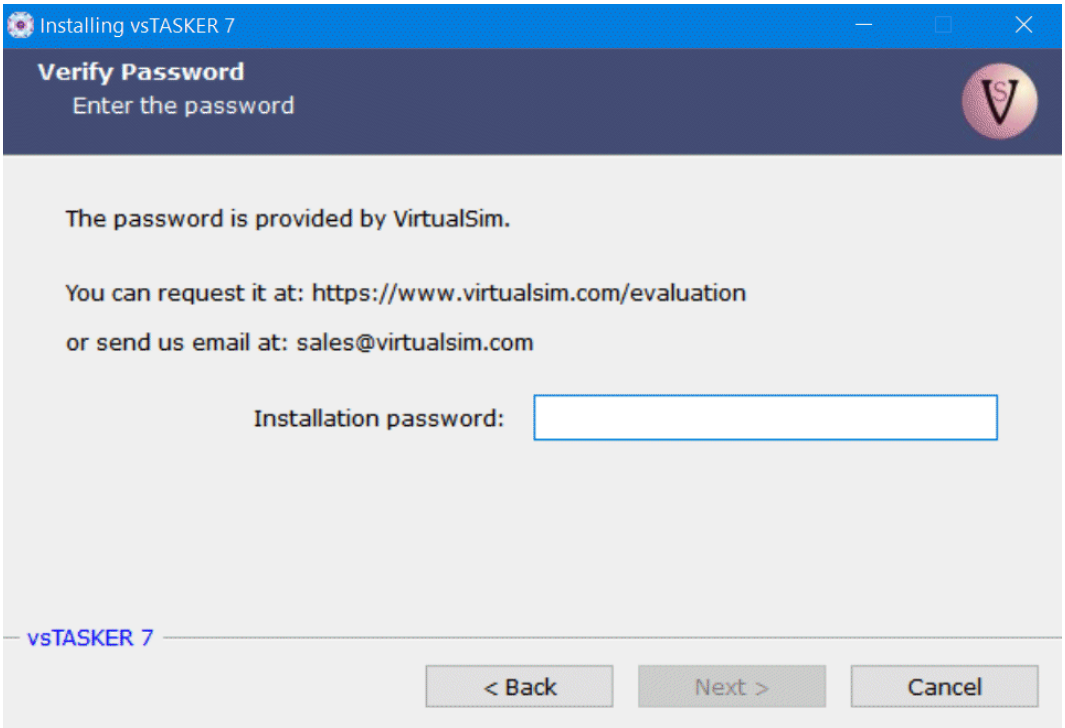

Database
Repository
vsTasker GUI
commands
Generates C++ code
Shared-Memory
vsTASKER
Library
Compile &
Links against...
monitoring
Third-Party
Library
produces
Executable
produces
produces
vsTasker embedded
vsTasker DLL
vsTasker COM
Third-party simulator
External synthetic environment
3D Engine...
Third-party simulator
External synthetic environment
3D Engine...

# Installation

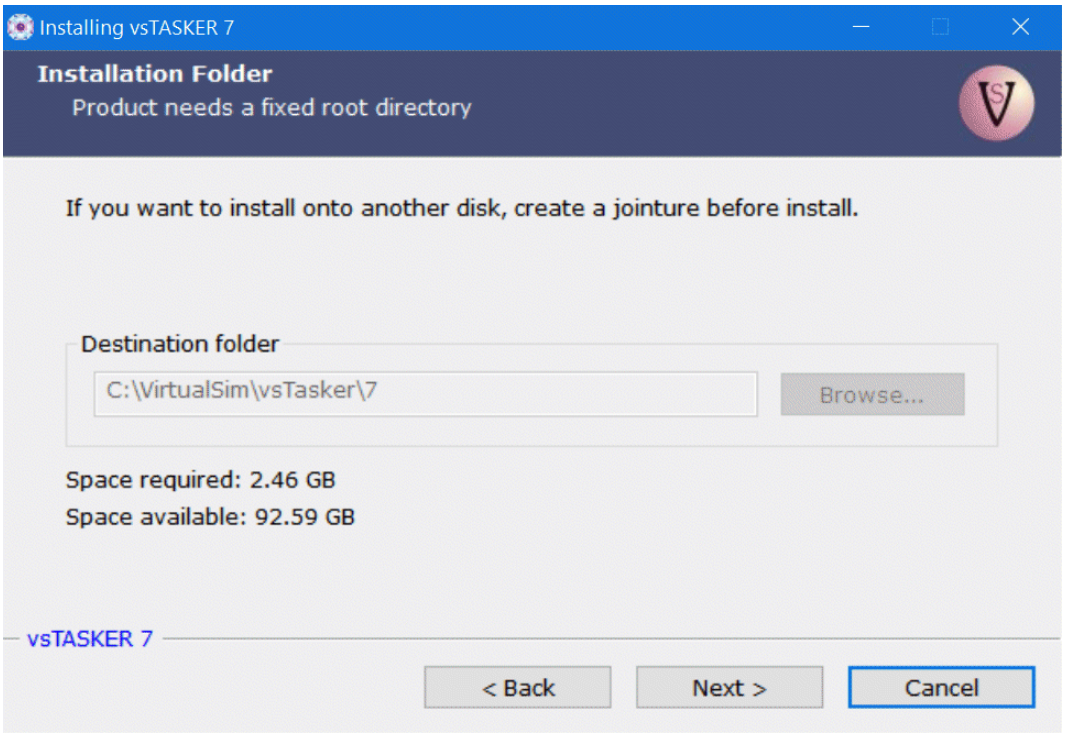
**If you do not want to use the system (C:) partition, see this chapter.**

Since v7, vsTASKER can be uploaded from internet (https://www.virtualsim.com/products/download) or is provided on a USB Key

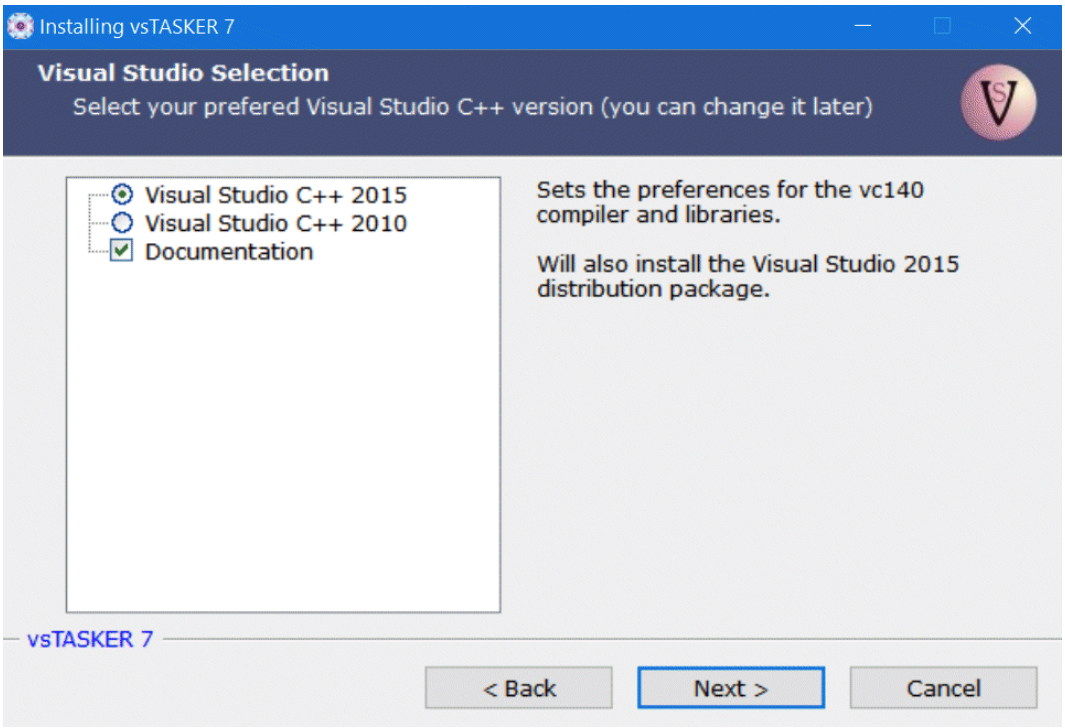

Plug the install dongle and if the installation windows does not appear automatically, open the file explorer and start: vstasker_7..._release.exe



The installation password must have been given to you either by email or is stored in the USB key in the file install_password.txt.
If the password is correct, Next > will be available




Although the software will be installed in C: drive by default, you can relocate it before or after the install by creating a soft link (see here). This method is recommended when the C: drive is an SSD with limited free space.




Select which Visual Studio environment you will be using for development. Although all libraries will be installed anyway, this preset will create the correct Path. You can easily change that later.
If you uncheck the Documentation, help files will not be installed (faster and less heavy). You will have the chance to install them at later time (vstasker_7_documentation.exe) or use the one already put in the Dongle.

The installation dongle contains some nice to see videos you can check during the install.

# Changing Partition

If VirtualSim products has **already** being installed on C: (previous version or current), do:

1. Move the D:/VirtualSim directory to the other partition (ie: D:/VirtualSim)
2. On Windows, select Start -> All Programs -> Accessories -> Command Prompt
3. Right-click on Command Prompt and select Run as Administrator menu to elevate the command
4. On the console type: **mklink /J "D:/VirtualSim" "D:/VirtualSim"** to create a junction (directory link) telling the OS that D:/VirtualSim folder is now equal to D:/VirtualSim
5. You do not need to change anything in the environment variable nor any of your settings.

If VirtualSim products has **not** being installed yet, do the following:

1. Create the VirtualSim directory on the partition you want (ie: D:/VirtualSim)
2. On Windows, select Start -> All Programs -> Accessories -> Command Prompt
3. Right-click on Command Prompt and select Run as Administrator menu to elevate the command
4. On the console type: **mklink /J "D:/VirtualSim" "D:/VirtualSim"** to create a junction (directory link) telling the OS that D:/VirtualSim folder is now equal to D:/VirtualSim
5. Install the product normally.

# Upgrading

## • Upgrading minor version

If you are upgrading inside a version, you can just overwrite your version with the new installation upgrade (backing up is always a good option).
It is important that you do not change solutions, samples or source/header code provided by the release install as these files are subject to change and your modifications might be lost after an upgrade. It is always advisable to duplicate the source code file you want to change and use the duplicated (renamed) one.

If you do not want to do so, lock the modified file (upgrade will fail overwrite) but you might then experience discrepancy problems and this might just be a work around temporary solution.

## • Upgrading major version

*We advise not upgrading to major version big projects or projects that are working fine under their actual version. It is always a good idea to start new projects under the latest version of vsTASKER but to keep the current version if the project is working fine, mostly if you added a lot of things around vsTASKER (components, maps, plugins, etc.) Simple database can be easily upgraded but big projects require a more rigorous approach. Keep a list of all files added to vsTASKER directories in order to upgrade cleanly.*

If you have version 5 (for i.e.) and want to move to version 6, this is a major upgrade.
We suggest you to keep the version 5 as a backup.
Install the version 6 normally.
You will have in D:/VirtualSim/vsTasker:

- 5/
- 6/

Copy from 5/data to 6/data all directories. **Skip all duplicated files**. This way, you will only add to 6/data your databases and user added data.
Do the same with the following directories:

- 5/Feature
- 5/Import
- 5/Model
- 5/Plugins

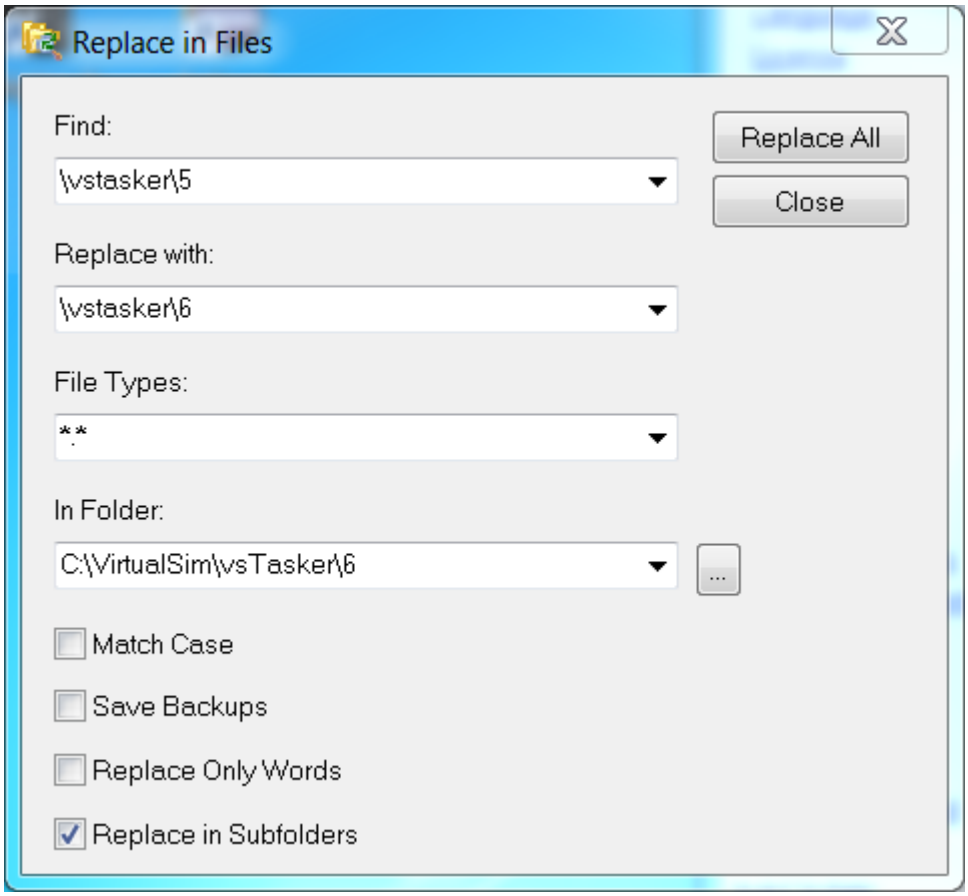
- 5/Validation
- 5/Runtime: copy only the solutions you have changed in 5/Runtime.

*Skip all duplicated files. If later, you find out that some files are missing, you will be able to retrieve them from the 5/ version. If you did modified vsTASKER source code files, a merge will be necessary. There are several tools to do.*

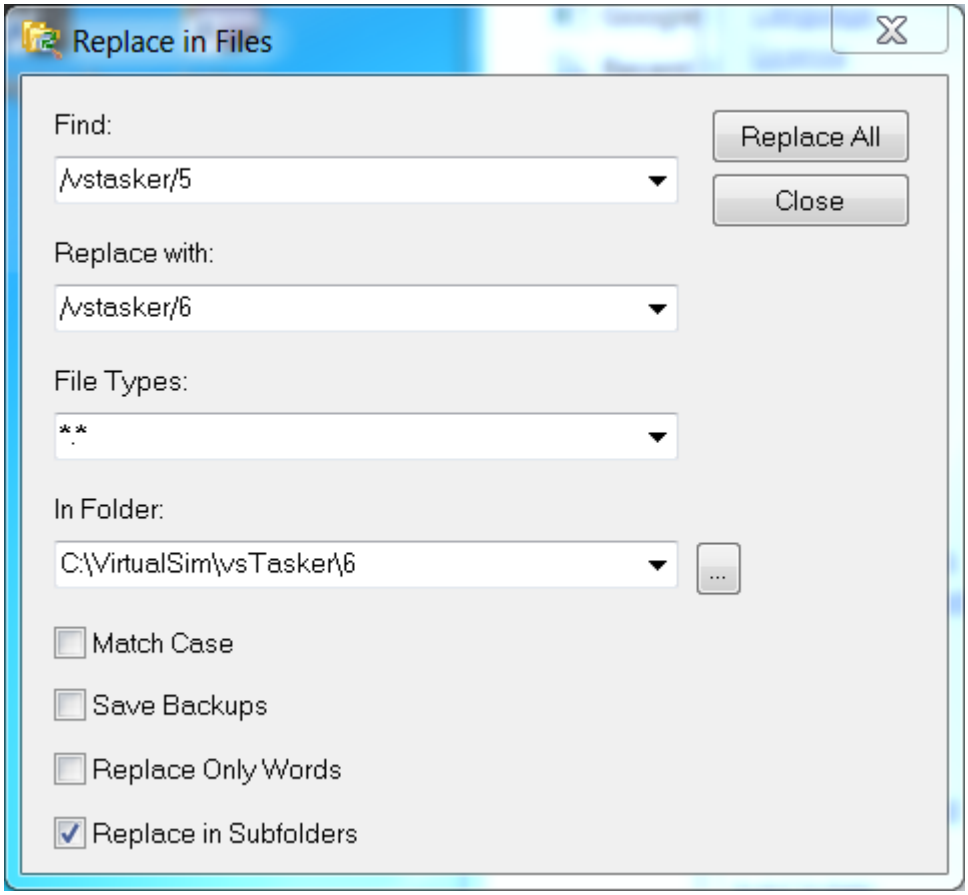
After the copy, it is mandatory to replace string so that references will not link to the previous version.
There is a utility provided into the installation disk that must be used:

ReplaceAll.exe

Start the utility and do the following (make sure you type exactly the same number of characters in Find: and Replace with: Select Subfolders option)




Once the process is done (might take a while - wait that file numbers no more change for 30 seconds), do the following again:



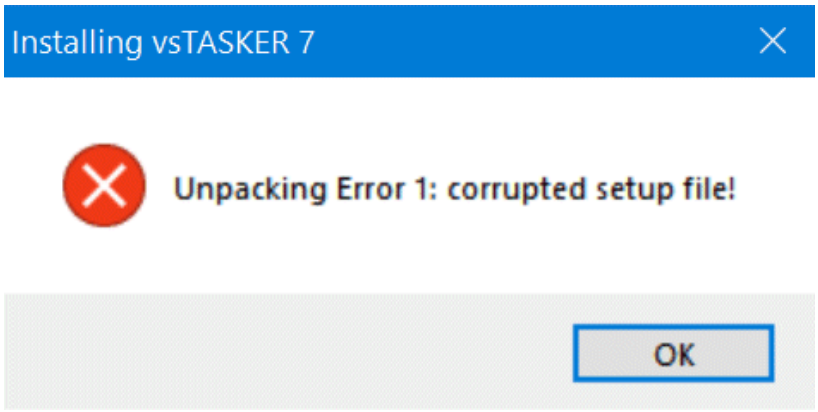

Note that \ has been replaced by / in the two fields Find: and Replace with: !!

You are done. You can now start working with the new version.
Enjoy it !

*If you need to later, copy databases from other older version 5/ or 4/, always do this string replacement on the file or directories you copied.*

# Troubleshooting

## • Unpacking Error 1




This error is typical of an incomplete download.
vsTASKER install comes with two files:
The **installer**, ending with .exe (the one which has been started and gave the above error)
The **data pack**, ending with .pak
You must download both files before starting the install (and wait for all files to be fully downloaded)

## • Upgrade fails at start

If you cannot install a product update, because the installer complains it cannot find the product or the version, maybe the product install did not complete. It looks like under Windows 10, a security level prevents vsTASKER installer to write to registry.

This is the case when the installer hangs at 99% because of the OS preventing modifying the registry (or any anti-virus).
The simple fix is to manually modify the registry (longer fix is to reinstall the software with admin privilege and anti-virus off)

Call **regedit** (Start, cmd regedit)

Open the following sub-keys:

```
HKEY_LOCAL_MACHINE
 ---- SOFTWARE
 -------- VirtualSim
 ------------ vsTASKER
```

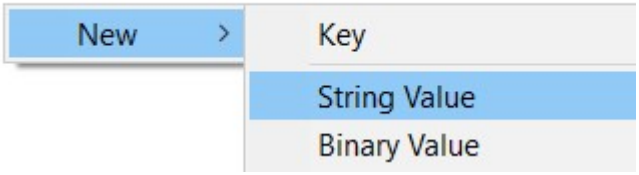
### Troubleshooting

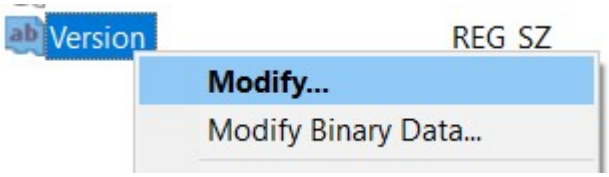
If no **VirtualSim**, add it as a new key.
If no **vsTASKER**, add it as a new key under ***VirtualSim*** one.

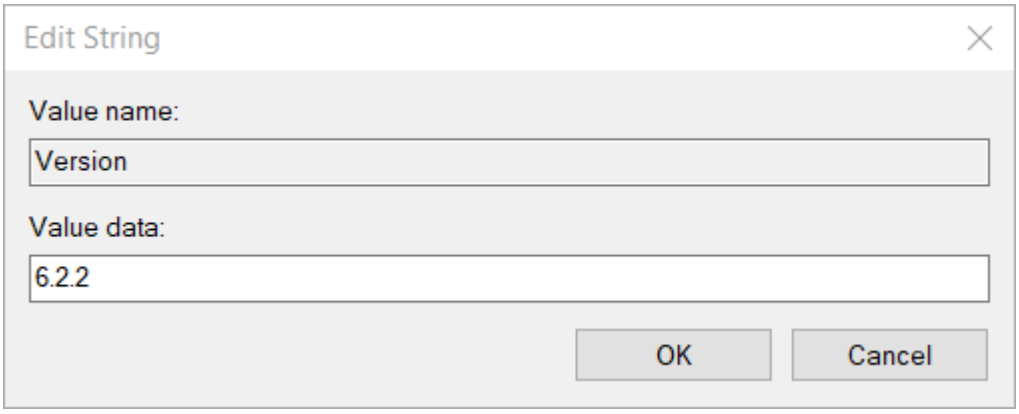
You should have the following datafor vsTASKER key:

```
Directory       REG_SZ       C:\VirtualSim\vsTasker
Uninstaller     REG_SZ       C:\Windows\vsTASKER Uninstaller.exe
Version         REG_SZ       6.0.22
```

if no **Directory** or **Version**, add new string (**Uninstaller** is optional) then double-click on the (new) string to set the value:




Quit ***regedit*** and install the update normally.

# Licensing

vsTASKER offers two ways for licensing.

**USB** dongle that is only provided with permanent licensed product. Dongle can work standalone on one computer or networked. Only standalone dongle is provided by default. Network dongle is for Enterprise version only and must be requested in the purchase order.
**LAN** based keys for temporary evaluation licenses only.


*Since version 5.3, permanent LAN licenses are no more supported.*

Select the following chapters for each installation support.

# Dongle Keys

vsTASKER permanent licenses are protected with USB dongle.

For all version up to v7, KeyLok vendor was used (green key). Users of previous versions and under the maintenance code can obtain an upgrade code which will enable them to use v7 and keep their access to v6 or v5 version (for migration). New users or the ones not under maintenance will need to buy the version 7 and will receive the new dongle (black IOLock)


*Since version v7, **IOLock** vendor has been selected because it does not require any driver to install. It is not backward compatible. IOLock standard dongles do not allow network usage. Ask for a Network model if you intend to use it remote.*

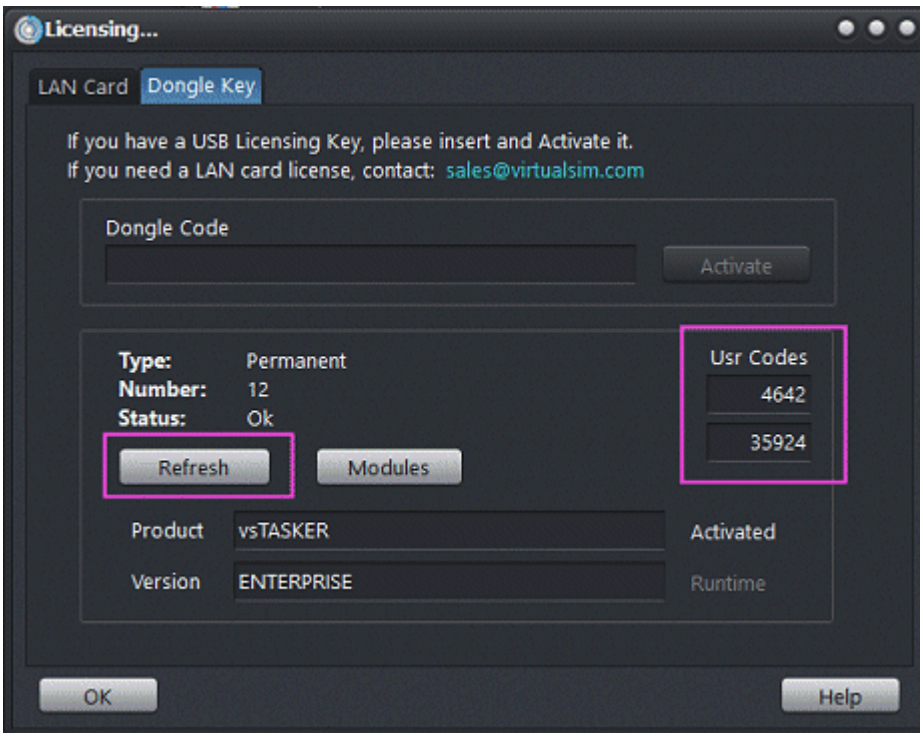
# IOLock

Since v7, a new dongle key is now protecting the software. The main advantage of this one is the absence of a driver. It is plug and play and limits the risk of conflicts.
The difference now is that each dongle comes with a two number user passwords which will reduce the risk of being stolen and reused.

## • Dongle activation

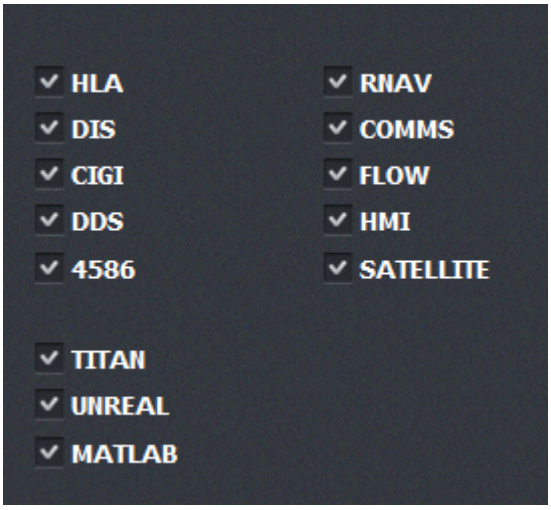
Dongle comes already setup. Once plugged, the 2 user passwords must be entered in the proper keys. They normally have been sent from email or with a separate normal mail.

Start vsTASKER then go to **Help::Licensing::Dongle**, then enter the two passwords and press **Refresh**.



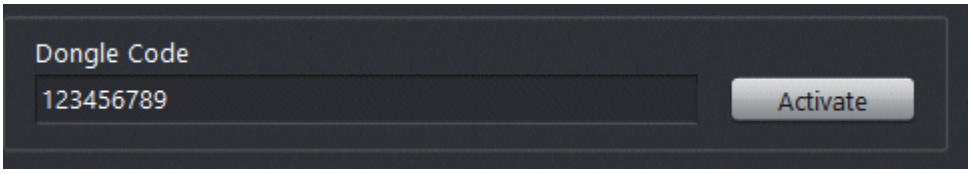

To know which modules are available, click on **Modules**. All available modules are listed with the **check** mark. Grayed out items are not available and can be bought separately.

**IOLock**




## • Dongle Upgrade

When new modules have been bought, VirtualSim will send an activation code to upgrade the dongle with the new modules. Enter the code (number only) into the **Dongle Code** field and press **Activate**. The dongle will be upgraded and you can check that the new Modules have been activated.



# Dongle Key


These dongles are no more provided. Nevertheless, they still can be upgraded to work with v7 and later.


vsTASKER permanent license is most of the time provided by a green USB Dongle Key. The dongle contains the license type and the product code. They are not linked to a particular software package. If you buy 3 vsTASKER, you will receive three dongle keys and will be able to use any of them.

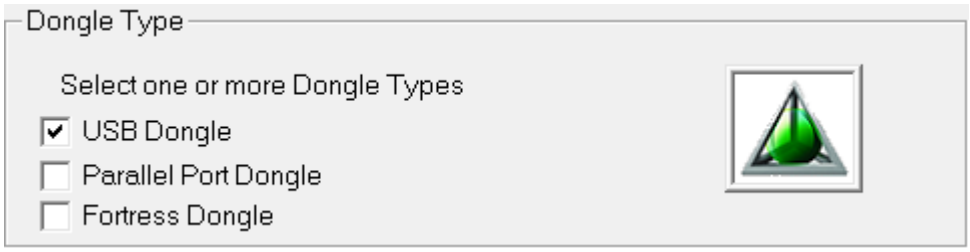
Also, you can install vsTASKER on several computers but only use the one with the dongle attached (the others will not be able to run).

Be careful not to plug the dongle key before installing the KeyLok drivers!

## • Standalone installation

For a Network license, see here.

1 – Install the driver if it is the first time you are using the Dongle on this computer. Installing it twice is not harmful anyway:
Go to: **Start::Programs::vsTASKER::Licensing::Usb Dongle** (or find the install.exe in /tools/UsbLic directory)




2 – Select USB Dongle then:

**KeyLok**

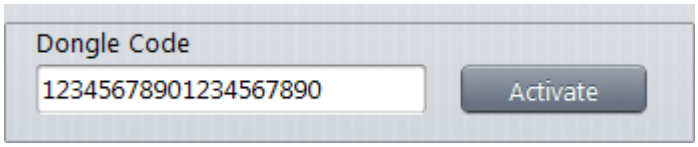

3 – Select Standalone for a single license that runs on the same computer (dongle and vsTASKER GUI altogether).
4 – And finally:


Then only, you will be invited to plug the USB Dongle.

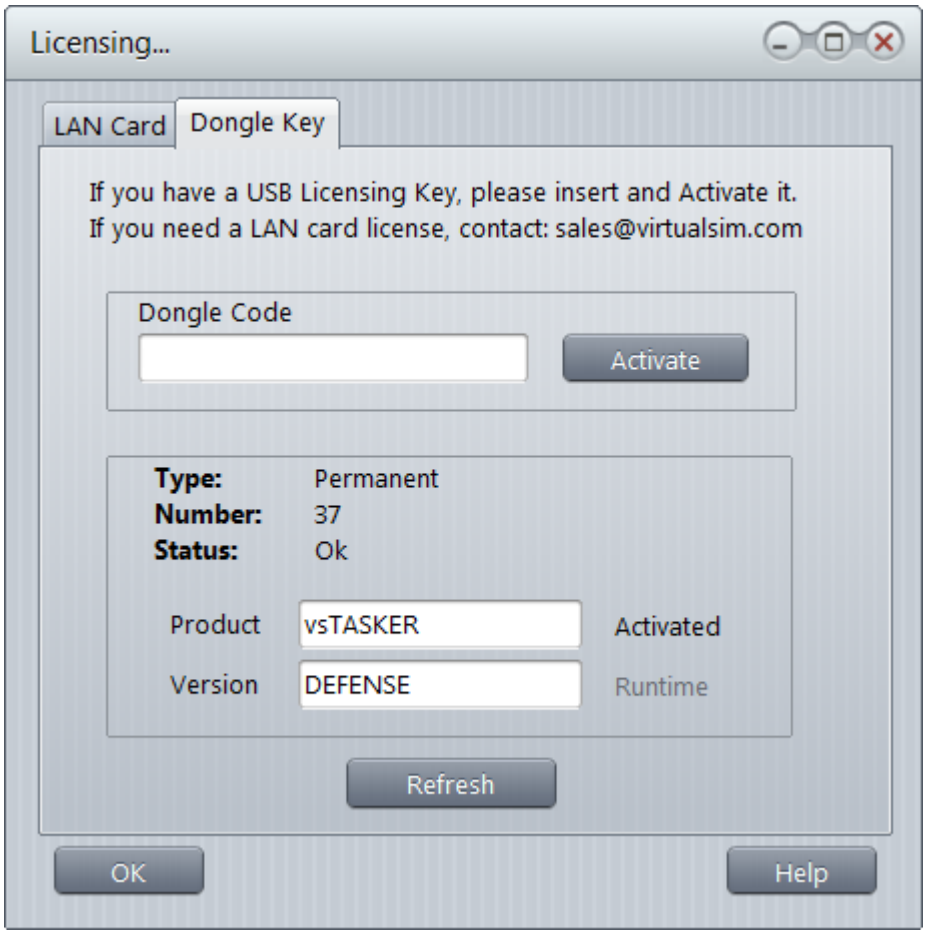
## • Dongle activation

Dongle must be activated before use.
If you did not received a Dongle code with your CD installation package, please ask for an activation code by sending the Dongle License # to : sales@virtualsim.com

Enter the activation code here:




You can check the validity of the Dongle in product **Help::Licensing** window.




# • Trouble-Shooting

1. **Dongle not recognized**

   This is typically the case when driver is missing, not yet installed or conflicting with another one. In such case, remove the driver and reinstall it.

2. **Removing the KeyLock Driver**

   Go to **Start::Programs::vsTASKER::Licensing::Usb Dongle**
   Select: uninstall
   Press: Begin Uninstall
   Reboot then, re-install the driver!

3. **Conflict with Pitch Dongle**

   If you are using Pitch Dongle, please, refer to the following documentation:
   $(VSTASKER_DIR)/tools/usblic/Pitch-Dongle.doc

# Network Licensing

In order to use vsTASKER on a remote computer while leaving the Dongle key connected to one license server computer, it is necessary to follow these steps:

## • Server Side

- Copy the /Tools/UsbLic directory to the server computer (the one holding the dongle key)
- Launch install.exe :


- Select USB Dongle, then:


Select Server and finally:


- You can then plug the dongle and let it connected (check that the driver installation by Windows succeed)

## • Client Side

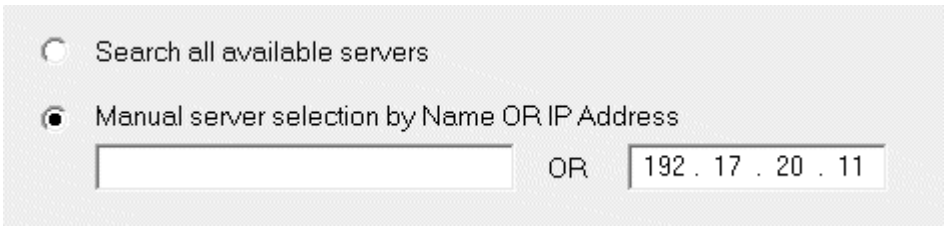
Launch /Tools/UsbLic/install.exe:


- Select USB Dongle, then:


- Select Client
- Then select the Server computer instead of letting the driver query the Network. If the dongle

is going to remain on the same server, doing so will make the connection faster.


- Give the IP of the server computer (check that the firewall is not blocking all access), then OK




- Finally:


-

Test the connection to the server using the following tool: /Tools/UsbLic/Troubleshooting/NetKeyMonitor.exe to see which servers are running and how many users they have.

## • Troubleshooting

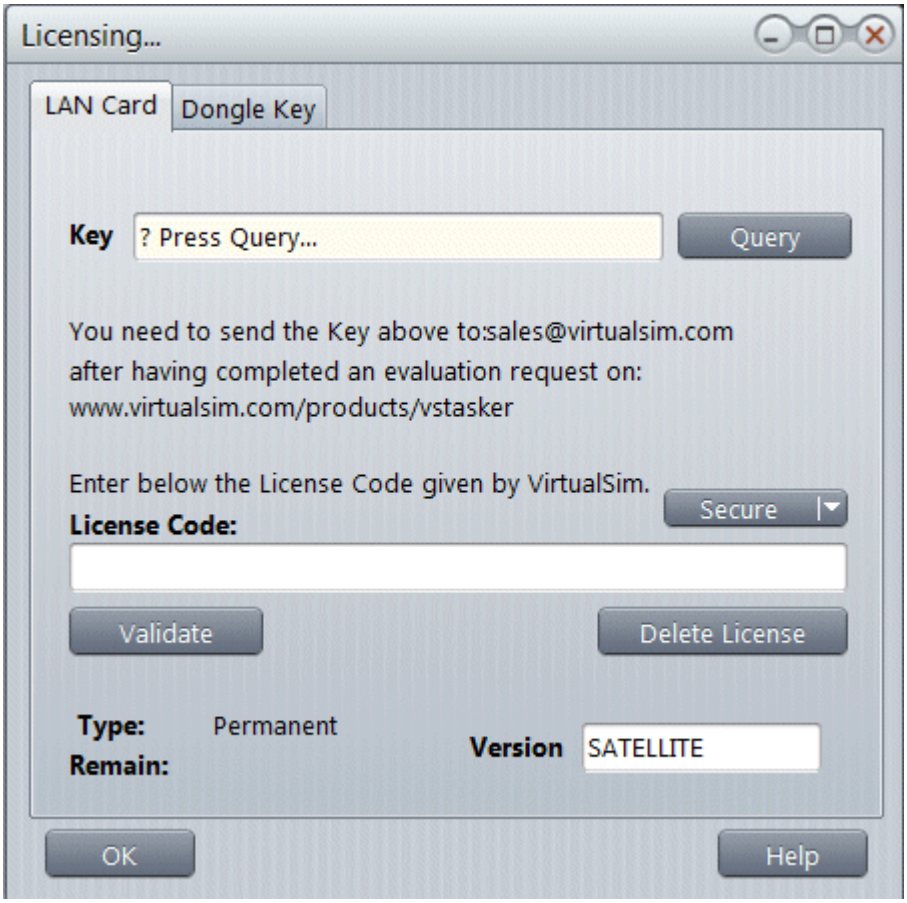
- If the number of sessions if reached (simultaneous users), the license cannot be acquired.
- You might need on the server to put the process klserver.exe into the firewall exception
- The default timeout period for a session is 540 minutes. To change that to 1 or 2 minutes, write the file windows/system32/KLTCPIP.DAT and put 1 or 2 inside (values in minutes)
- In the client computer, you can write the file windows/system32/TCPIPSVR.DAT with the IP address of the server computer to use (i.e: 192.17.20.11)

# MAC Address

To obtain and validate a temporary license for vsTASKER, you must open the License Manager.

*Disabling the wifi card is only necessary for versions earlier than 5.3.20*
*See here how to temporarily disable the wifi card.*

Run vsTASKER as **Administrator** to authorize Registry access and guarantee the proper installation of the license.




Then, in the “About…” window that pops up, press the ***Query*** button.

Send to sales@virtualsim.com the value that appears in the ***Key*** field. Mention in the mail your Name, Surname and Company name.

VirtualSim will provide to a Temporary (or Permanent) License Code. Copy-Paste it into the License field of your license manager.

Then press button Validate.

**MAC Address**

After 2-3 seconds, Type and Validity fields should display correct values.

*If you have disabled your wifi card, you can now enable it again and keep it that way.*

If Type field displays Invalid, please contact VirtualSim again to get another License code. Version should specify you the licensing version covered by the license. Refer to the comparative chart between Basic and Military version to see the difference in features.

If you have bought DEFENSE version and receive a BASIC activated license, please contact VirtualSim support team for a change in the code. Same process applies for an update.

## • Trouble-Shooting

If the License code is rejected of does not work, check if you have the environment variable set (VSTASKER_DIR) and pointing to the right directory. Try then to
reboot.

## • License Transfer

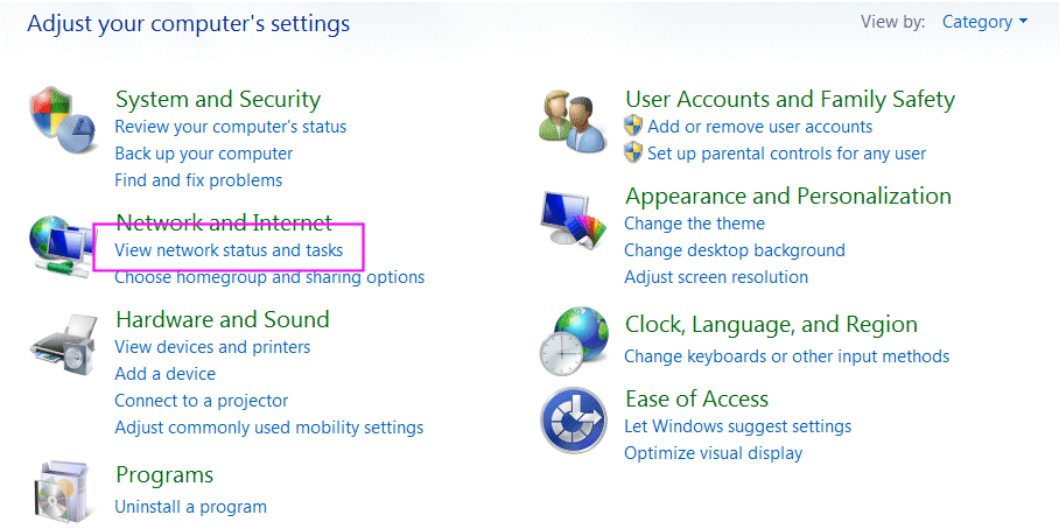
If the computer must be changed because of a breakdown or if the LAN card must be replaced, a License Transfer must be requested.
Follow the same procedure as above.

For each License Transfer, a **Form** must be filled. This form will be sent to you on request. You will receive the new License Key once the Form is returned by fax, completed and signed.

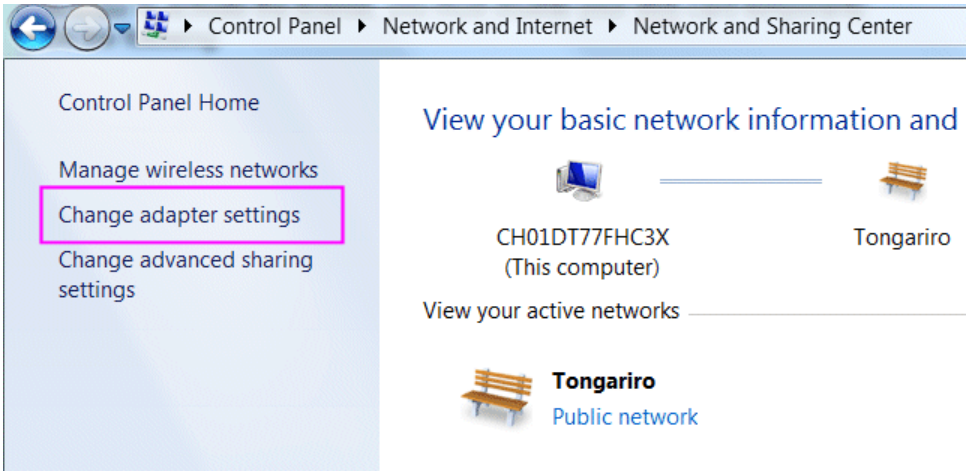
# Disabling Wifi Card

In order to activate the license for version of vsTASKER older than 5.4, you must disable your wifi card temporarily.
Do the following:

Call the Configuration Panel, then, select **View network status and tasks**

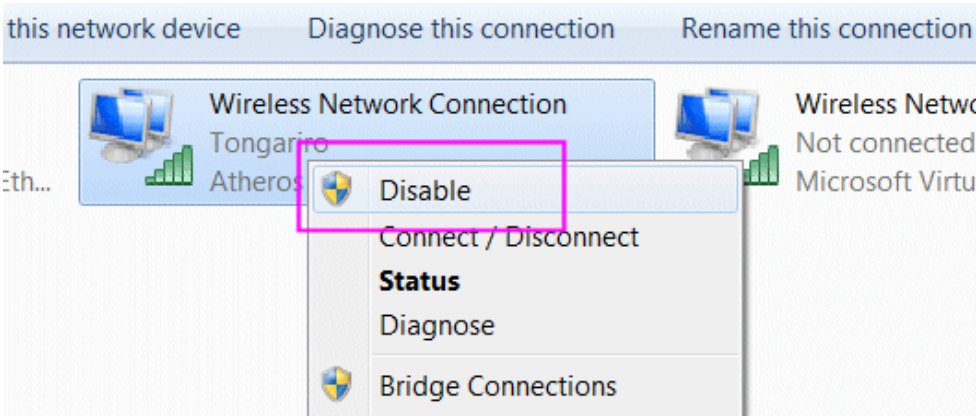



then **Change adapter settings**

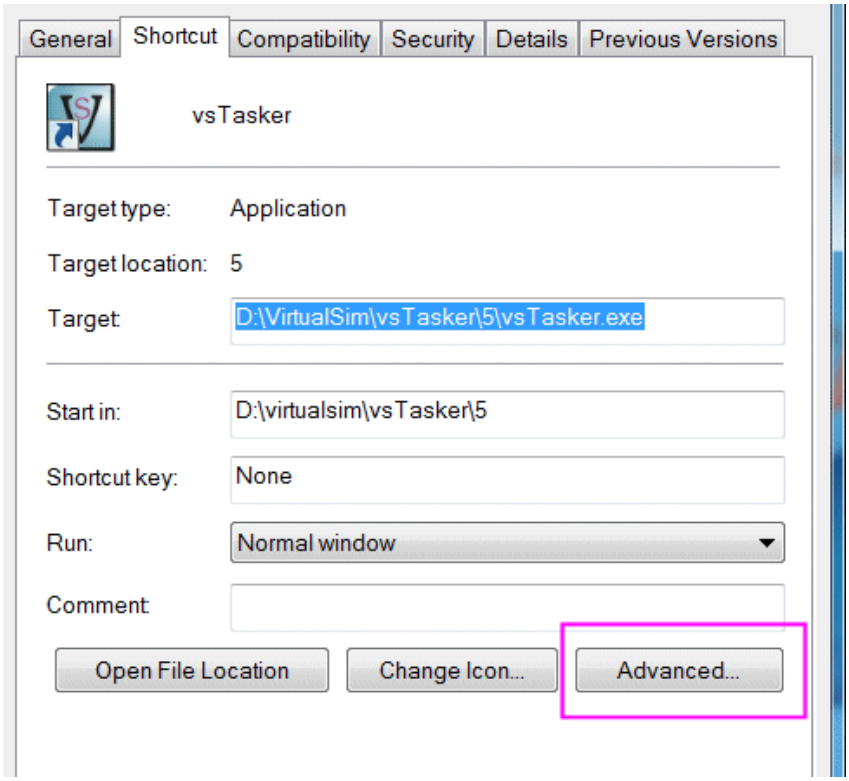


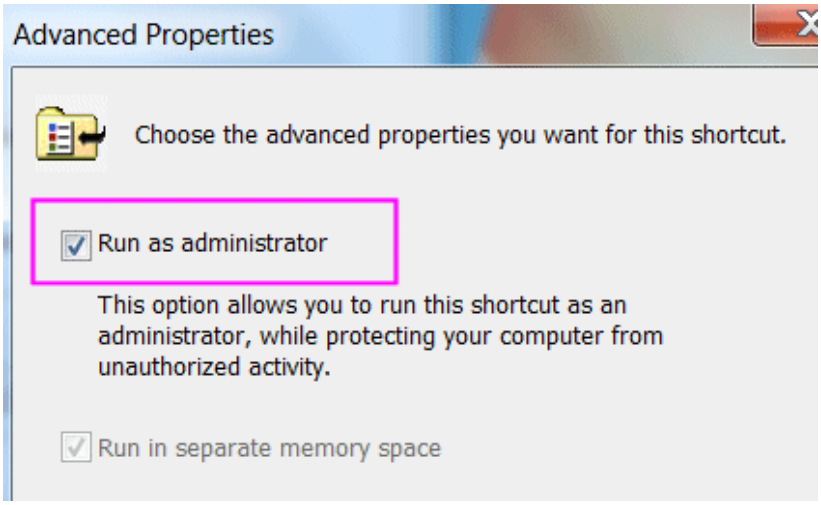

finally, right-click on the Wifi card to disable it.

### Disabling Wifi Card

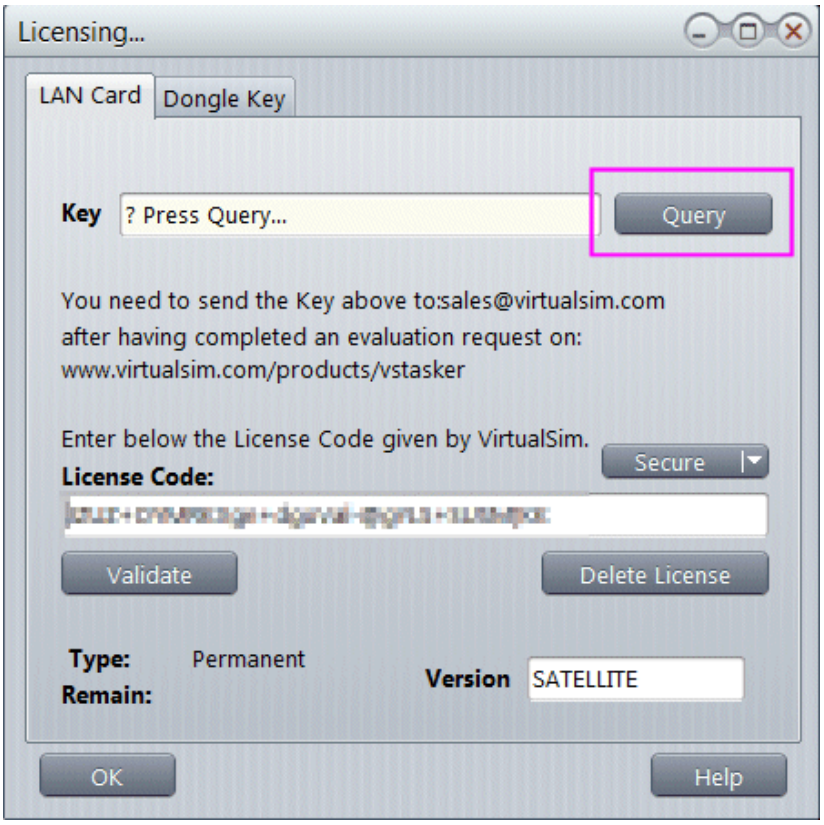



Then, start vsTASKER from the icon with **Run as Administrator** mode enabled, to do so, right click on the vsTASKER icon of the Desktop, select Properties then Advanced and select the mode





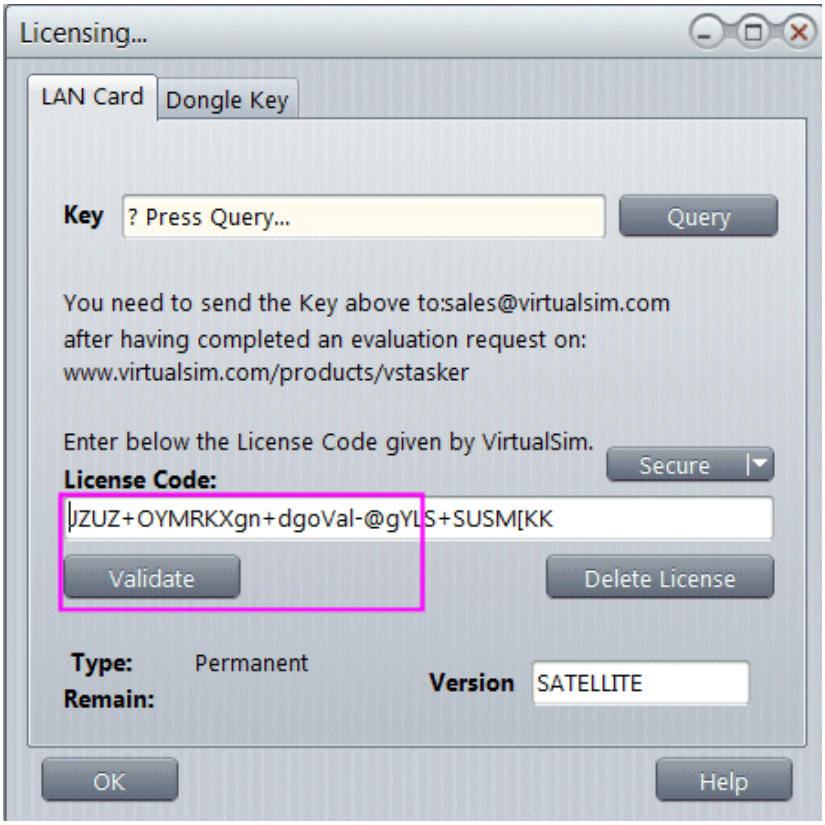

Once in vsTASKER, select Help, Licensing, LAN Card and press Query.




Send the Key value (copy-paste) to VirtualSim

### Disabling Wifi Card

When you receive the License Code, enter it at the proper location on the same window, then Activate it (if you have enabled again the Wifi card to send the email, make sure it is disabled when you activate the license Code!)




Quit vsTASKER.

**Enable back the WIFI** card the same way as you disabled it (see above).
You must enable the card only after having entered the license code.

If a problem persists, please contact VirtualSim support.

# Version 5

Users of old version of vsTASKER can ask for an upgrade, only if they are under a **valid Maintenance plan**. They will receive a new activation code for their dongles. Once the dongle activated for the new release, it will not support the old product, although reverting the activation with the prior code will enable it back. User will then have
the capability to switch from one version to the other alternatively using the same dongle.

Without a Maintenance plan active, upgrade from version 4 to version 5 will be done at upgrade price (**reduction percentage** from normal list price with a new dongle, meaning the older version will be kept by the user). Please, contact your distributor for pricing (vstasker.com, at distributors). If you bought the vsTASKER directly from VirtualSim, contact VirtualSim sales.

Version 5.3 including Satellite module does not upgrade automatically from version 5.2.
Only customers under a maintenance plan will be offered the upgrade at no cost.

Customers without a maintenance plan will be able to upgrade to 5.3 and above with a significant rebate.
They will also receive a new dongle key activation code.
Once upgraded, 5.3 and above dongle will also work with 5.2 version.

**Version 5.2** will be supported until July 2015.
**Version 5.3** will be released January 2015.
**Version 5.4** will be released October 2015.

Early buyers of 5.4 (beta version) will automatically be upgraded to release version at no cost.


*Version 5 is no more supported. We encourage all customers to migrate to version 6 or 7.*

# Version 6

Version 6 has introduced numerous bug fixes.
This version has been fire proofed under several industrial projects which brought many enhancements in all components of the product.

Amongst the main new features:

- CIGI
- Web Map service
- SVG import for VBS scenarios
- VBS ASI demos and communication mechanism
- Undelete capability
- Import/Export of HMI windows including merging
- User data log for record & replay
- ARINC 424 database import revisited

**Version 6.0.1** released in January 2016
**Version 6.1.1** released in April 2017 (will be last built)

Upgrade from version 5 is only available to customers on maintenance plan.


*Version 6 will be supported until end of 2021 and for customers on maintenance only.*

# Version 7

Version 7 has been developed entirely under and for Windows 10 (the skinning will be optional)
It will introduced numerous new features to match the demand of the industry in the Defense simulation field.

- Version built on Windows 10 (v7.0)
- OpenSceneGraph 3.4.0 support (v7.0)
- SVG Map with UTM and Lat/Lon projection (v7.0)
- Visual Studio 2010 & 2015 support (drop of 2008 and 2012) (v7.0)
- x64 libraries for vc100, vc140 and vc141 (v7.0)
- Dynamic User-Interface (v7.0)
- Folders to simplify and organize visual objects (v7.0)
- Code Synchronization (VisualStudio -> GUI) (v7.0)
- CIGI with VBS-IG support and demos (v7.0)
- Titan Vanguard support and demos (v7.0)
- Dark Mode (Sprites & Features excluded)
- ADS-B ready (OpenSky-Networks & Aviation-Edge) (v7.0)
- 1440p high DPI screen resolution compatibility
- Raster map free rotation
- Add arc-circle leg definition for Roads
- MQTT support (based on Mosquitto) (v7.1)
- DIS visual interface + components (based on OpenDIS) (v7.1)
- Rail support and component (v7.1)

- Network Simulation with channels communication (v7.1)
- DDS support (based on OpenDDS) (v7.2)
- Unreal Integration with proprietary protocol (v7.2)
- Radio Network based on channels (v7.2)
- wxWidget integration demo + tutorial (v7.2)
- Navigation Mesh for any entity(v7.2)
- RCS use for Radar simulation (v7.3)
- Scenario TimeLine (v7.3)
- OsgEarth 2.10 with Orbits and Satellites (v7.4)
- Behavior Trees (v7.5)
- OpenIG Plugin (v7.6)
- Multiple Plans for Entities (v7.7)
- JSBsim Flight Model integration (v7.7)
- Runtime Behaviors (v7.7)
- Private/Protected Logics, Knowledge and Routines (v7.7)
- Technical cruise with tickets process (v7.8)

**Version 7.0** was released late 2019
***Version 7.1*** will be released end of 2021

## • Module licensing

A lot of changes have been introduced in this version, including the module licensing. In version 7, users need to select which modules they want instead of getting a version with full capabilities. With the principle of modules, price can be lower for basic product use when specialized functionalities are not required.

v7 is derived into four product versions:

- **Demo**
- **Player**
- **Academic**
- **Enterprise**

With the following optional modules:

- LAN
- HLA
- DIS
- CIGI
- DDS
- MQTT
- HMI
- MATLAB
- COMMS
- SATELLITE
- RNAV
- FLOW
- TITAN
- UNREAL

Although they all come with the product, activation of modules can only be done through special codes obtained upon purchase. Contact VirtualSim for more details.
**Upgrade from version 6 will only be available for customers on maintenance plan**.

*A major version upgrade or migration is not advised in a middle of a project or for a project which is running fine. The migration process must be undertake with serious concerns. It is safer to start a new project with the latest version to make sure it will be supported later.*

# Migration from v6

Upgrading from v6 to v7 is a version migration and even if the new code tried to be as much as possible backward compatible, several changes must be done manually in order for the upgraded database to work under the new version.

Although we recommend for sensitive big project to stay with the major version they were developed with, some maintenance policies encourage the use of the latest version. We will list here the main upgrade process we encountered from migrating v6 projects to v7.

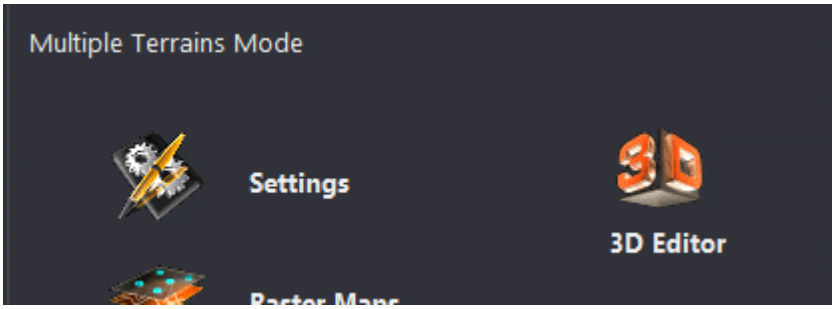

*The v6 green (KeyLoK) dongle, once upgraded to v7, will still be able to work with v6 versions, allowing the users to keep both versions running. With the new ioLock dongle, the v6 version is not accessible, but as this new dongle is only sent to new v7 users, the need of a v6 compatibility is void.*

## • Scenarios with OSG terrains

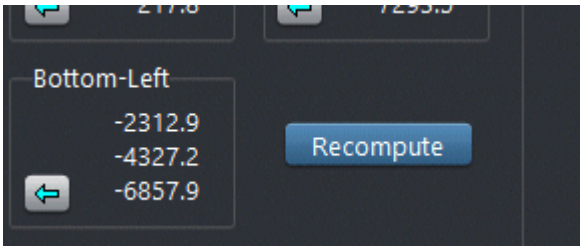
### • Entities no more on terrain:

The OSG terrain locator needs now (and most of the time) to be centered according to the OSG terrain center and not the MAP center. Sometimes, both do match but if it does not, do the following process to match the MAP entity position and the corresponding OSG terrain position:

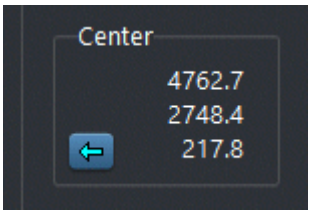
First, open the **Terrain 3D Property** window. Select terrain on the **Environment** list and click on 3**D Editor** icon




Then, click on **Recompute**. A little window will appear on the top left desktop and will load the OSG terrain, then the window will fill the Center, Bottom-Left and Top-Right parameters, then close.




Now, press the **left arrow button** to copy the Center values to the Terrain Offset values. You can cross check the result with the **3D Editor**. If it does not match, try to use the Terrain **QuickSetting** or change the **Projection** type until you have a correct match.

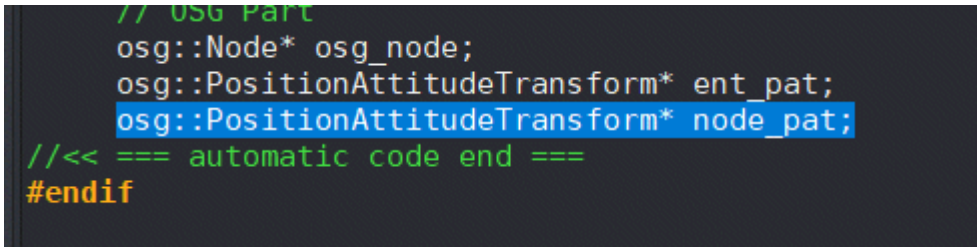



- **Compilation error: node_pat not found:**

For this error, open the Entity Class


And in Declaration panel, add the following line:

```
osg::PositionAttitudeTransform* node_pat;
```



## • setEvent & setFact

Before, any integer could be used to mark the data pointer. Now, the `UDataType` enum must be used instead. See its definition in vtypes.h

## • Scenarios with RNAV

**unresolved external symbol "public: virtual void __thiscall HiAltCtrl::setUserCode(void)**

To fix this, add in any **Model** Container the DataModel: **Hi_AltCtrl** (inside *Controllers*), then recompile

## • Scenarios with Motion{Goto|Slide|Follow}

The API `move()` for all these components have changed.

**Before:**

```
MotionGoto* mg = E:findMotionGoto();
mg->move(10); // m/s
```

**Now:**

```
MotionGoto* mg = E:findMotionGoto();
mg->setSpeed(10, _m_s);
mg->move();
```

# Directories


## 1 Bin


Here as stored some DLL needed by vsTASKER GUI and some generated simulation engines, according to the libraries used.
You will also find the vst2008_pro.msi and vst2010_pro.msi redistributable package normally called at installation time. They are requested if you do not have Visual Studio installed.

# 2 Data


This directory contains all the databases needed by vsTASKER for design and runtime.

| | |
|---|---|
| 3DS | Contains some 3DS Max models to use for OpenGL perspective mode or OSG viewer |
| 4586 | Editions 2 and 3 of the STANAG 4586 definitions |
| ArcInfo | vsTerrainBuilder converted .geo terrain data from .shp files |
| Coast | vsTerrainBuilder converted coast line terrain data from .dat or .lin files |
| Db | Contains all design database vsTASKER loads at design time. The database node file .db is inside a directory holding the same name. |
| Elev | vsTerrainBuilder converted elevation terrain data from .dem or .dted files |
| Fom | Contains HLS federation files in 1.3 (.fed) or 1516 (.xml) format needed by the HLA module |
| HMI | Contains all textures needed by the Sprites. Files are going by pair. One is the .tga with alpha-channel for transparency. The second one is the .bmp for display in the File selection window (as .tga cannot appear as bitmap) |
| Log | Directory used to store Record & Playback files from the Logger |
| Map | Contains all the image terrain files used for Raster maps and the tiled-map databases (x.map file and associated x directory) |
| NavChart | vsTerrainBuilder converted s57 .000 maritime files |

| | |
|---|---|
| Omt | Contains the OMT definitions used by the HLA module, for 1.3 federations mainly. |
| OpenDrive | To come... |
| OpenFlt | vsTerrainBuilder converted .opf terrain data from OpenFlight .flt files |
| OsgFrame | vsTerrainBuilder converted .osg terrain data from OpenSceneGraph. Only vertex are imported. |
| Rep | Contains all Repositories created by the user. A Repository is made by a file.rep and its corresponding directory holding the same name. |
| Resource | Contains all the different resources used by the GUI. Some files are editable manually (using notepad). 2525b and Entity symbols are also stored here. |
| RNav | Contains the translated ARINC 424 databases to be used by |
| Satellites | Norad satellite TLE files |
| Samples | Get all loadable samples organized by types. Any loaded sample must be copied into the Db directory in order to be modified. |
| Shared | Contains all Exported objects. |
| Snap | Contains snapshot & restore runtime data, stamped by scenario name and time. |
| Sprites | *Not used at the moment* |
| Template | Contains all database templates, built-in and user defined. |
| WFrame | vsTerrainBuilder converted .wrf wire frame terrain data. Each file is associated with a texture that is mapped to the wire frame. |

## 3 Doc


This directory contains some documentations, like the licensing instruction manual.
The release_notes HTML file lists all updates added to the previous versions.

## 4 Feature


Contains all scenarios Features available. User can modify any of them, add his own using the template provided or rebuild the corresponding .lib (need of Visual Studio) and .dll (need of C++ Builder 2010 or higher) to resolve some problems that might occur on some systems at dll load time.
features.lst lists all built-in Features. features_user.lst lists all user added Features (if any).

## 5 Gen


Contains all generated code files.
A database named foo will produce three files:
foo.h
foo_intf.cpp
foo_code.cpp
Because of the first file, it is for now advisable to mind the name given to the database (a database named windows for example will conflict the system windows.h header file).
db_config.h/cpp contains the latest generated header file.

## 6 Gui


Contains some samples on how to create a custom GUI replacing vsTASKER GUI for runtime environment.
chinese and simple are two basic applications written in Visual C++ to load a scenario, control some entities and run it, from inside a specific panel and using some vsTASKER dll map plugins (see below).
QT is a simple GUI developed under QT.

## 7 Help


Contains the vsTASKER User Manual (this document, that also includes the online help), the Reference Manual (automatically generated using Doxygen), the TerrainBuilder User Manual and the deprecated Tutorial document.

## 8 Import


Contains the Importer and Exporter dll, like the STK Importer/Exporter module.
User can create his own Importer/Exporter module using the provided samples.
import.lst lists all build-in importers.
import_user.lst lists all user-added importers. Use this file to add/remove your own importers.

See the documentation regarding expending the Import/Export functionalities


*Built-in lists must not be modified as each new release/update will replace them with a new version.*

## 9 Include


Needed to compile simulation engines and to build custom GUI.
See Utilities for a list of convenient API and Classes.

## 10 Lib

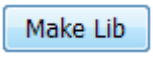

Needed to link simulation engines against vsTASKER libraries and to build custom GUI.
Note that C++ Builder libraries are listed in /lib while Visual C++ libraries are listed in /vc90 (for Visual 2008) and /vc100 (for Visual 2010)

## 11 Linux


*Contains the Linux ANSI compatible source code to create Ubuntu libraries.*


*Linux integration is not a plug & play process. VirtualSim engineer must be part of the process.*

# 12 Model


Contains all Components and DataModels used as built-in items in vsTASKER GUI and linked against the simulation engine.
The user can add his own Components and DataModels in specific directories (to avoid mixing built-in with his own) either by using the template_cpt.h/cpp, template_dml.h/cpp (see src/user and include/user) or by using the 

button of the Component or DataModel object. See Models.

**built-in items:**
default_cp.lst lists all default components
military_cp.lst lists all defense related components
vrvantage_cp.lst lists some vrVantage components used to integrate vsTASKER Sim engine with the MAK 3D Viewer (use them as an example)
satellite_cp.lst lists all components used with the Satellite version, like orbit propagators
stk_cp.lst lists all STK components (used with STK software or to replace some STK capabilities, after scenario import)

default_dm.lst lists all default built-in data-models
military_dm.lst lists all military data-models, mostly used by military components

**User items**:
user_cp.lst template for user components

user_dm.lst template for user data-models


*Built-in lists must not be modified as each new release/update will replace them with a new version.*

# 13 Plugins


Contains all Scenario map layers used by vsTASKER GUI or any custom GUI to draw OpenGL terrain data.
User can modify any of them or create its own by using the empty Extra plugin.
Source code files and makefiles (for Visual Studio 2010 and 2017) are available for each.
See Plugins.

| | |
|---|---|
| ArcInfo | Draw converted arc-info data on the terrain map |
| Coast | Draw converted coastlines data on the terrain map |
| DrawFont | Display Chinese characters on OpenGL map |
| Editor3D | Draw the scenario area in the 3D Editor panel, using Delta3D or OSG libraries. |
| Elev | Draw the elevation data on the terrain map |
| Entity | Draw the entity symbols. Modify this plugin if you want to draw your own symbol. |
| Extra | User plugin template for any use. |
| Flight-Plan | Draw all Flight-Plans on the GUI map |
| Gfx | Draw all runtime graphics sent and updated to the GUI by the Simulation Engine. |
| Grid | Display the grid of the terrain |
| Make | Rebuild all plugin DLL for Visual Studio 2010 and 2017 |
| Mesh | Draw all the Meshes defined in the Scenario Meshes pane |
| NavChart | Draw converted S57 maritime charts |
| OpenFlt | Draw converted Open-Flight data on the terrain |
| Orbit | Draw orbits in terrain and globe modes |

| | |
|---|---|
| Overlay | Draw all Overlay windows like the Vertical View |
| Plan | Draw the entity plan (if any) with all localized routines |
| Raster | Display the raster maps on the terrain area |
| RNav | Draw all ARINC 424 air navigation database |
| Road | Draw all roads defined in the Scenario Roads pane |
| Runtime | Build and returns the list to be displayed in the runtime tree-list environment |
| Scenario | Draw all scenario level data like LOS, distance string, coordinates labels, etc. |
| Wind | Draw all wind tubes and area in all terrain modes |
| Wireframe | Draw the converted polygonal wire frame of the terrain data |

## 14 Runtime


Contains directories per Viewer or third party software.
It is not mandatory for a user to keep these directories for his own database development but all vsTASKER samples are using them.
So, in each directory, you will find the corresponding vc10 (Visual Studio 2010) or vc14 (Visual Studio 2017/19) .sln project file to debug the generated simulation engine. Each project has already libraries and include path set, ready to compile, link and run.

If you need to create a specific target environment for your simulation engines, it is advisable to create your own directory under Runtime, set the path in Database Runtime setup and include there the Visual Studio project file and relocate here all models or directory the environment will need at runtime.

## 15 Snapshot

Snapshots

Contains some promotional GUI screen shots.
Can be deleted after install to gain some space.

## 16 Sprite

Sprite

Contains all Sprites available to build HMI. User can modify any of them and rebuild the corresponding .lib (need of Visual Studio) and .dll (need of C++ Builder 2010 or higher).
sprites.lst lists all built-in Sprites.
sprites_user.lst lists all user-added Sprites. Use this file to add/remove your own Sprites (you will need C++ Builder 2009 or higher)


*Built-in lists must not be modified as each new release/update will replace them with a new version.*

## 17 Src

Src

Contains the HLA ambassador class file needed to link with the respective RTI (1.3, 1516 or Evolved)
Template directory has some template.cpp files needed as `#include` if used by the code.
OSG directory contains the layer vsTASKER is adding to provide some visual effect for the user (to be used from logics). SilverLining and Triton source files are also provided.
4586 directory has some glue modules for Edition 2 or Edition 3. These files are automatically built.

## 18 Tools

Tools

Directory used for the vsTerrainBuilder application and the USB key dongle.
Data directory contains some data samples to be loaded by vsTerrainBuilder for conversion to vsTASKER terrain map format.
UsbLic directory contains the install.exe for KeyLok dongle driver installation/ removal, plus some documentation. See Licensing.

# 19 Validation


Contains some subroutines that are attached to a specific Dyn-UI and are called every time a change is made by the user on any of the Dyn-UI variables.
validation.lst lists all built-in validation functions.
validation_user.lst lists all user-added validation functions. Use this file to add/remove your own validation.


*Built-in lists must not be modified as each new release/update will replace them with a new version.*

# Getting Started

## • Main Concept

vsTASKER is divided into two parts:

- The GUI that allows user to define visually the entire simulation, from behavior definitions to scenario edition;
- The Simulation Engine that loads and run the scenario.

Both parts are connected using Shared Memory or LAN.

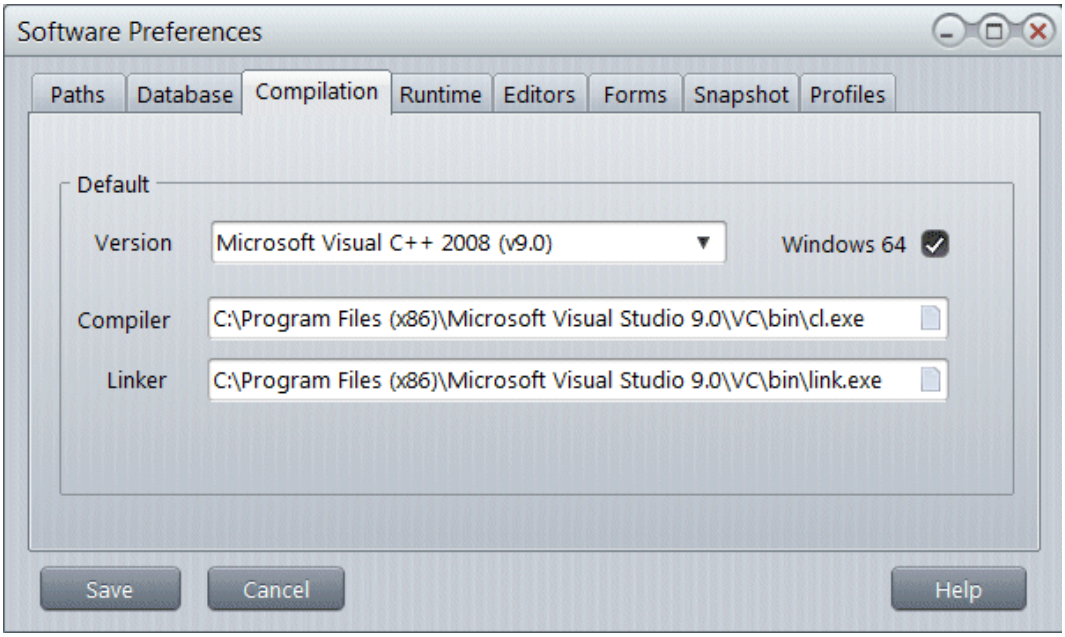
## • C++ Code Generator

vsTASKER needs a compiler like Microsoft Visual C++ in order to build simulation engines. Without MS Studio, you will only be able to load and run already compiled sample databases (*.rt) that are provided with their associated sim-engine.

If you have Studio (2008 or 2010) installed, check that the configuration is correct. Go to Tools::Options, select Compilation panel and chose the version you have installed. You must change the location if your compiler is installed on a different directory.

Compiler needs “cl.exe” and Linker needs “link.exe”.

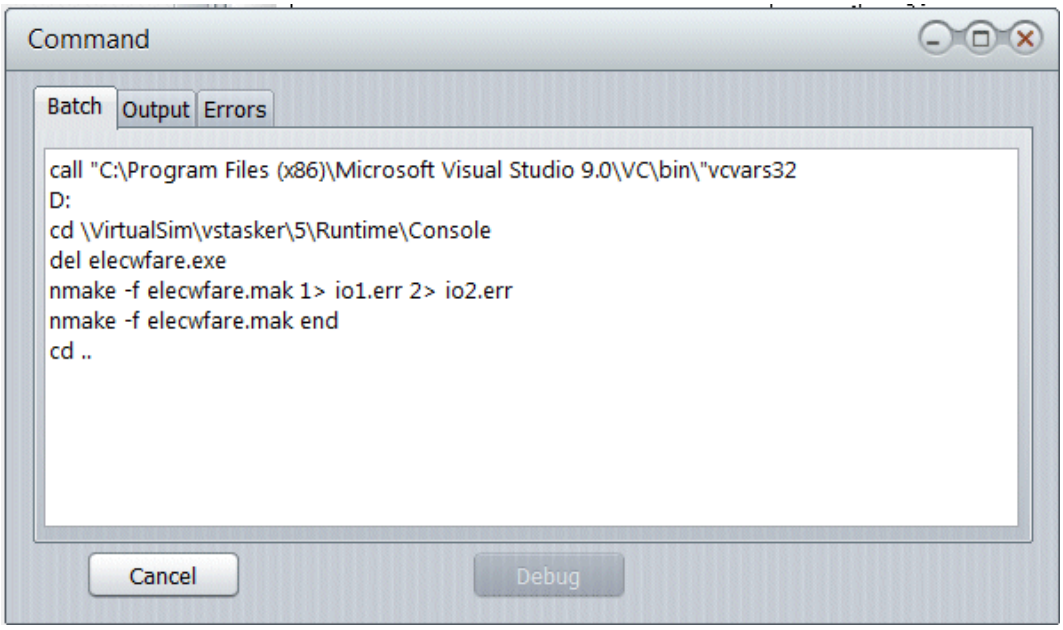
Select **Tools::Preferences::Environment**



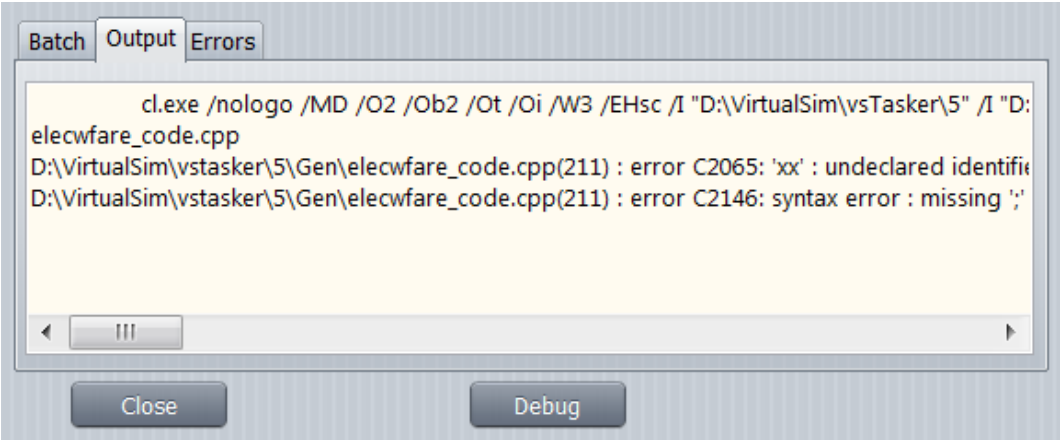

*Microsoft Visual C++ 2003 (vc71) 2005 (vc80) and 2009 (vc90) are no more supported since v7*

## • Compilation

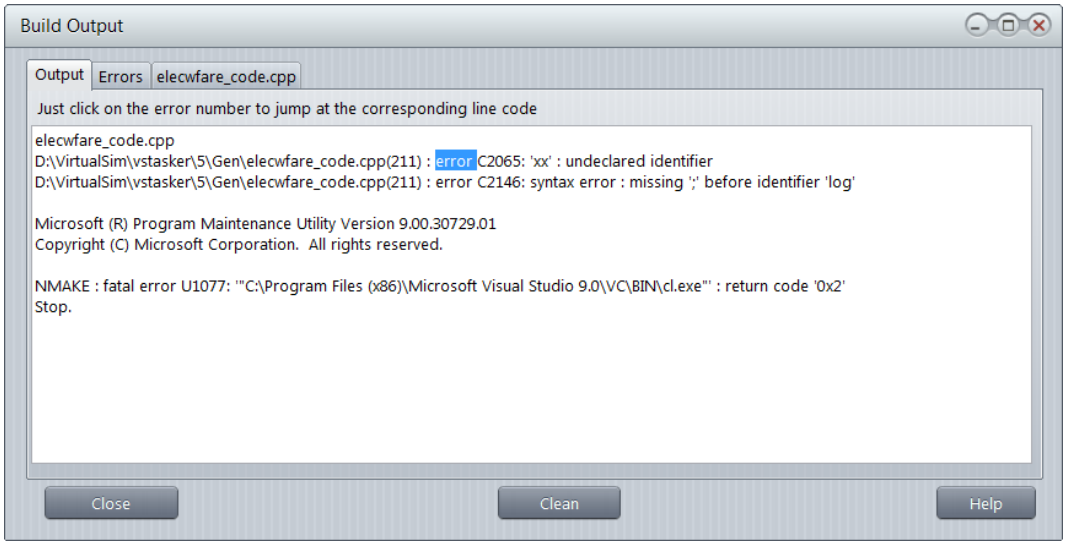
In order to run the simulation engine, vsTASKER must built it using the database definition.
Use the toolbar  button or the menu Tools::Compile or Tools::Recompile All (F7 or Ctrl F7) to do so:



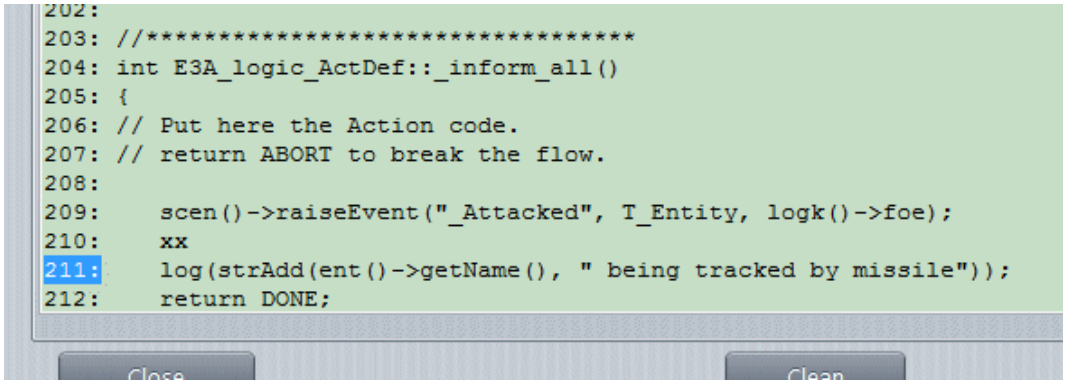

If the compilation fails, you will get the following window:




Click on the button **Debug** to open the Build Output window:

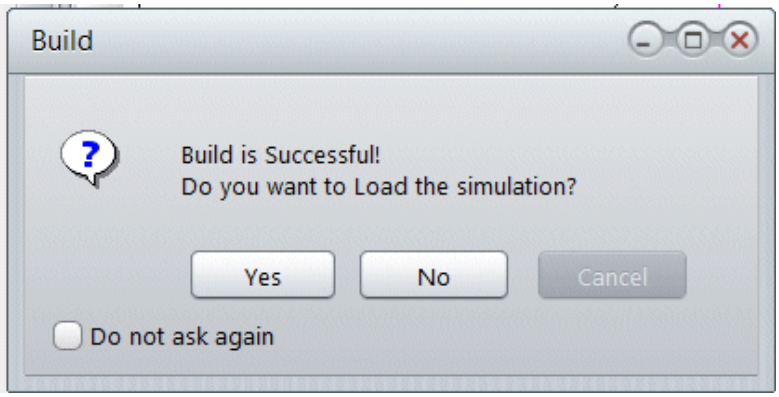



The display is the cl.exe or link.exe output. Nevertheless, you can **double click** on the **error** word to directly **jump to** the **line** of the faulty code:



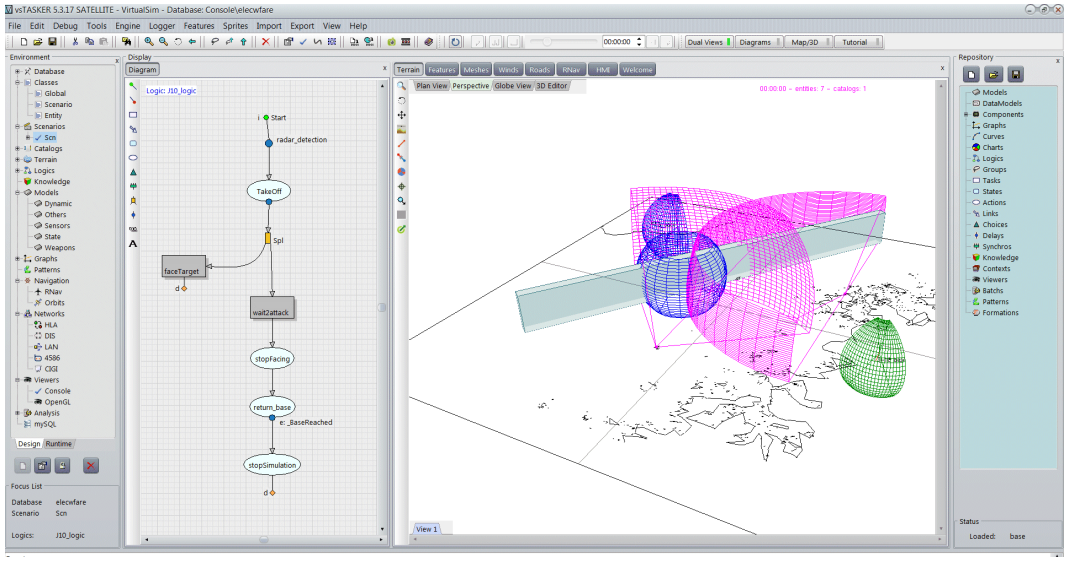

From here (`xx` is the problem, line 210), you must correct the code in the object itself (vsTASKER does not yet opens the object automatically).
To find the faulty object, the name of the function is always a good help.
Here: `E3A_logic_ActDef::_inform_all()`
That means the code is inside the Action named `inform_all` of the Logic named `E3A_logic`.

Once the compilation, link and build all succeed, the following window invites you to load the simulation engine (you can also use the  toolbar button)


Build
Build is Successful!
Do you want to Load the simulation?
Yes
No
Cancel
Do not ask again

# GUI Overview

# Main Panel



You can use the mouse over most of the icon buttons to display the **hint** associated with each of the object below.
Using Enter key (when the hint is displayed), you can edit the displayed text.
To quit editing, use the ESC key.
Automatic Hint can be disabled from the Environment settings Forms panel.

# Main Toolbar

# 1 Main

Used to **create** a new database, **open** an existing one (found in Data/db) or to **save** the current one (must be named first).

Used to **cut**, **copy** and **paste** any object selected either in the Environment Tree-List or the Diagram. Copied objects are stored in a buffer so that to be pasted into the same database or into another one. Paste can be used several time for multiple copies.
Note that this functionality does not work for multiple selected objects.
Cut does not work if the database is changed. It then works like a Copy.
The Copy buffer is cleared when vsTASKER exits.

Used to **search** and/or **replace** string patterns in the entire database.

Used at runtime only to send a message either to the scenario or to the selected entities (one or many)

Used to **reset** the Diagram and Terrain display mode (zoom and sliders) to **zoom 1**, default centered.

Return to the **previous level**. Does not work as a Back button but only to return to the parent call.

Used to **group**/**ungroup** a set of Logic objects. Conditions must be met for grouping while ungrouping can only be used when a group is selected.

Used to **go up** from nested **groups**.

Used to **delete** a selected object in the Diagram. Works also for multiple selected objects. Can be used before the selection.

Code Updater. Press this key to import from the generated code the changes made by the user.

Performs a consistency **check** on all the database and displays warning messages to correct defects.

 Used to display the **property window** of the selected object in the Diagram. If inside a Group and no object selected, display the Group property window. If inside a Context and no Rule selected, display the Context property window.

 Force the **code updater** to merge the generated code with the local database one. See Code Updater chapter in the **Programming Guide**

 Used to change a **connector shape** from line to spline to broken. The Object or the Link must be selected first.

 Revert the object **shape** to the **default** one.

 Used to **generate** the **C++ code** from all graphical descriptions and added user code, and **build** the simulation **executable** using database settings.

 Will immediately dump all the current memory to a file for later restore.

 Ccall the Restore manager window.

 Switch ON or OFF the CIGI connection with third party IG. See here for more explanation.

 Call this **documentation**.

# Control Toolbar

# 1 Control


 **Detach** the simulation engine from the shared-memory and force terminate the runtime mode.

 **Load** the Simulation Engine (if available) linked with the current database.

 Used to **start/restart** the simulation (once executable is loaded).

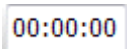
 **Pause** the simulation when running (use the start button above to resume)

 **Stop** the simulation (when running or paused). Will then ask to unload or not the simulation engine (see Detach). Answer no if you want to restart again without reloading (not all databases allow restarting after a stop, mostly the ones which are linked with external graphic engines like OSG or when user data are not properly cleaned).

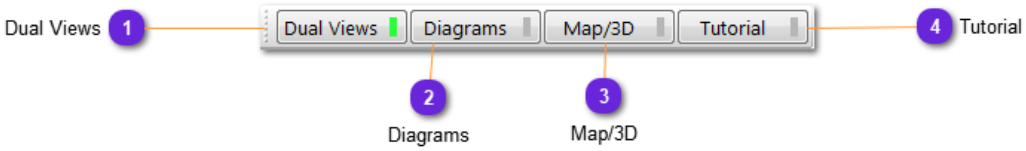
 Used to **increase** (right) or **decrease** (left) the **simulation speed** without cycle loss.

 When the simulation is running at higher or lower speed than normal (x1), this button force immediate return to **x1 speed**.

 Press this button to **time jump** if the target time is at least 10 seconds higher than the current simulation time.
If not, the simulation engine will toggle to **maximum speed**, meaning that there will be no idle time (neither yield) between two cycles.

 Used to **time jump** to the specified time. Must be set before time jump is pressed. Can take some processing time depending of the jump length and the models complexity behind as no cycle are loss and no extrapolation is used. This functionality guarantees that the situation at time jump is exactly the same as it would have been in normal play. Time jump is not reliable if the simulation implies inputs from other sources that are not time -jumping in sync.

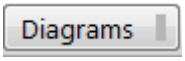
# Window Control




## 1 Dual Views

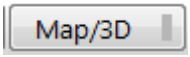

Split the display panel with Diagrams on the left and the Terrain Map/Globe on the right.

## 2 Diagrams



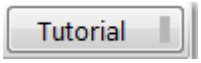

Show the Diagram full page on the display panel.

## 3 Map/3D




Show the Terrain Map/Globe full page on the display panel.

## 4 Tutorial




Show/Hide the User Manual (online help) on another panel on the outer right.

# Environment

The Environment tree-list displays all categories belonging to the loaded database.
Most of the database items can be accessed from this hierarchical list.
Selecting one item toggle views in the *Diagram panel*.
Double clicking (or Editing) items opens the Property window.

The tree-list contains all the **Categories** needed to build a full simulation environment.
Whenever a member of any Category is selected, the *Drawing Area* displays the corresponding graphical representation.

To **create** a new member of one Category, first select the Category on the tree-list then use the  button below the tree-list. The new member will be created with a default name that can be changed by slowly double clicking it in the list.

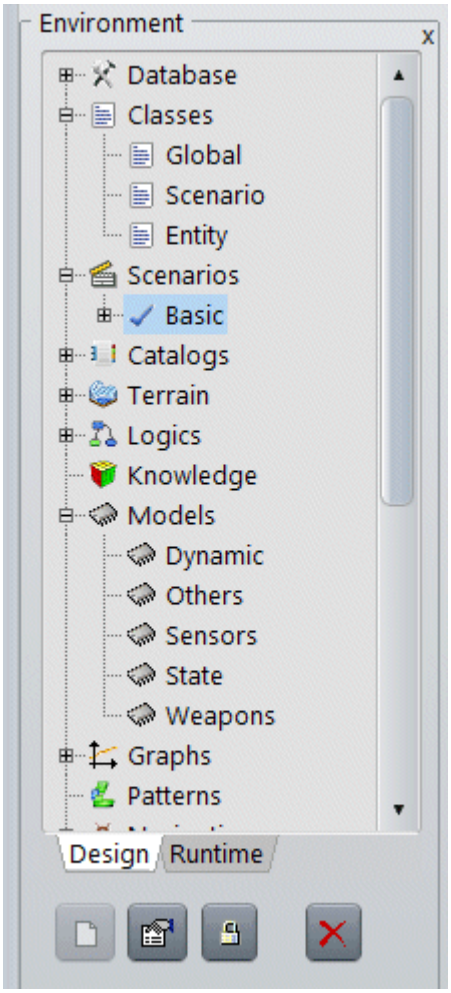
To display the **Property Window** of a Category Member, first select it in the tree-list then use the  button below the tree-list.

To **lock/unlock** a member from being modified by the user (from the *Drawing Area* or using its Property Window), first select it in the tree-list then use the  button below the tree-list. The corresponding symbol normally shows/hides a red circle sign to indicate that the object is locked.

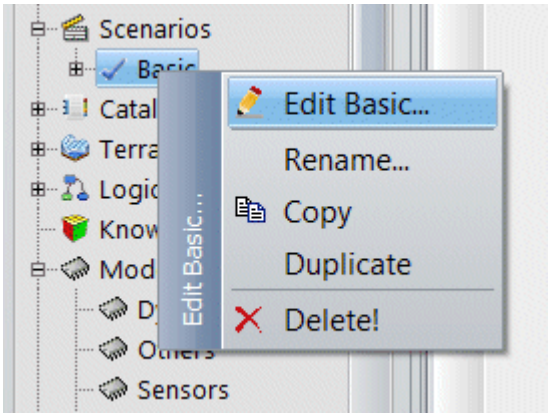
To **delete** a member from the Category, first select it in the tree-list then use the  button below the tree-list. All references to the deleted member/object are marked *_orphan* and shown in red. Build is not possible if unresolved references still exist.

Use  to remove a selected item (or DEL key). You cannot remove Category entries:

### Environment




The right mouse button displays a contextual menu to shortcut some push-buttons:



# Runtime List

This plugin maintains the runtime list that is displayed in the GUI Environment when the simulation is running.
The user can replace it using its own code.

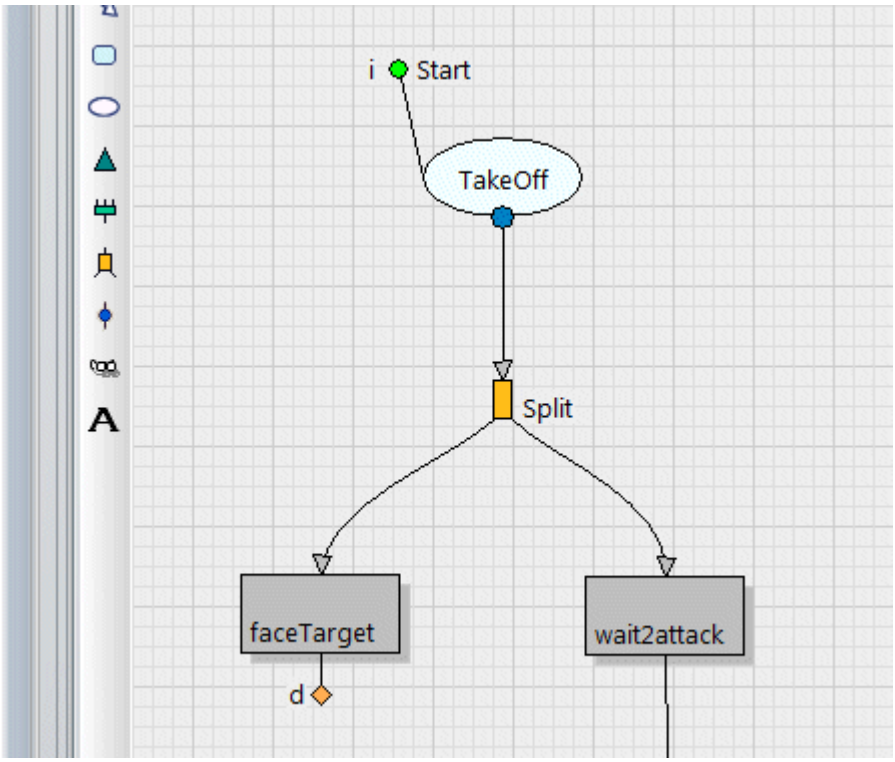
See Runtime.cpp in /plugins/runtime directory.

```
 //============================
typedef struct _RtEntList {
   char name[NAME_SIZE];   ///< name of the node
   int  image_id;          ///< image to use
   int  ent_ptr;           ///< optional pointer to EntScnRep
   int  touched;           ///< if touched, rebuild node + children
   int  count;             ///< number of children
   int  max;               ///< number of allocated slots
   _RtEntList** item;      ///< children [0..max]
} RtEntList;

//============================
typedef struct {
   int sorted;
   int treeview;
   RtEntList* elist;
} RuntimeList;
```

# Diagram

The Diagram part shows different aspects of the simulation database using graphical paradigm.

The toolbar on the left offers all objects user can drop on the Diagram area for code generation.
To drop an object on the area, click on the toolbar symbol then click on the Diagram area to drop it at the mouse position.
Object can be edited by double clicking, or right clicking then choosing Edit or Properties, or by using the  button.
The Diagram part reflect the selection of any item of the Environment list.



# Map Display

The *Display Area* displays **Scenario Map, Features, Meshes** or **Drawings** whose symbols is specific to vsTASKER. The *Drawing Area* is used to create visually most of the object logic. The point and click paradigm is intuitive for most of the manipulations.

The **Vertical ToolBar**, left of the *Drawing Area* shows which objects can be used in the graphical representation. It is context sensitive meaning that there is as many toolbars as environments.
Normal use of the Toolbar is as follow: select the object in the ToolBar then click on the *Drawing Area* to create it.

Once created:

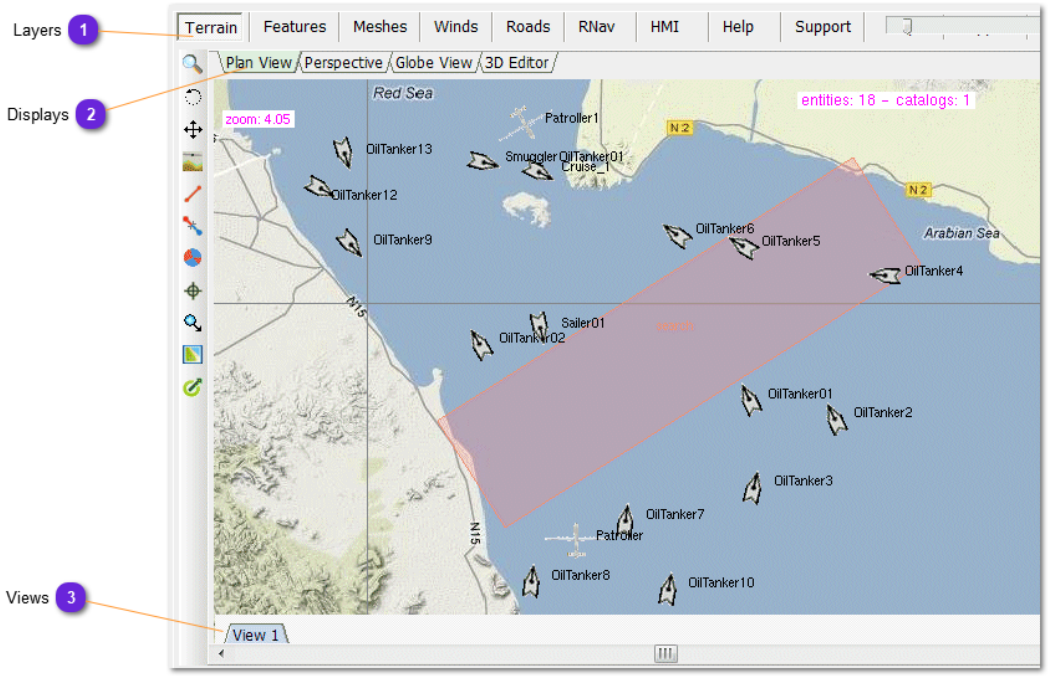
- To **select** an object, just click on it with the left mouse button or select it in the *Environment Tree-List*. It turns blue.
- To **move** an object, select it by clicking it then move the mouse while maintaining down the left mouse button. Release the button at end.
- To **deselect** the current selected object, simple click anywhere on the drawing area.

**Multiple selection** can be performed using the left mouse button. Depress the left mouse button without selecting anything then move the mouse to define the region. A dashed rectangle displays the region. Depress the mouse button to activate the selection. All objects inside the select area are selected and turned blue.

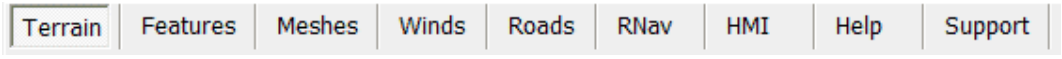
The map display is used to represent the gaming area. It provides several layers to create and edit different features that will be part of the simulation.
Mouse is used to drop entities onto the gaming area, edit their properties, define trajectories, paths or special zones, assign routines and interact during runtime with the simulation engine.

The map display is also used to design and draw human machine interfaces, using predefined sprites.

## Map Display




# 1 Layers




**Terrain**: basic terrain display including Entities
**Features**: trajectories, paths, special zones (...) editing
**Meshes**: edition of plan grids for path finding algorithm
**Winds**: definition of wind grids areas and wind tubes
**Roads**: edition of networks for road navigation algorithm
**RNav**: ARINC 424 air-navigation database drawing
**HMI**: use of sprites to create graphical user interfaces based on OpenGL/GLUT
**Help**: display the online help system
**Support**: display the support page (must be connected to Internet)

# 2 Displays


**Plan** view represents the God eye view, flat earth projection.
**Perspective** shows the Plan View with altitude effects.
**Globe** view shows the earth. Use this mode for viewing satellites.
**3D Editor** uses an external plugin (Delta3D or OSG), if available.

## 3 Views


Window views. Each of them keeps the zoom and centering, allowing the viewer to switch from one area of the scenario to another, or from one zoom factor to another or both, with several combinations.
Entity tracking mode is also kept, enabling to follow multiple entities on multiples views at runtime.
The number of views can be changed from the Database::Settings::Map menu.

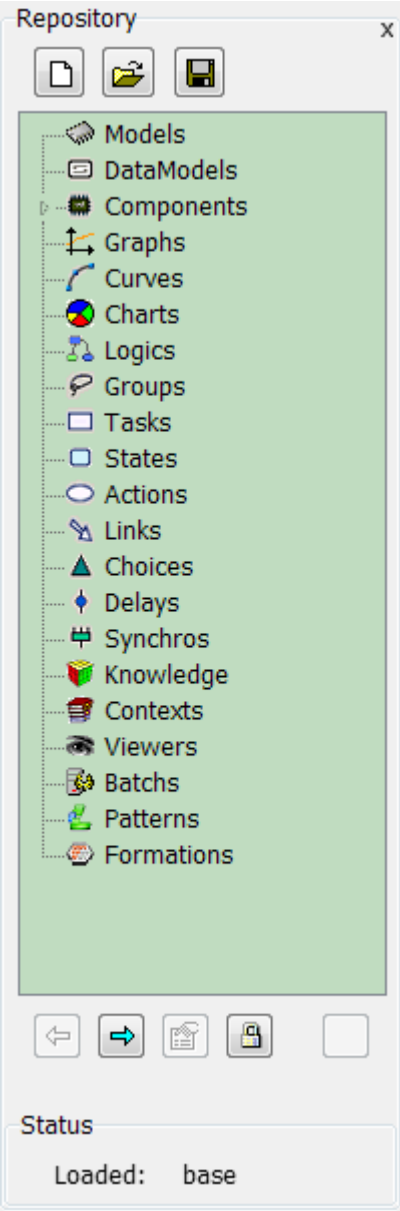

*This feature is not available yet.*

# Repository

Repositories are specialized storages organized by Categories.

User can select and drop any stored object to quickly create Logics, Knowledge, Components, etc.
There can be any numbers of repositories, organized by types of simulation. For example, you can create a repository for Defense, another one for Air Traffic Control, etc.

You can see a Repository as a set of predefined objects stored to accelerate the database setup.



To drag any selected single object to the current Repository, select the object on the Environment or the Diagram then use  . The Object will be visible below the corresponding Type.

To drop any Object from the Repository, select the object in the list, use  then paste it in the correct environment (if the object belongs to a Logic, open a Logic before pasting it.)

A Repository can be saved or retrieved. By default basic is used.
Some special repositories have been made by templates, to ease the scenario edition with drag and drop.

Create a new Repository using 

Save the current Repository using 

Load a previously saved Repository using 

Repositories can be exchanged between users. They are saved in data/rep. The repository definition ends with .rep. The associated data is a folder. Both are requested.

Import & Export capability can also be used instead of Repository. It is up to the user to decide which way he likes the most. Better to avoid mixing the both to reduce complexity.

The **Repository** can be used as a convenient storage for all purpose members of vsTASKER categories.
Normally, the repository should only contains templates or general purpose objects that can be re-used as is instead of being build from scratch.

The user can create as many repository as he needs. It is a good idea to have a repository per project or per domain of expertise, gathering only category members that apply to the project or domain.

To **create** a new Repository, use the  button above the *Tree-List*. The new repository has no name. One will be requested at Save.

To **open** a new repository, use the  button above the *Tree-List*. Repository are stored in *Data/Rep* directory. Select the [.rep] from the list.

To **save** a repository, use the  button above the *Tree-List*. If the repository is a new one, you must give it a name. Please, insure that the repository is located in the *Data/Rep* directory. Be aware that no warning is issued if the given name exists already!

### Repository

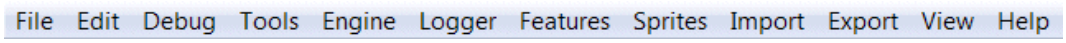
To **add** a new object in the Repository, select it from the *Environment Tree-List* then use the  button, below the *Repository Tree-List*. This action adds a copy of the selected object. If an object of the same name exist in the Repository Category, an error message is shown.

When an object is selected in the repository, the Diagram shows the graphical description (if available). Note that the Diagram background turns light-green. All graphics are locked in Repository mode by default. The user must **unlock** each object using  in order to modify the graphics.

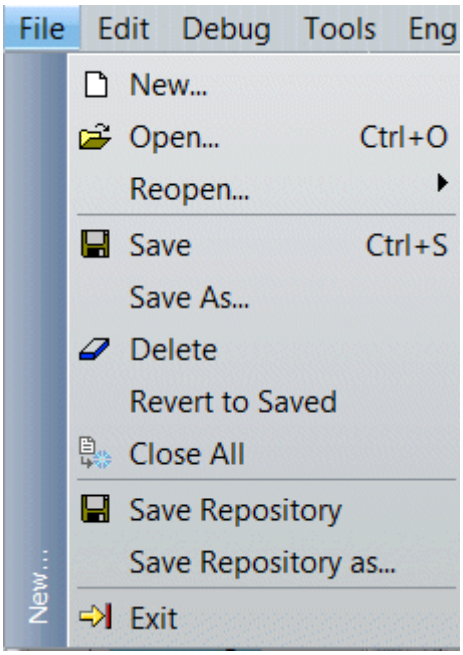
Whenever an object is selected in the Repository, the  button, below the *Tree-List*, displays its **Property Window**. No change can be performed on the repository. *OK* and *APPLY* buttons are grayed out.

To **delete** an object from the Repository, first select it in the *Tree-List* then use the  button below the *Tree-List*.
You can **hide** the Repository panel from the **View** menu.

# Main Menu


File
Edit
Debug
Tools
Engine
Logger
Features
Sprites
Import
Export
View
Help

# File




**New**: Create a new database

**Open**: Open an existing database located in %VSTASKER_DIR%/data/db preferably. This is working directory. A database file ends with .db

**Reopen**: List the last opened databases

**Save**: Save the current database. The previous version is renamed database-name.bakX where X is a sequential number.

**Save As**: Rename and save the database under another name. Creates the corresponding directory.

**Delete**: Delete the database on the disk, including the directory, then wipes out the database from the memory. Cannot undo!

**Revert to Saved**: Revert to the currently saved database. Useful to forget last changes.

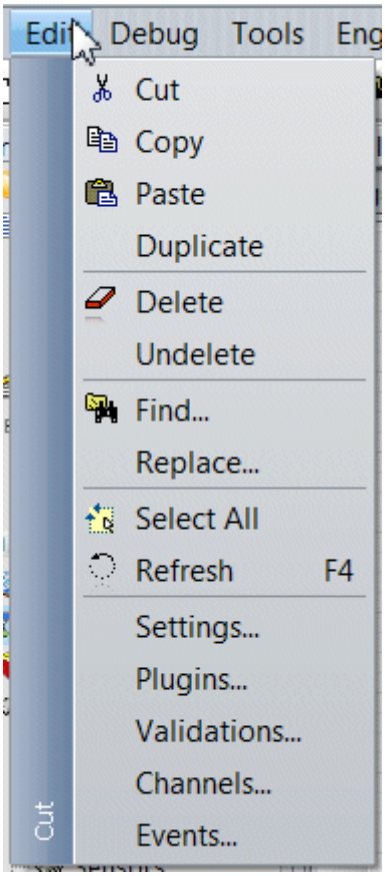
**Close All**: Close the database and free the memory.

**Save Repository**: Save the current repository.

**Save Repository As**: Rename and save the repository under another name.

**Exit**: Close and terminate the program. If database has not been saved, will be asked for save before closing.

# Edit




**Cut**: Copy the selected items into clipboard and remove it from the database.

**Copy**: Copy the selected items into clipboard

**Paste**: Copy from the clipboard to the database, where it should be copied, so according of its type. For example, if a Logic is in a clipboard, then, pasting it will add it as a new (or duplicated) logic, event if the user is focusing another category. After a paste, the clipboard still contain the items.

**Duplicate**: Copy/Paste locally whatever has been selected. This does not use the clipboard.

**Delete**: Remove from the database whatever is selected. Confirmation dialog appears.

**Undelete**: Tries to undo the deletion of the latest objects. Undo list is cleared when the database is closed.

**Find & Replace**: Use this window to find and replace strings into the whole database. See Finder

**Select All**: Selects all visible items in the focused category

**Refresh**: Force redraw of both Diagram and Maps

**Settings**: Database properties, see here

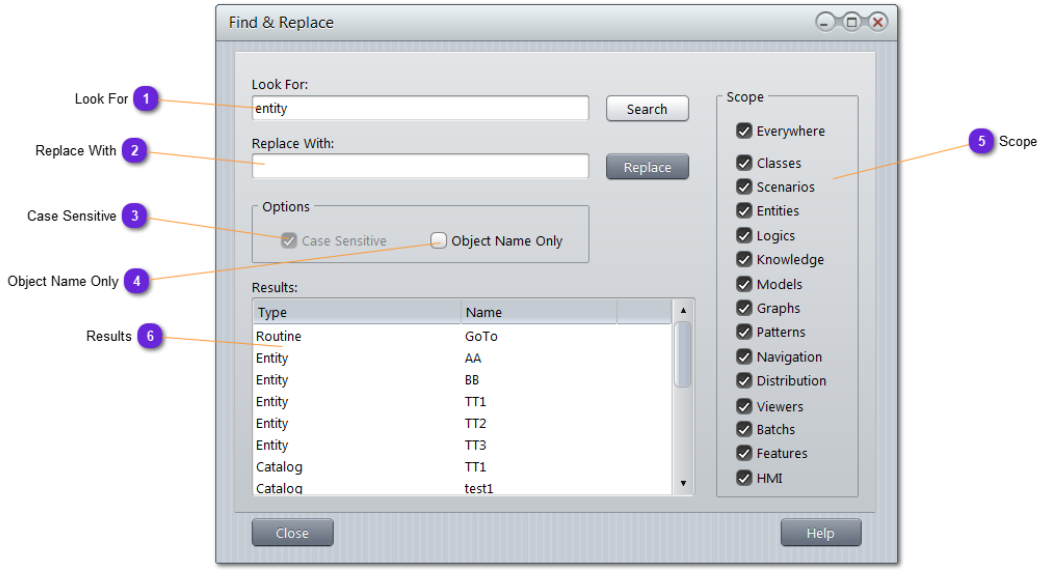
**Plugins**: Plugins editor, see here

**Validation**: Validation routine editor, see here

**Channels**: *Coming soon...*

**Events**: *Coming soon...*

# Finder



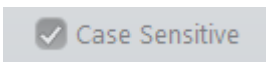

## 1 Look For


Enter here the **pattern** to be searched.
Click on Search button to start the process.

## 2 Replace With

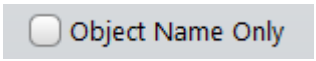

Enter here the **string** that will replace every occurrence of the found **pattern** (see above).
Click Replace to force replacement of all.

## 3 Case Sensitive



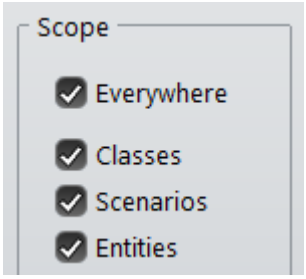

Check this option to force character **case** to become a search factor.

## 4 Object Name Only



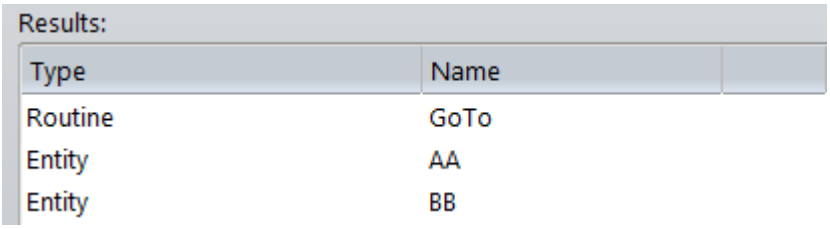

If this option is checked, only the **name** of each item will be searched. All occurrence inside code area will be excluded.

## 5 Scope



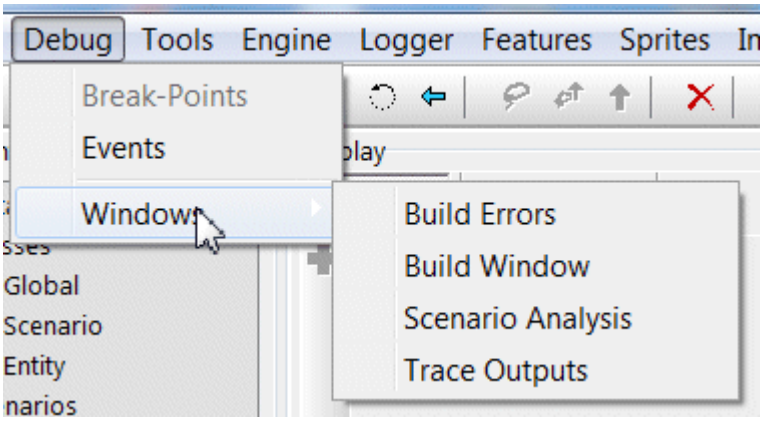

Check the categories that will be searched (or processed for replacement).

## 6 Results




List all categorized items with the searched pattern found inside.
Double click each item to open their property window.
It is up to the user to look for the pattern as this one will not be highlighted in the code.

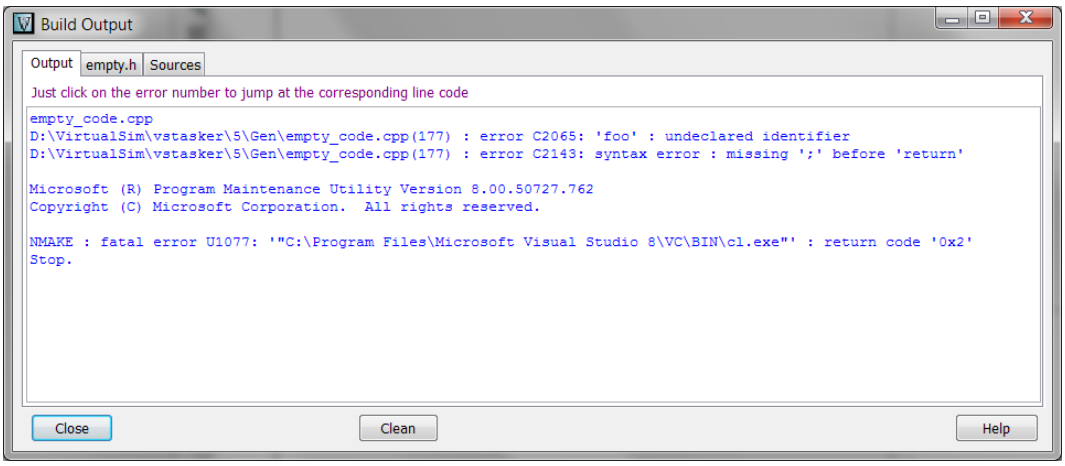
# Debug


Debug
Tools
Engine
Logger
Features
Sprites
Break-Points
Events
Windows
Build Errors
Build Window
Scenario Analysis
Trace Outputs
Global
Scenario
Entity

# Build Output

This window displays the compilation output when error occurs.
Output panel displays the makefile output.

In this example, double-click `error C2143` and the window will directly jump to the correct position:




When you see this window at compilation time, that means something went wrong during the compilation of the generated code or the link or bad user-paths or missing user-libraries (...) and must then be corrected.

Typically, the error comes from the user added stuff.


*If the window is white blank, this can come from the missing C++ environment variables. Publish them at Visual Studio install or use vcvars32 before starting vsTASKER.*

## • Output

This panel shows the result of the **compiler/linker output**. It is a raw display of the compilation/link of the generated files.
You should try to understand the meaning of the error in the displayed report.

If the database contains a lot of entities and Logics, the window might take some time to show up.

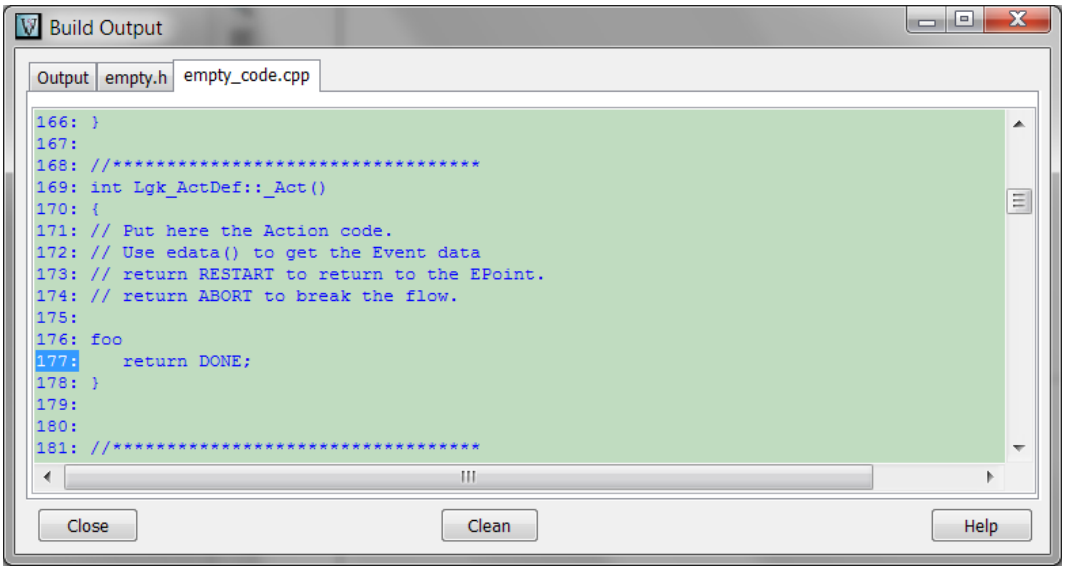
To jump at the corresponding error line, **double-click** the line where the error is mentioned.
In the above example, double-click `error C2143` and the window will directly jump to the correct position:

# • Errors

This panel shows the **generated C++ code** (one of the generated files in /Gen). Normally, most of errors occur on this file.
In the *Output* panel, a line number is given for each error. The *Errors* panel displays line numbers for retracing the error.

As the user-code is normally embedded into methods with generated names, it might be difficult to find the faulty code in the correct object.
Nevertheless, the name of the method is self explanatory to guess which object is concerned with the error.

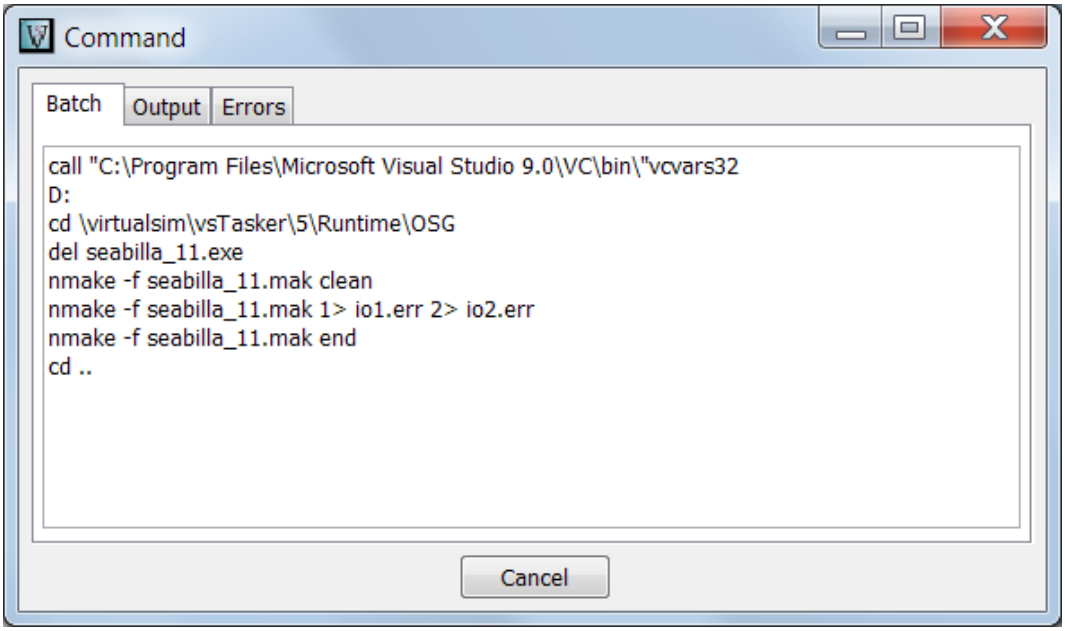
User-names given to Logics, Knowledge, Models and other objects are kept to build the method and class names.
With Entities and Scenarios, it is a little more tricky because user-names are not used, only IDs.



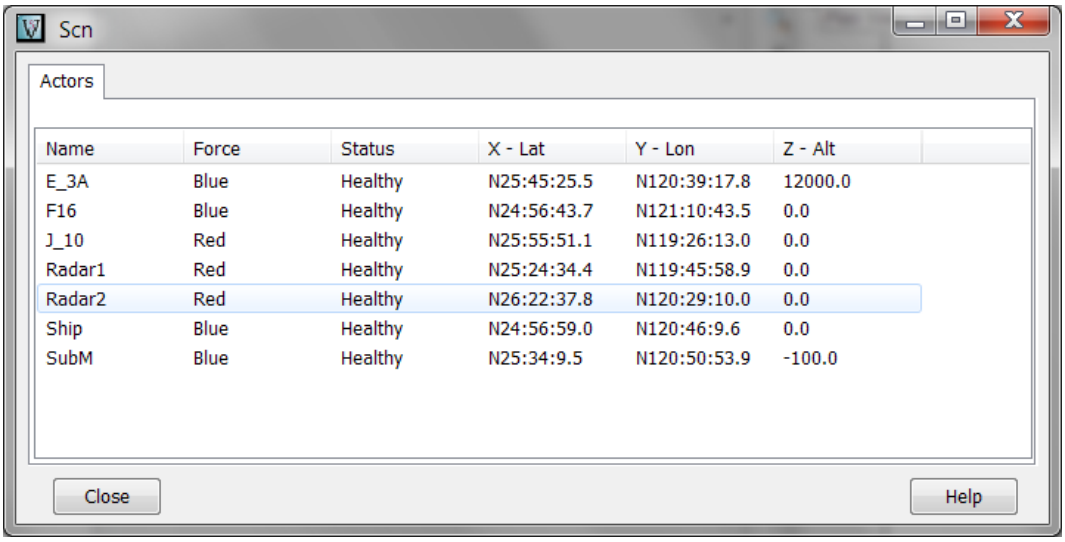
# Build Window

This window displays the batch builder, usually embedding an external compiler call.

The Batch panel lists the batch command sent to the Console (command)
The Output is displayed after the batch (process) ends. It shows the normal output as 1> (file io1.err)
The Error is displayed after the Output. It shows the error output as 2> (file io2.err)



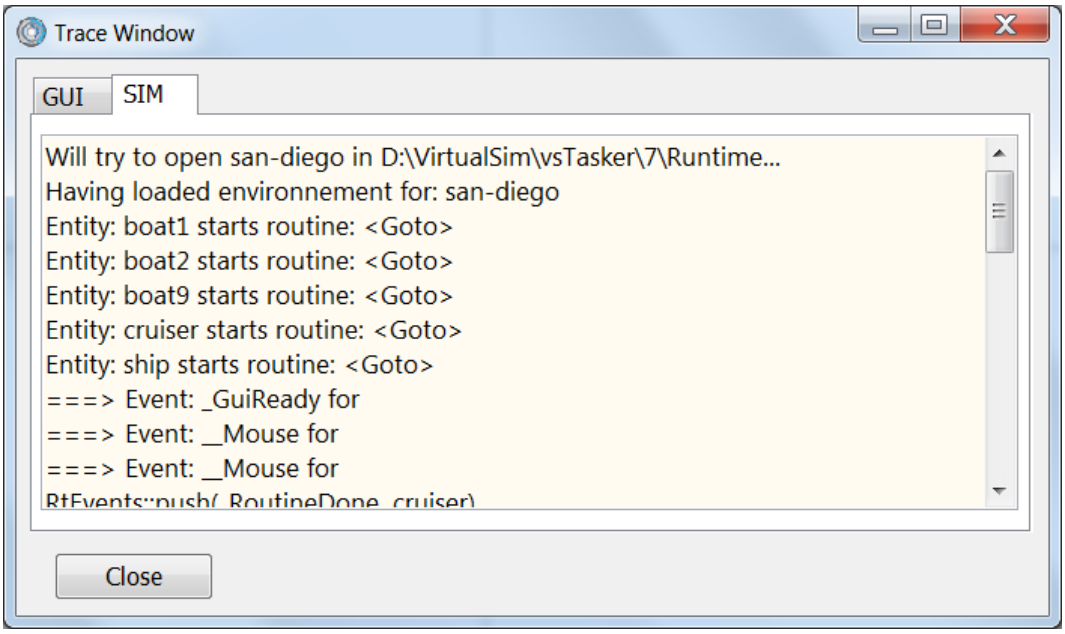
# Scenario Analysis



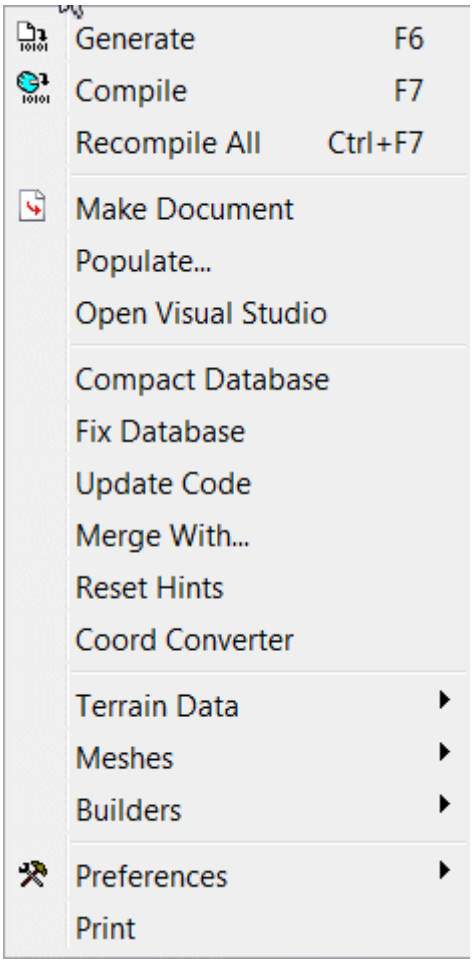

During runtime, this window lists all entities of the scenario with their actual status (Name, Force, Status, Position)
Automatically closes at scenario end.

# Trace Outputs

Use this menu to read the two trace/log files generated by the GUI and the SIM.
These files are location in the application root directory:
gui_out.txt and sim_out.txt



# Tools




**Generate**: Generate the C++ code

**Compile**: Compile the generated C++ code

**Recompile All**: Clear the Debug or Release directory for a clean link. To be used for each new database at first.

**Make Document**: Try to build a text document that lists all database definitions including user added help descriptions. Limited to Logic.

**Populate**: Call the populate window.

**Open VisualStudio**: This command is useful to launch the proper C++ MS-Studio (if several installed on the computer) according to the setting done in the compiler section. Also, the MS-Studio will inherit from the environment of vsTASKER GUI which might be helpful if local paths and environment variables are used.

**Compact Database**: Remove all backup and empty files of the current database to reduce its size. Do it after each stable phase.

**Fix Database**: Redefine all object IDs and refresh all links to make sure of the uniqueness of each reference. Remove orphan objects.

**Update Code**: Force comparing the generated files and the current database user code to retrofit the changes in the file into the database. This feature allows user to change the code inside Visual Studio and have it automatically and properly retrofit back into the vsTASKER database. See Code Updater chapter in the **Programming Guide**.

**Merge Databases**: See here

**Reset Hints**: Delete the hint database (the one the user hide by clicking: *"do not show again"* on some warning windows)

**Coord Converter**: See there

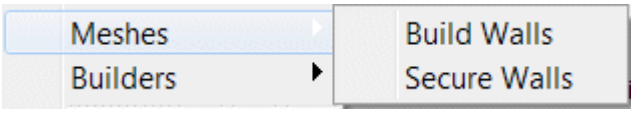
**Terrain Data**: See there

**Meshes**: See below

**Builders**: See below

**Preferences**: See there

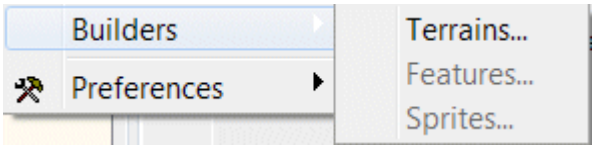
## • Meshes




**Build Walls**: automatically put walls on the selected Mesh according to the thresholds defined in the Mesh property window.

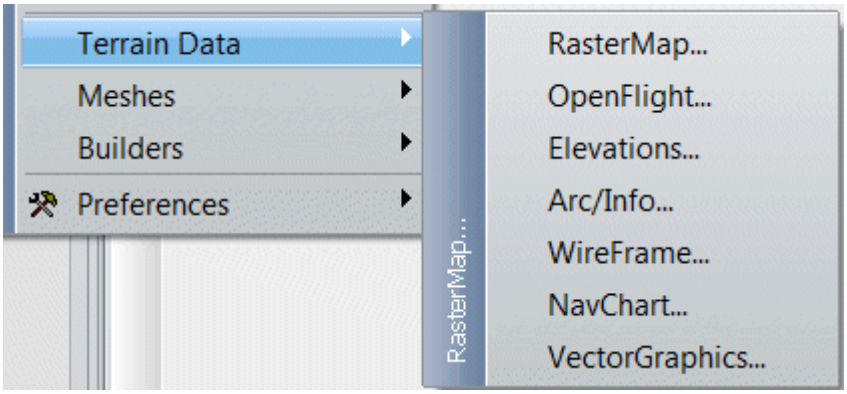
**Secure Walls**: automatically surround walls with a no-walk area to prevent entities from getting too close from walls.

## • Builders




**Terrain**: call the VirtualSim Terrain Builder tool.

# Terrain Data




**RasterMap:** call the RasterMap setting window

**OpenFlight**: call the OpenFlight setting window

**Elevation**: call the Elevations (DTED/DEM) setting window

**Arc/Info**: call the Arc/Info setting window

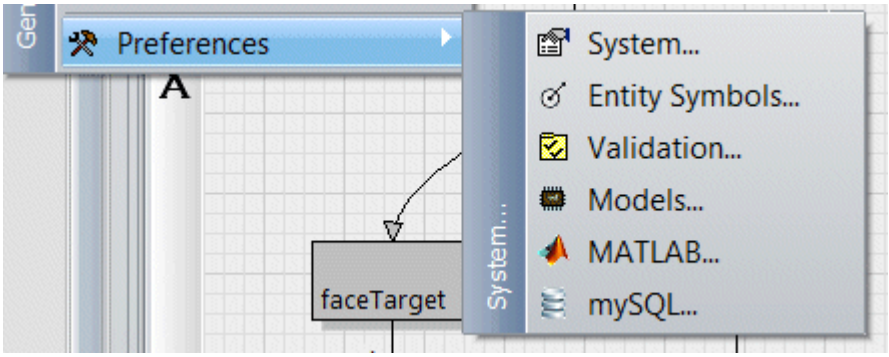
**WireFrame**: call the WireFrame setting window

**NavChart**: call the NavChart setting window

**VectorGraphics**: call the SvgMap setting window

# Preferences

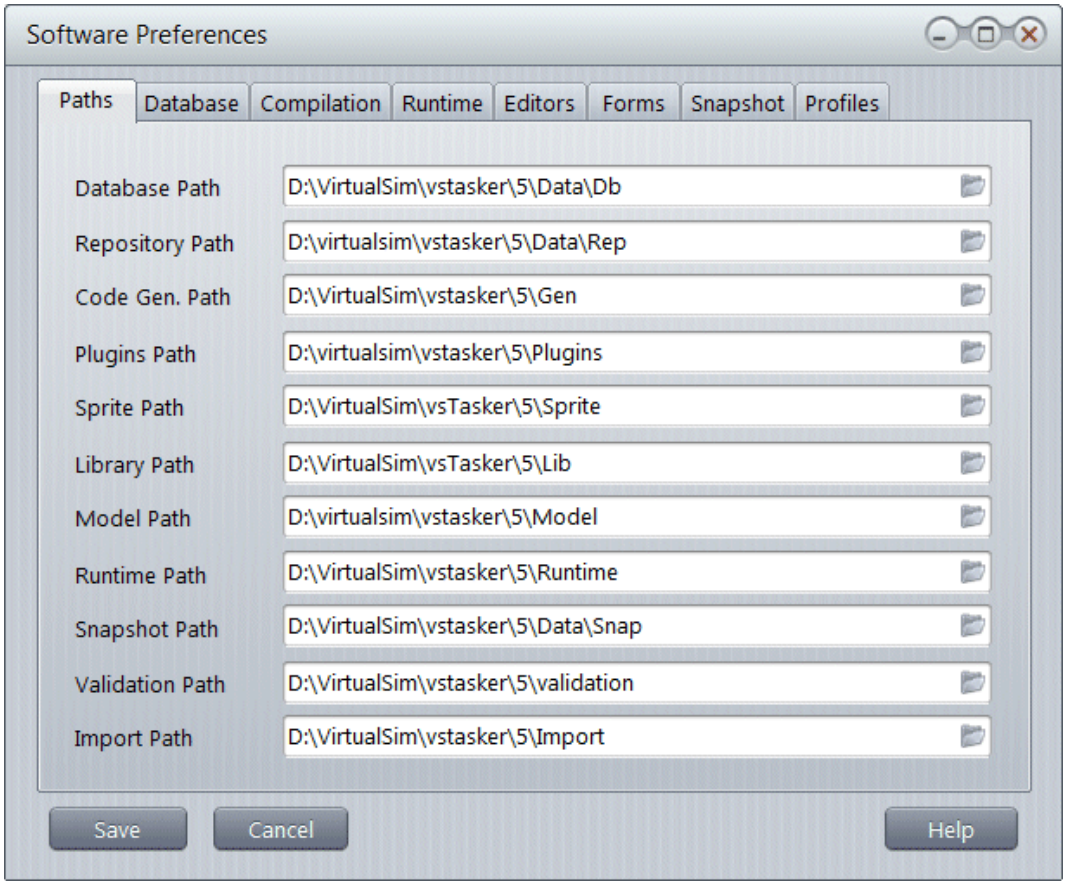
We will see here how to setup and customize the GUI panel and the simulation environment.



# Environment

Environment settings covers the main application options.
It also provides default values to Database options that may overwrite (for the database only) Main application settings.

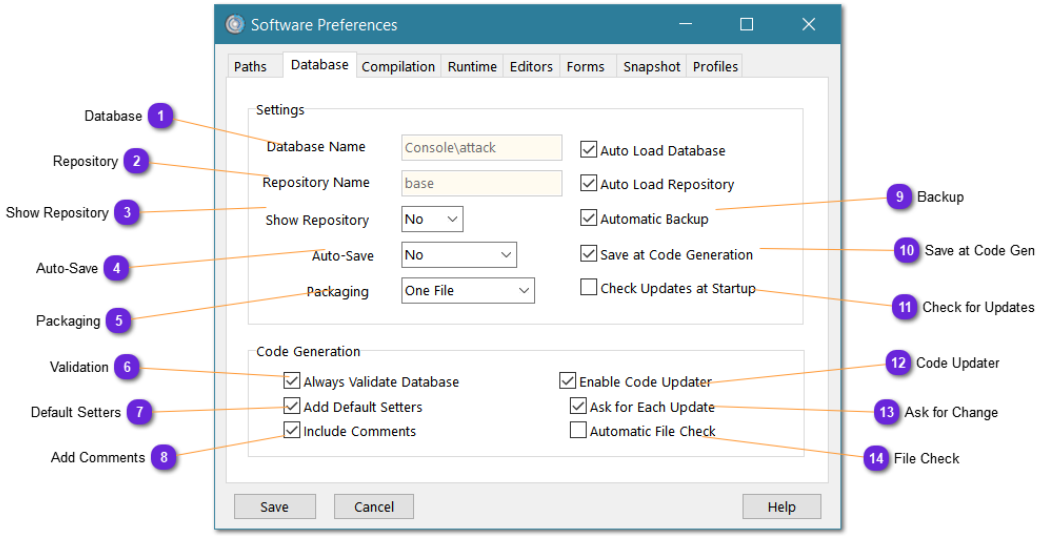
# General



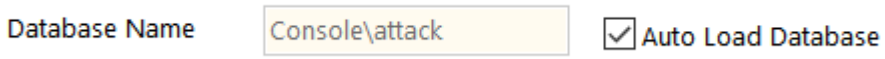

Change the **default** path of major vsTASKER parts.
This setting becomes application dependant.
The above are the default values.

It can become interesting to change some of these values to share the application amongst several projects or users.

# Database




## 1 Database

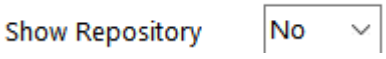



Default database to be loaded at startup (if option is checked).
This one is normally the latest one loaded before closing the application.

## 2 Repository

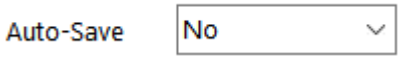

Name of the repository to be loaded by default at application startup (if option is checked).

## 3 Show Repository



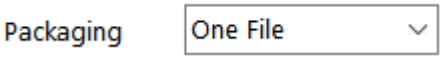

When set to YES, repository panel will be displayed at startup (on the right).

## 4 Auto-Save



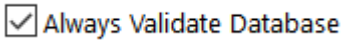

When this option is set to YES, auto-saving of the database will automatically occur periodically as defined by the user from the selection list:
**10 minutes**
**20 minutes**
**30 minutes**
**45 minutes**
**1 hour**

## 5 Packaging



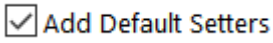

When this option is set to One File, vsTASKER will embed the database under one main file (default)
When the option is set to Multiple Files, vsTASKER will create one file per Environment category (Logic, Knowledge, Models...) This will allow database sharing.

## 6 Validation



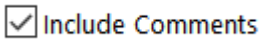

When checked, the database validation tool is called before generating the code. It is checked by default.

## 7 Default Setters



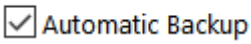

This option force the generation of the setters with default values in the `setDefault()` function generated for each vsTASKER objects supporting a Dyn-UI. It has become optional since version 6.1 as since, it is possible (and recommended) to specify default values directly into the header file for embedded interfaces INTF[], using DEF[] keyword (see here).
Initially, embedded Data-Model or Components (like `LinearCtrl`) could not be initialized with default values other way than from the code.
Activating this option for version 6.1 and above is useless and should remain off.

## 8 Add Comments




When disabled, all comments (system and design) will be removed from the generated code. This can produce smaller code or difficult to understand code when this one must be shipped with a locked database (for recompilation). Default is ON

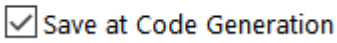
## 9 Backup




If this option is checked, every time database is written, the previous one is backup with a sequential number (db_name.bak223).
If unchecked, the old database is erased.

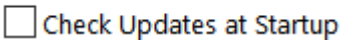
Option is checked by default even if this can lead to having many files in the database. Reverting to a previous version is easy as the highest number represents the latest version of the database.

## 10 Save at Code Gen




When this option is checked, database (design) is saved every time the code is generated.
The runtime database is always saved at code generation as this one will be loaded by the simulation Engine.

## 11 Check for Updates




When checked, vsTASKER will try to connect to the Internet at each application startup to check if a new release or update is available.
This option is off by default as it slows down the launch when Internet is not available, slow or behind a proxy.

## 12 Code Updater

☑ Enable Code Updater

When enabled, the generated code will contain all the necessarily blocks to retrieve the user changes in the remote files and retrofit them. Default is ON.

See Code Updater chapter in the **Programming Guide**

## 13 Ask for Change

☑ Ask for Each Update

When Code Updater is enabled, whenever a change has been detected in the code, user can accept or reject the update. Disabling this option will authorise the updater to retrofit all changes silently. Default is ON (user must accept every change)

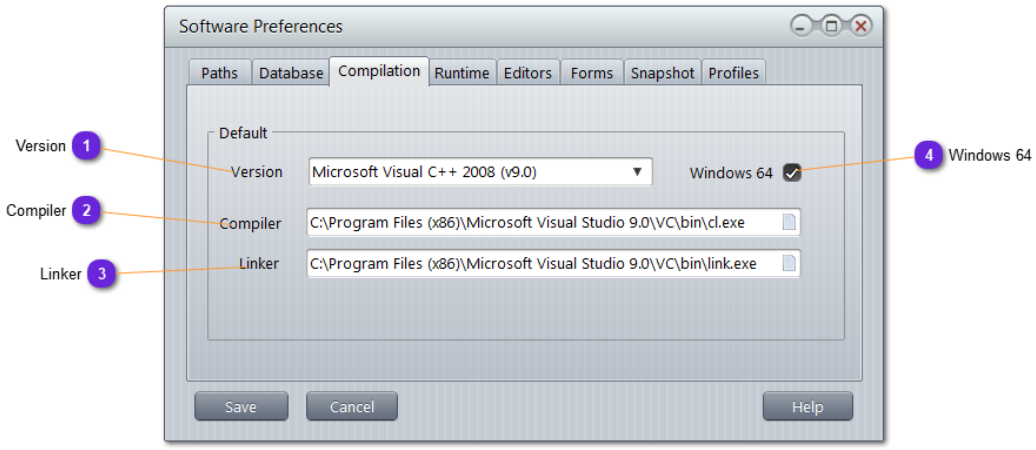
## 14 File Check

☐ Automatic File Check

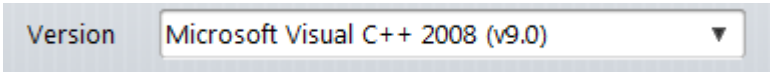
When checked, each time the GUI is focused (or database loaded), the modified time stamp of the generated code files is compared with the database one. If one of the files has been modified, the updater will trigger.

# Compilation

This compiler settings are used to initialize the default database settings for each new database. The user can change at the database level, what compiler and linker to use if they are different from the default one.

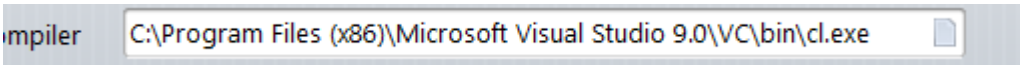



## 1 Version



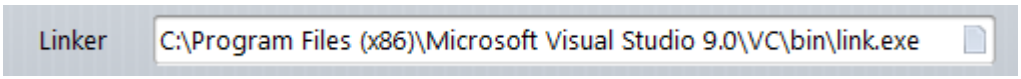

You can select here the default compiler you want to use and the target platform: win32 simulation engine (default) or x64.

## 2 Compiler



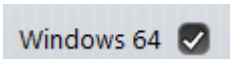

Default location for the C++ compiler to use (**cl.exe**). Microsoft Visual C++ only supported.

## 3 Linker




Default location for the C++ linker to use (**link.exe**). Microsoft Visual C++ only supported.

## 4 Windows 64




Check this option if you are running under Windows 64 bits. This option automatically updates paths accordingly.
This option **shall not be confused with 64 compiler settings** which will need 64 bits library and will generate a 64 bits simulation engine. Only the compiler version defines if the target simulation engine will be 32 bits (win32) or 64 bits (x64).

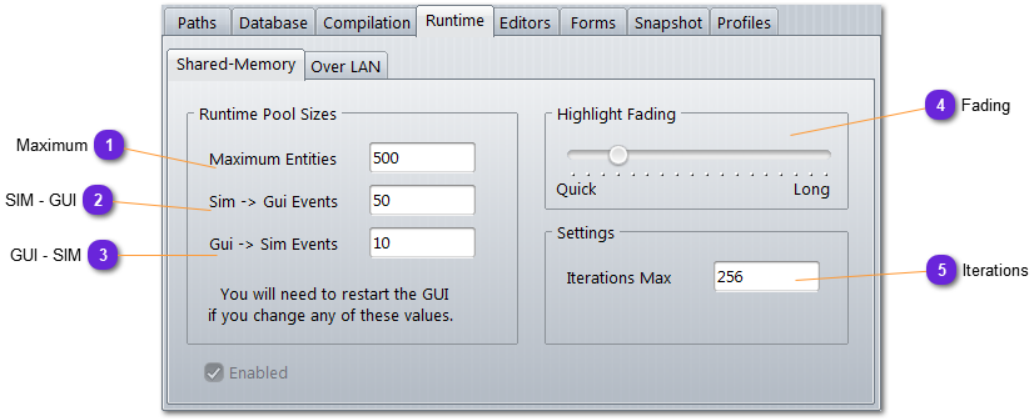

This option has been disabled in v7 and checked by default as Windows 32 bits is no more used. Besides, vsTASKER will become 64 bits native eventually.

# Runtime

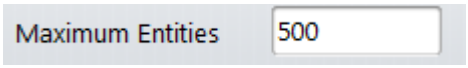
vsTASKER creates a shared memory segment with static pools for communication between GUI & SIM engine.
The pool increase the communication speed but has size limitations.
This does not affect LAN connection mode.

Here are the settings.

# Shared-Memory

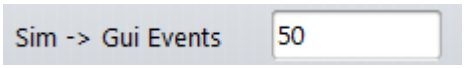



## 1 Maximum



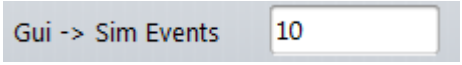

If your simulation can have a runtime more than the maximum number of entities, increase this value to the maximum number of entities that can exist simultaneously at a given time.

## 2 SIM - GUI



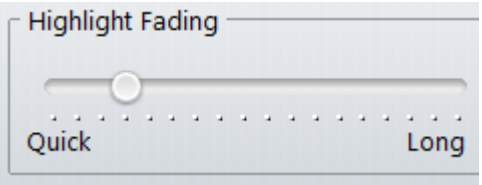

Size of the pool to store events coming from the Sim engine to the Gui. As the Sim can run at a much higher frequency than the Gui; this pool can become full and messages can then be lost. Increase this pool size if you start having warning messages displayed.

## 3 GUI - SIM




Events sent from the Gui to the Sim engine are few and does not require a big pool as the Sim engine is fast enough to process the pool much quicker than the Gui can fill it.

## 4 Fading

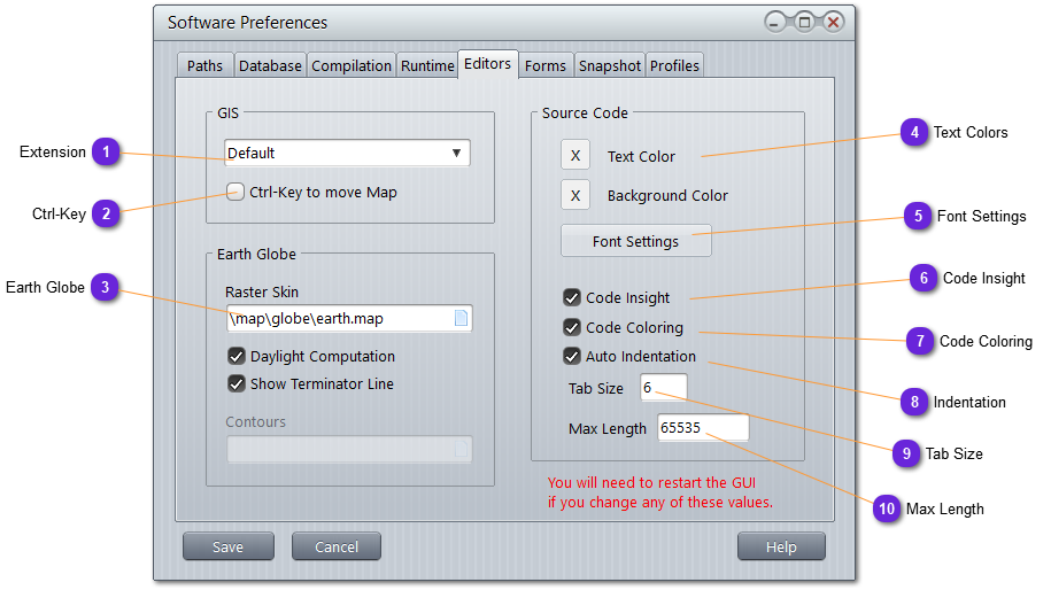



During runtime, activated objects (for selected entities) are shown with a magenta color on the GUI diagram panel.
The SIM is constantly sending refresh signal so that they maintain the magenta color.
The normally fade out to black color gradually.
The above scale sets the time needed for an activated Object to return to its "inactive" state (note that this is only a visual hint and does not reflect the simulation state at a very moment).

## 5 Iterations


Define here the maximum number of iteration vsTASKER GUI will attempt in a single loop if the incoming event buffer still contains data.
This can occur when the simulation is running at high speed and that the GUI overrun (and then constantly freezes).
The higher the value, the quicker the pool is emptied but the longer the GUI will react to user events.

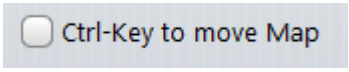
# Editors




## 1 Extension

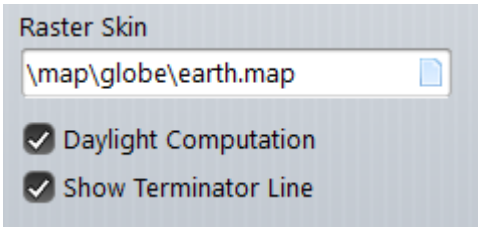

Use another GIS in addition to the default OpenGL main one.
Not available on standard product line.

## 2 Ctrl-Key



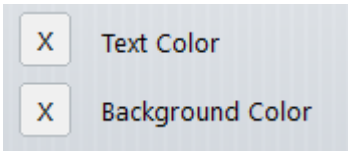

When checked, map can be moved/panned using the keyboard Control Key with Left mouse button depressed.
When unchecked, mouse Right button must be used.

## 3 Earth Globe



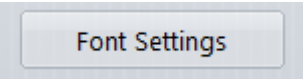

Select the skin (raster tiled maps) that will be used to shape the earth. You find them in data/map/globe directory.
If **Daylight Computation** option is checked (only for raster skins), shade will be applied at runtime and design according to the UTC time set in the scenario.
If **Terminator Line** option is checked, day/night line will be shown on both the terrain and the globe.

## 4 Text Colors

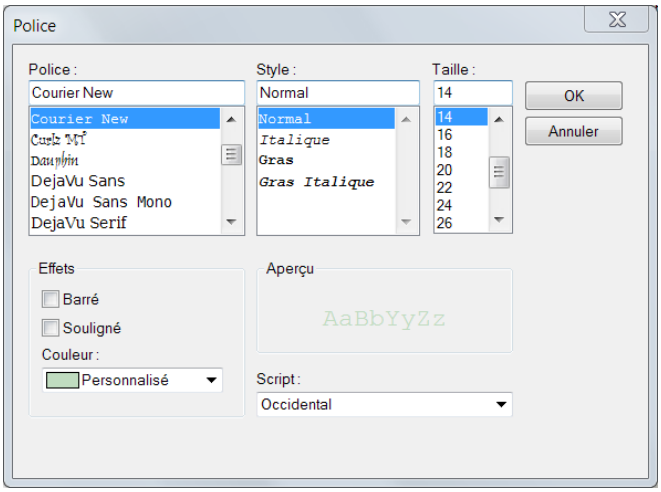



Set the text color and background for the embedded vsTASKER text editor. Affects all windows.
*Application restart mandatory.*

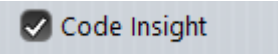
## 5 Font Settings




Select here the font Police you want for all text editors used in vsTASKER.

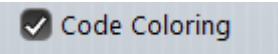



## 6 Code Insight




When this option is enabled, text editor will provide popup list for completing the entered source code.
Code Insight module refreshes its database using the database generated code.
See CodeInsight.exe description for rebuilding the code insight database.

## 7 Code Coloring

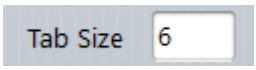



When this option is enabled, text editors will automatically highlight C/C++ keywords.
See /data/resource/keywords.txt for a list of all predefined keywords. User can update this list.

## 8 Indentation

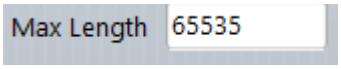

When this option is enabled, carriage return inside a text editor will automatically position the cursor according to C/C++ block standards. Auto Indentation also visualize the pairing parenthesis or block.

## 9 Tab Size




Number of space used for each Tab key pressed.

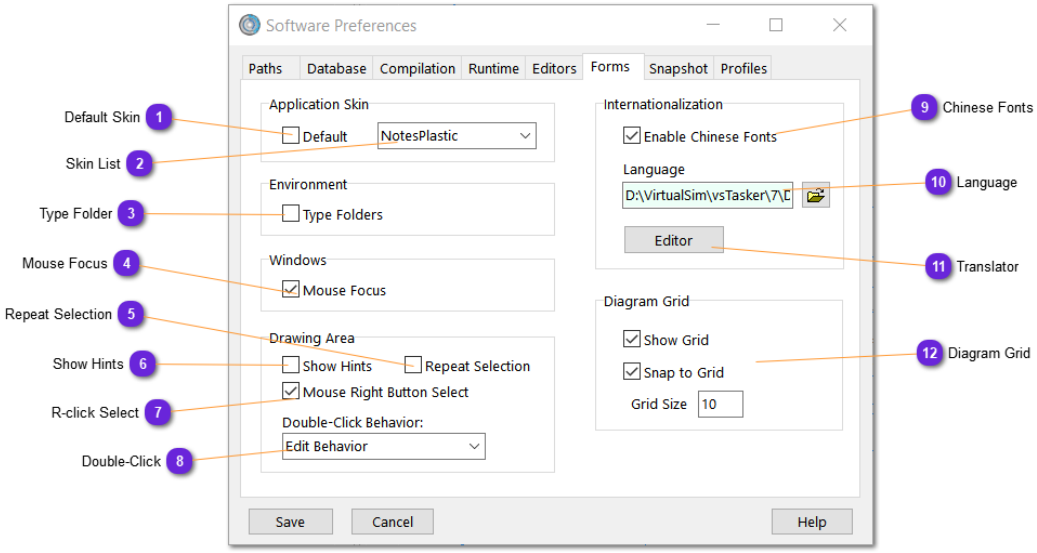
## 10 Max Length



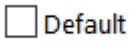

Default maximum length for a text editor panel in terms of characters. Value of 0 is default (64K or 65535 characters).
Change this if you face limitation in the edition of your code.


*The longer the length of the text, the more difficult to edit. Processing of the format might slow the edition.*

# Forms




## 1 Default Skin




Click this option to force the use of the version default skin (manufactured one).
Uncheck it to select from the drop down list another one.

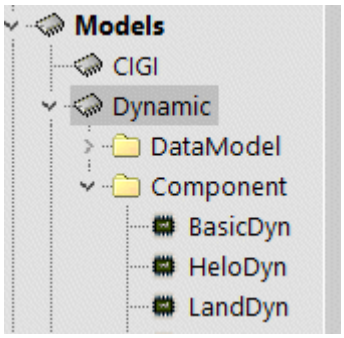
## 2 Skin List


Select from the list the skin to use for the application (need restart).
If No Skin is selected (first entry of the list) then the default windows style will be used (need restart).

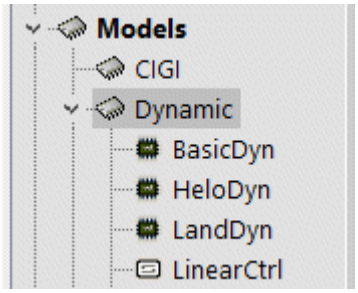
## 3 Type Folder

☐ Type Folders

When selected, Environment tree-list gathers objects by types and list them under a specific folder (see below):




When unselected (default), objects are listed in sequence and only icons reveal the type:

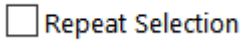



## 4 Mouse Focus

☑ Mouse Focus

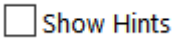
When selected, the panel below the mouse is automatically focused, allowing the scrolling with the mouse wheel.
This mode is automatically deselected at runtime to allow other windows to be displayed on top of the vsTASKER GUI (like the runtime window or any viewer output).

## 5 Repeat Selection



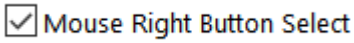

If checked, when a Diagram ToolBar button is depressed (for creating a Logic or Knowledge object), clicking on the Drawing Area creates as many objects as clicks. To stop the mode, depress the ToolBar button again.

## 6 Show Hints




When checked, hint bubbles will automatically appears over mouse when the cursor will pass above an item (in the Diagram pane) that has an Help section defined.

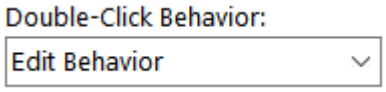
## 7 R-click Select




When checked, using the mouse right button over something selectable will select it first then apply the right button function (normally, contextual popup menu).
When unchecked (default), only contextual popup menu will be built.

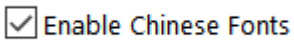

*With this mode on, calling the contextual menu over an empty map area deselect everything first. Because the Map Move function can be associated with the Right Mouse Button + Mouse Move, it is a good idea to activate the Ctrl-Key Map Move option (see Editors settings) if Right Click Select mode is activated, to avoid deselecting anything during the map panning.*

## 8 Double-Click



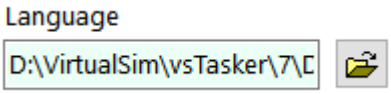

Choose action when double clicking on a behavior entity symbol.

## 9 Chinese Fonts




Activate the OpenGL Chinese symbols. Must have Windows Chinese fonts set up.

## 10 Language

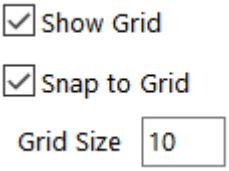



Select which resource file to use for relocation. These files can be edited using the translate.exe tool.

## 11 Translator

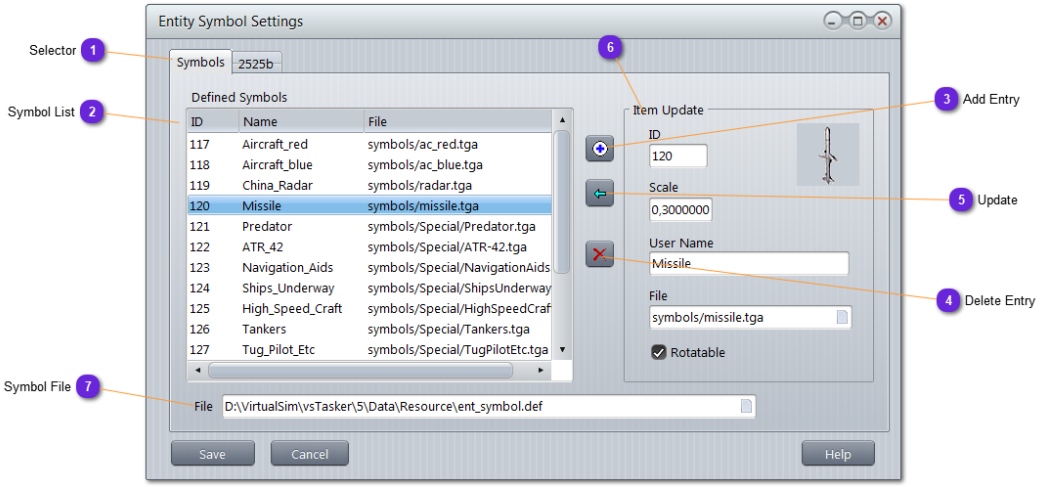

Call the Translator tool.

## 12 Diagram Grid

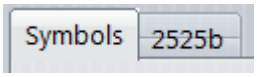



Check/Uncheck the **Show Grid** option to activate the paper grid on the Diagram panel.
Check/Unckeck the **Snap to Grid** option to force object to magnet on the below grid, or not.
**Grid Size** is expressed without unit. Bigger value for more space between lines.

# Entity Symbols

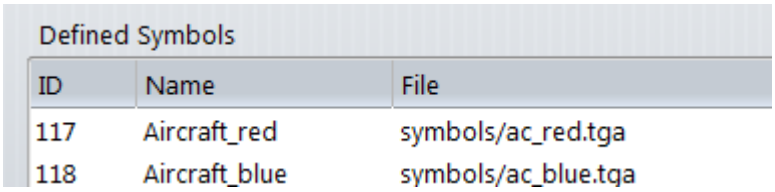



## 1 Selector




Select which database you are going to define.
They are all structured the same way, reason why they share the same window.
The file loaded are ent_symbol.def (for symbols) and ent_2525b.def (for 2525b) in Data/Resource.
Each of them look for files in Data/Resource/Symbols or Data/Resource/2525b directories.

## 2 Symbol List




This list displays all symbols extracted from the database file according to **selector** setting.

## 3 Add Entry

To **add** a new entry. Set the name and the file before using  to update the list.

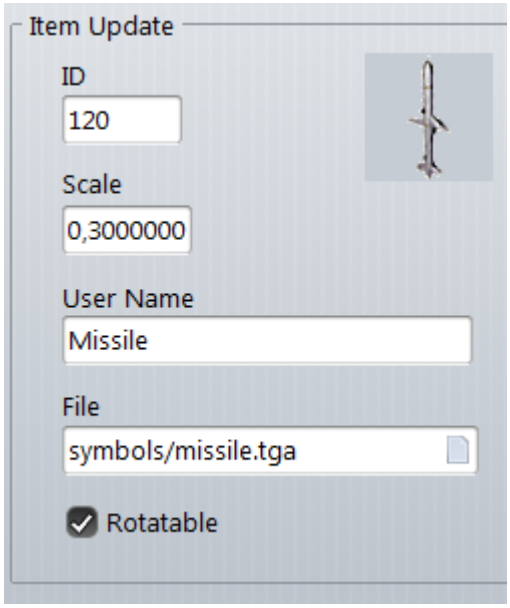
## 4 Delete Entry

To **delete** the selected entry. Be careful that this action cannot be undo. All entities using this symbol will lose the reference.

## 5 Update

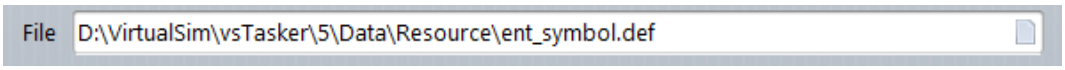
Use this button to update the entry with the modified fields in Item Update block.

## 6 Item Update



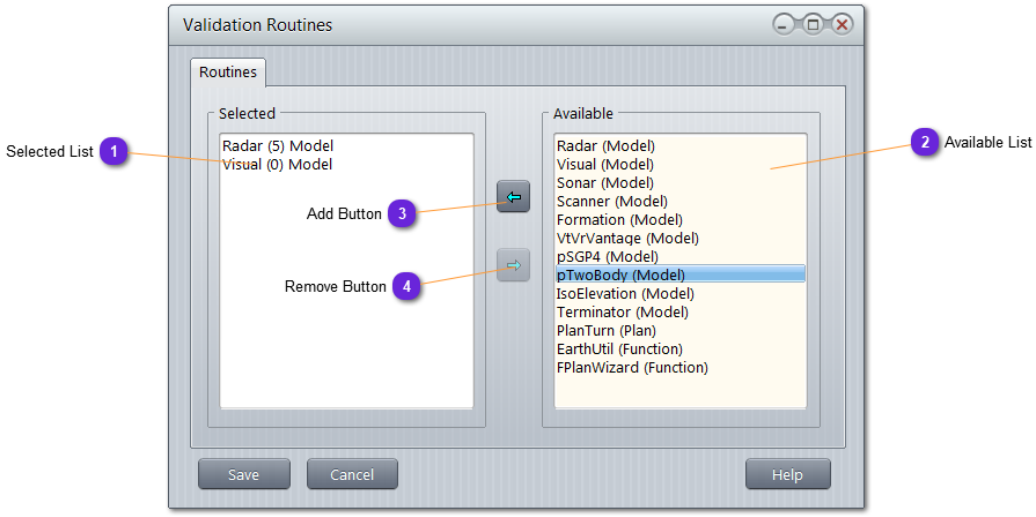

Select for adding/modify an item in/from the Defined Symbols list.
**ID**: unique ID for the symbol definition.
**User Name**: name of the symbol. Do not use space.
**Scale**: scale applied to the bitmap. 1.0 is the normal size. 0.5 is half the size.
**File**: location of the bitmap TGA file.
**Rotatable**: if checked, the bitmap will rotate according to the entity heading.

## 7 Symbol File

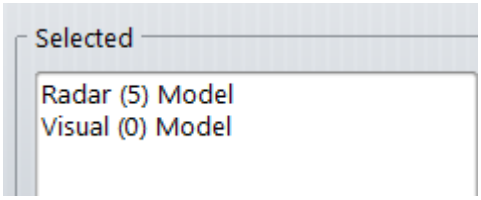



Select here the definition symbol file you intend to use in your simulation. User can create his own or expand a default one.

# Validation Routines

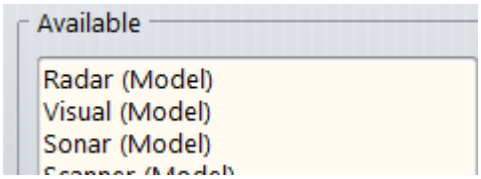



## 1 Selected List




This list shows all Validation Routines (dll) loaded and activated.
In parenthesis, the number of models that are currently using them.

## 2 Available List




This list shows all the loaded Validation Routine **dll** from the files validation.lst and validation_usr.lst in /Validation directory.
In parenthesis, the **dll** file is mentioned.

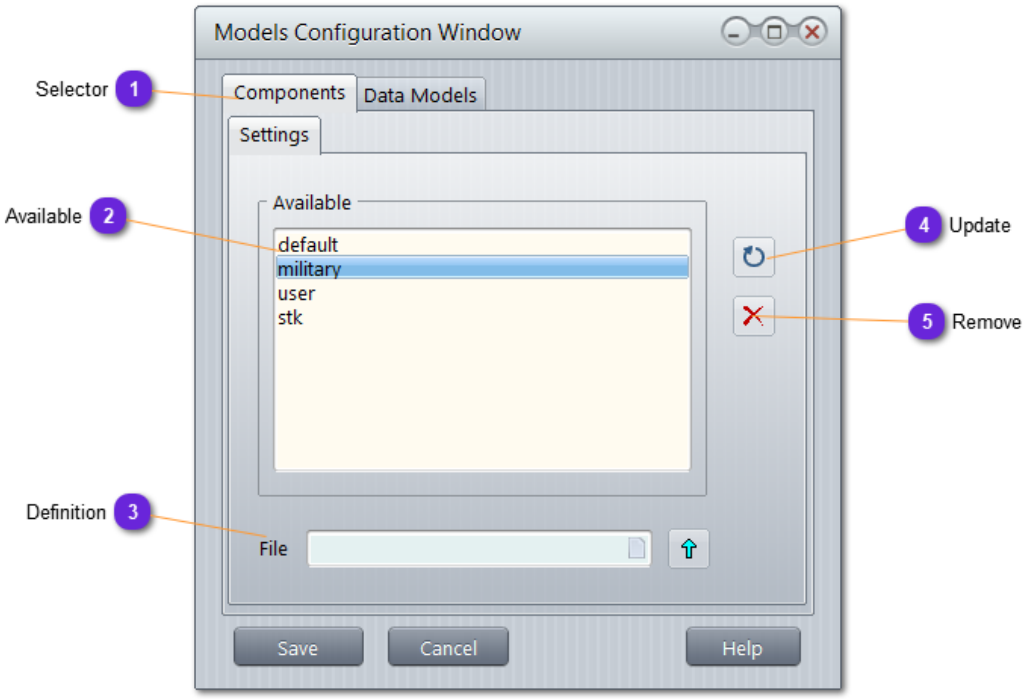
## 3 Add Button

When a Routine is selected in the Available list, this button is enabled and allows the Routine to be moved to the Selected list in order to be activated and used.

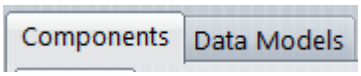
## 4 Remove Button

When a Routine is selected in the left inside list, this button is enabled to allow removing the Routine from the activate list.

# Model Configuration

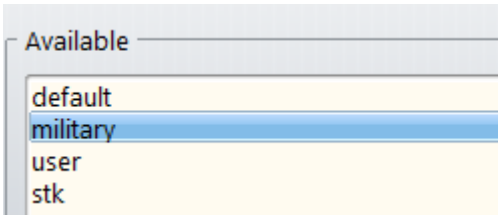



## 1 Selector




Select which package you are going to define.
They are all structured the same way, reason why they share the same window.
The file loaded are located by default in /Model directory and suffix is .cpt for Components and .mdl for Data Models.

## 2 Available




This list shows all the definition files you want to load to build the Component library.
By library, we mean a set of all items available for being moved to Model library (from file to database)

## 3 Definition


Use this field to add another definition file to the current list. vsTASKER will build the library based on the content of all these definition files.

*A library restrict what can be used by vsTASKER, according to licensing availability or third party software environment.*
*For example, if STK (from AGI vendor) is not used, there is no need to import the STK definition file.*
*Same thing for military (of defense) component library if vsTASKER is used only for ATC.*

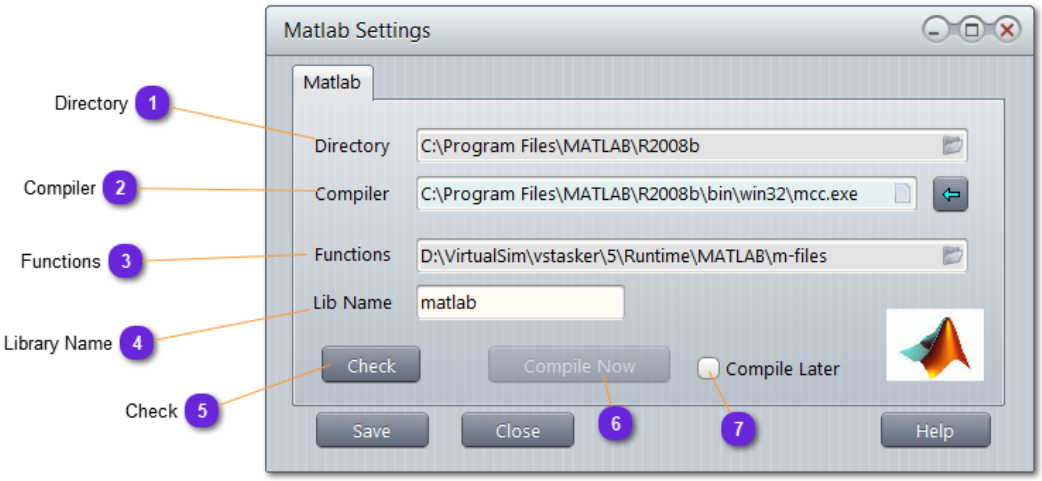
## 4 Update

Use this button to force the reload of all the definition files and the rebuild of the Component/DataModel library.
This can be useful when the definition file is modified.

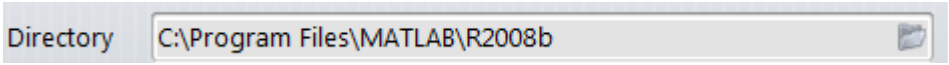
## 5 Remove

Select an entry in the Available list and remove it from the library. The definition file associated is not deleted, just the reference is discarded.

# Matlab Settings

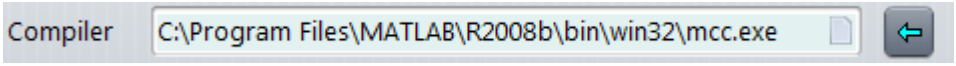



## 1 Directory




Location of the MATLAB directory to use. Normally, the environment variable `%MATLAB%` shall be used.

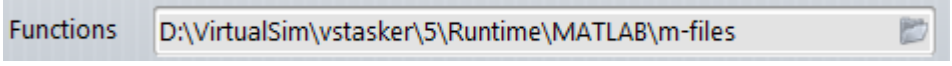
## 2 Compiler




Program used to compile the m-files (MATLAB functions).

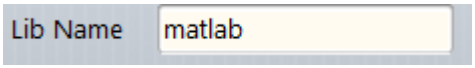
The  button will try to fill the Directory and Compiler fields based on the `%MATLAB%` environment variable.

## 3 Functions




Directory where MATLAB functions used in the database are stored or will be saved (for compiling issues as vsTASKER will build the makefile for mcc.exe)

## 4 Library Name

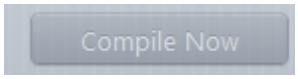



Give here the name of the library that will gather all the MATLAB functions used in the current database and compiled by mcc.exe.
This library will be used by the linker to produce the simulation engine.

## 5 Check


Use this button to check the Matlab configuration (directories, compiler, etc).

## 6 Compile



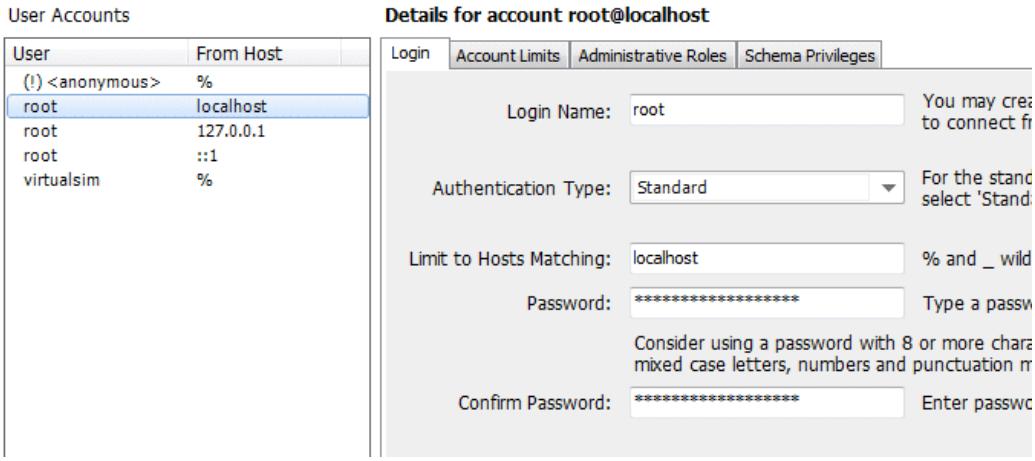

User can force the compilation of all MATLAB functions of the database and produce the library that will be used later at link time.
This can also insure that the compiler setting is correct.

## 7 Compile Later

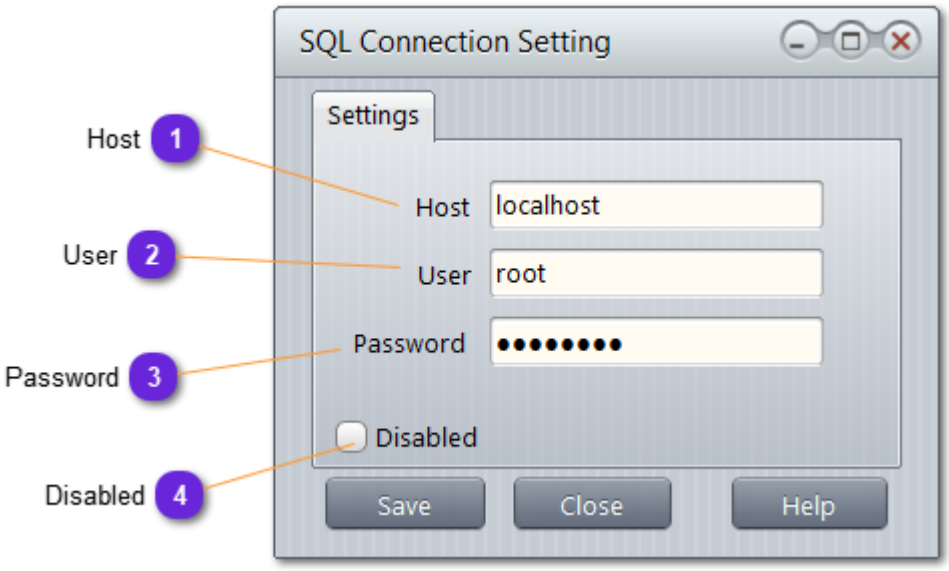

When this option is checked, the compilation of the MATLAB functions will happen at database compile time (if needed) to produce the library.
If unchecked, the compilation of the MATLAB functions will never happen (and will be only manual using the Compile Now button).

# SQL Connection

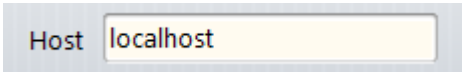
These settings are requested to connect to mySQL server.
If you use Workbench, you should get the same information as here:




Go here for more information.




## 1 Host




Name of the host of the SQL database server

## 2 User


Name of the user (account) registered or known by the SQL server.

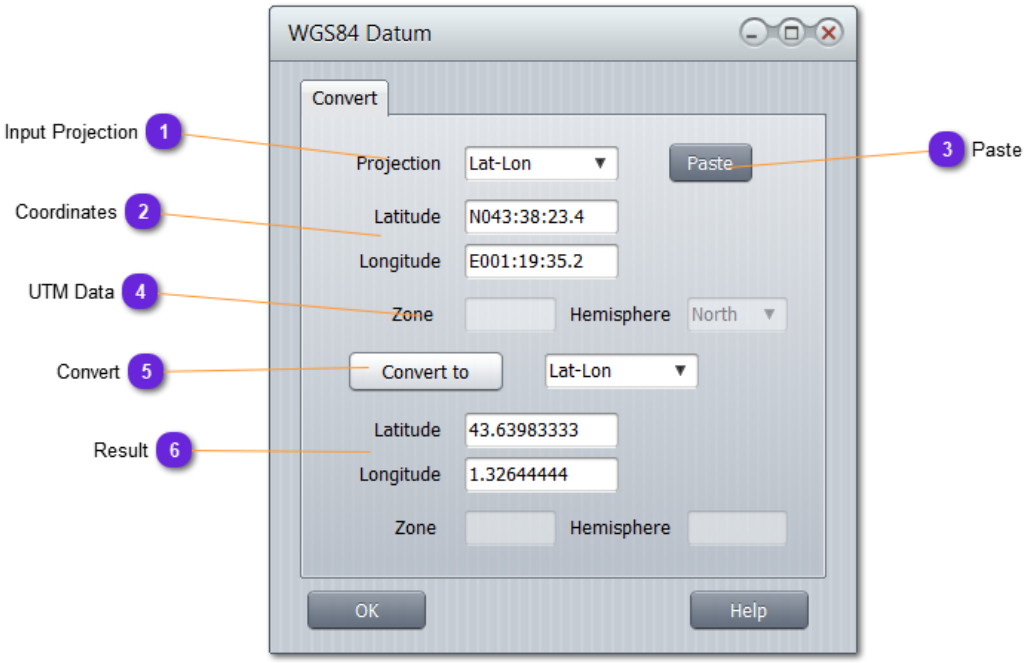
## 3 Password


Password associated with the user (account) on the SQL server.

## 4 Disabled


If checked, mySQL code will not be generated, even if tables are defined in the Database.

# Pos Converter

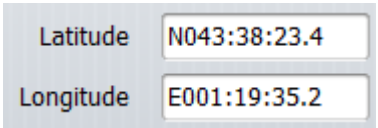



## 1 Input Projection


Select the projection type for the input coordinates:

- XY Flat earth
- Latitude/Longitude
- UTM Standard
- ECEF

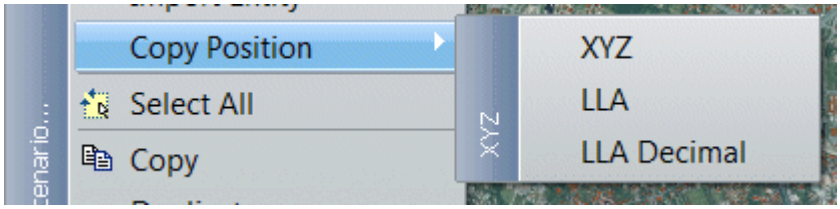
## 2 Coordinates



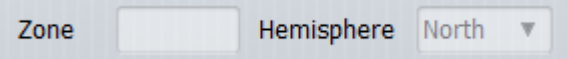

Enter here the XY or Lat/Lon or Easting/Northing coordinates to convert.

## 3 Paste


Use this button to paste here the copied coordinates on the map, using the right click option of the map:

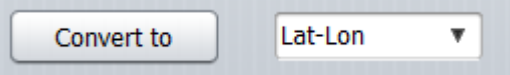



## 4 UTM Data




When the **Input Projection** is UTM, **Zone** and **Hemisphere** data is enabled and mandatory for **Easting** and **Northing** entries.

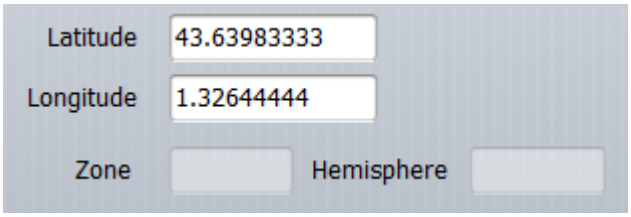
## 5 Convert




Convert the input coordinates (above) to the selected output projection:

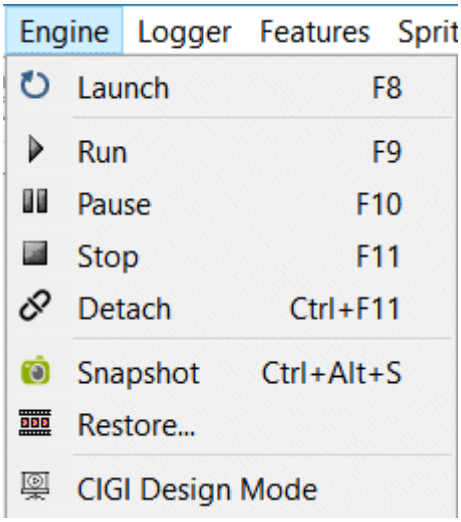
- XY Flat earth
- Latitude/Longitude in literal
- Latitude/Longitude in degrees
- UTM Standard
- ECEF

## 6 Result




Display here the converted input coordinates in the selected **Output Projection** (5).

# Engine




**Launch**: Load the simulation engine associated with the current database. See database settings to learn how the simulation executable is located and named.
If the engine is already loaded, will unload it (terminate the process).

**Run**: Once the simulation is loaded, the Run button can be used to start the simulation.

**Pause**: Once the simulation is running, pause will freeze the simulation (HLA thread will continue running but nothing will be pooled)

**Stop**: Terminates the simulation.

**Detach**: Unload the simulation engine and properly unlink the shared-memory.

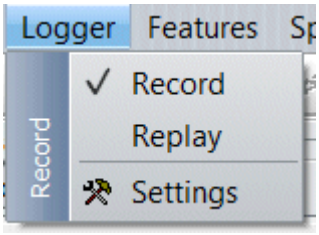
**Snapshot**: will immediately dump all the current memory to a file for later restore.

**Restore**: call the Restore manager window.

**CIGI Design Mode**: switch ON or OFF the CIGI connection with third party IG. See here for more explanation.

# Logger

The Logger records and replay entity positions, attitude, speed, heading (...) and some events (like entity creation/deletion) allowing the scenario to be replayed like a VCR.
During the replay, models are not running so, it is possible that the review (forward or backward) will not produce the same outputs are normal simulation run, mostly if external 3D engine is used and controlled by components, or if HLA/LAN means of communication are used to broadcast the entity positions.




Select the **Record** option to record the simulation when **Start** is pressed.

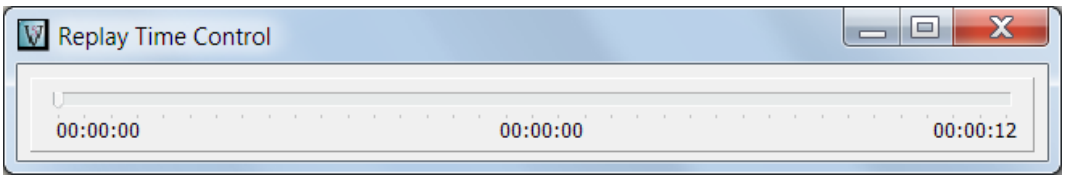
Select the **Replay** option to replay the recorded simulation using the **Replay** time bar.

**Settings** window for Record and Replay modes.

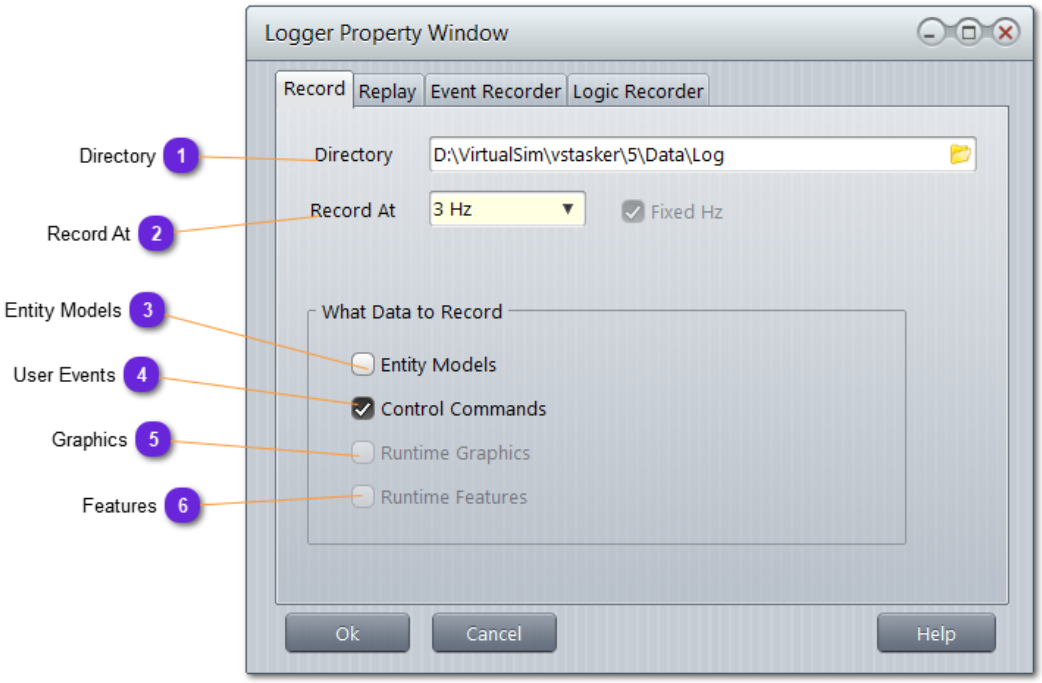
When the Record mode is activated, the **Terrrain Map** displays the following header:


When the Replay mode is activated, the **Terrain Map** displays the following header:

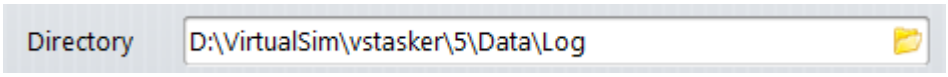

In Replay mode, the **Replay Time Bar** is visible and can be used to **fast-forward** or **fast-backward** (or jump in time) the simulation like a VRC.
First, **Start** the simulation to engage the Replay. The **Time Bar** cursor will start moving.
Grab it with the mouse and move it along the scale. The **Terrain Map** will update the Entity positions accordingly.
You can Pause/Stop the Replay the usual way.



# Record




## 1 Directory

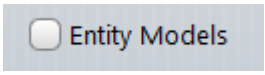



Set here the location of all the log files.
A Log set is made of four files:
<database_name>_<number> then the four extensions: ent, evt, log, rec

## 2 Record At

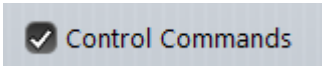

Specify here the frequency at which the entity position must be recorded.

## 3 Entity Models



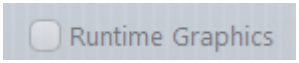

If this option is checked, the Entity **components** (attached to each Entity) will be called to store specific data.
At *Replay*, these **components** will also be called with the stored data, in a special function.
As no **component** is running, the processing of this data will be left to the user code.

## 4 User Events



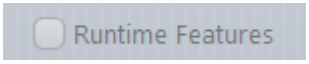

If this option is checked, all user-defined Events will be time stamped and recorded. At replay (and only going forward), these Events will be triggered.
As no **logic** nor **component** are running, the processing of these Events will be left to the user code.

## 5 Graphics



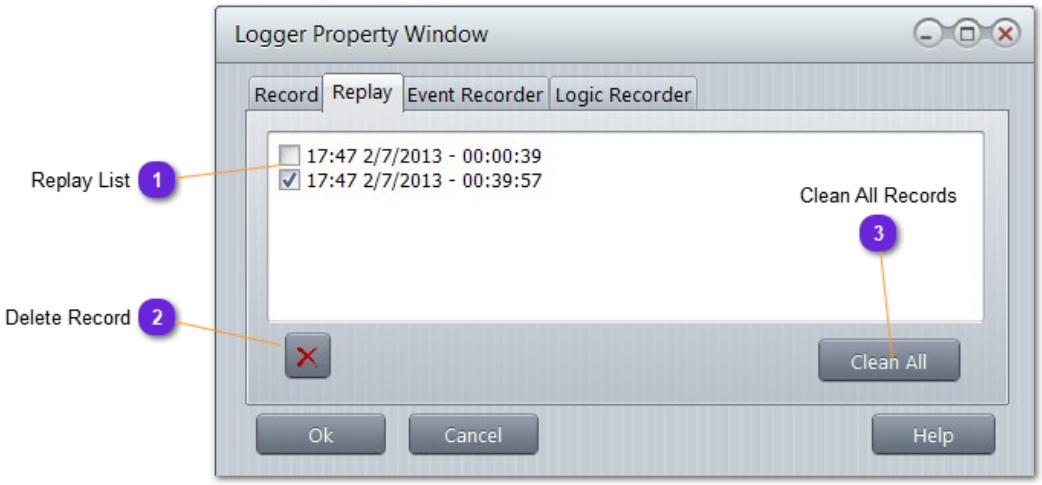

Option to record and replay all Graphic commands sent from the SIM to the GUI (sensor shape for i.e)
*Not available yet.*

## 6 Features

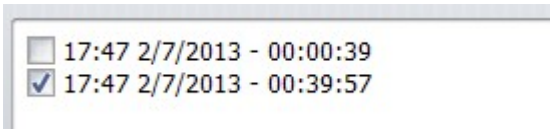



Option to record and replay all runtime Features initiated from the SIM to the GUI (new trajectory or special zone for i.e)
*Not available yet.*

# Replay




## 1 Replay List

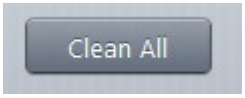



This panel lists all the recorded sessions for the current database and the selected scenario.
The checked one will be the one used at Replay.

## 2 Delete Record

Use this button to delete the selected Log set (all four associated files will be removed).

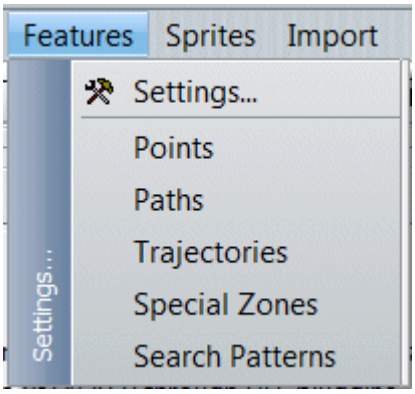
## 3 Clean All Records




Use this button to remove all Logs of the current database (same effect as selecting each entry and using the delete button.

# Features

For this menu to be active, Features panel must be selected for the Terrain display.




**Settings**: call the Feature setting window

**Points**: call the Point manager window

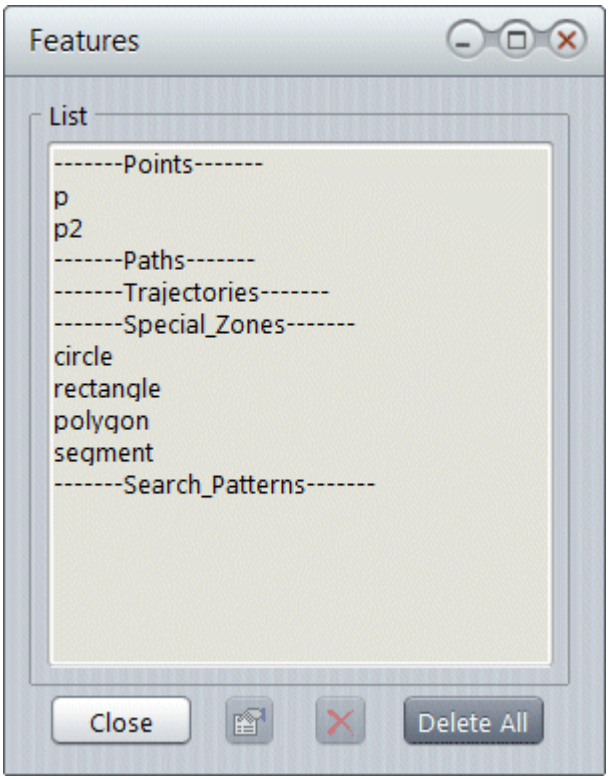
**Paths**: call the Path manager window

**Trajectories**: call the Trajectory manager window

**Special Zones**: call the Special Zone manager window

**Search Patterns**: call the Search Patterns manager window

The Feature Manager window lists all typed features for the current scenario.

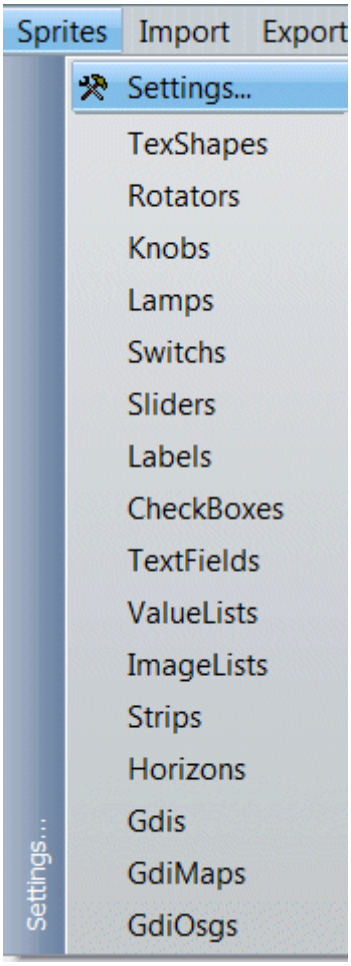



**Delete All** button will delete all defined features in the scenario.

Go to the Features chapter to learn more on this topic.

# Sprites

For this menu to be active, **HMI** panel must be selected from the Map display




**Settings**: call the HMI setting window

**TexShapes**: call the Textured Shape manager window

**Rotators**: call the Rotator manager window

**Knobs**: call the Knob manager window

**Lamps**: call the Lamp manager window

**Switchs**: call the Switch manager window

**Sliders**: call the Slider manager window

**Labels**: call the Label manager window

**CheckBoxes**: call the CheckBoxe manager window

**TextFields**: call the TextField manager window

**ValueLists**: call the ValueList manager window

**ImageLists**: call the ImageList manager window

**Strips**: call the Strip manager window

**Horizons**: call the Horizon manager window

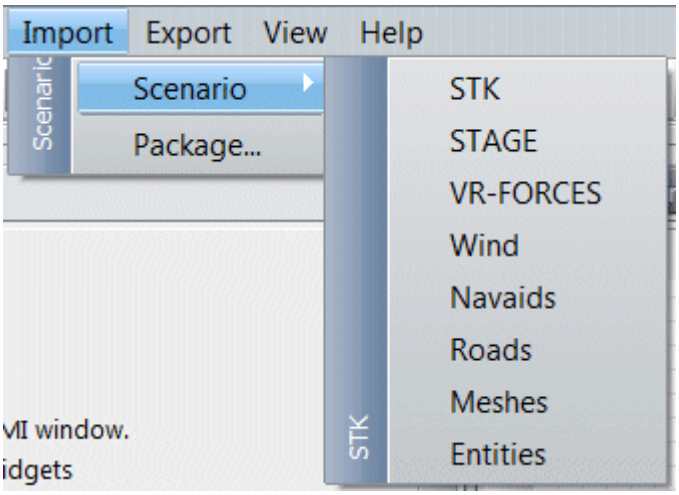
**Gdis**: call the Gdi manager window

**GdiMaps**: call the GdiMap manager window

**GdiOsgs**: call the GdiOsg manager window

Go to the Sprite/HMI chapter to learn more on this topic.

# Import




**STK**: call the importer window

**STAGE**: call the importer window

**VR-Forces**: call the importer window

**Wind**: see below

**Navaids**: see below

**Roads**: call the importer window

**Meshes**: see below

**Entities**: see below

## • Wind

The Wind importer loads a tabular definition file as available in Import/Wind/weather_grid.txt and creates the 3D **Wind zone** in the scenario.
When the importer is called, the file selection dialog pops up to select the file to load. If the format of the file differs from the sample one, imp_wind.cpp can be modified and the DLL imp-wind.dll must be rebuild using CodeGear C++ Builder.

## • Navaids

The Navaids importer parse a given file to import all defined navaids with the following data per line, separated per tab:

```
int     station_id,
int     type,
int     nav_id,
double  lat,
double  lon,
double  alt,
double  properties,
double  coverage,
```

```
      int     status,
      double  frequency
```

```
type can be on any of these values:
      1: VORTAC
      2: DME
      3: VOR
      4: TACAN
      5: ILS
      6: NDB
      7: ATC
      8: TIS-B
```

## • Meshes

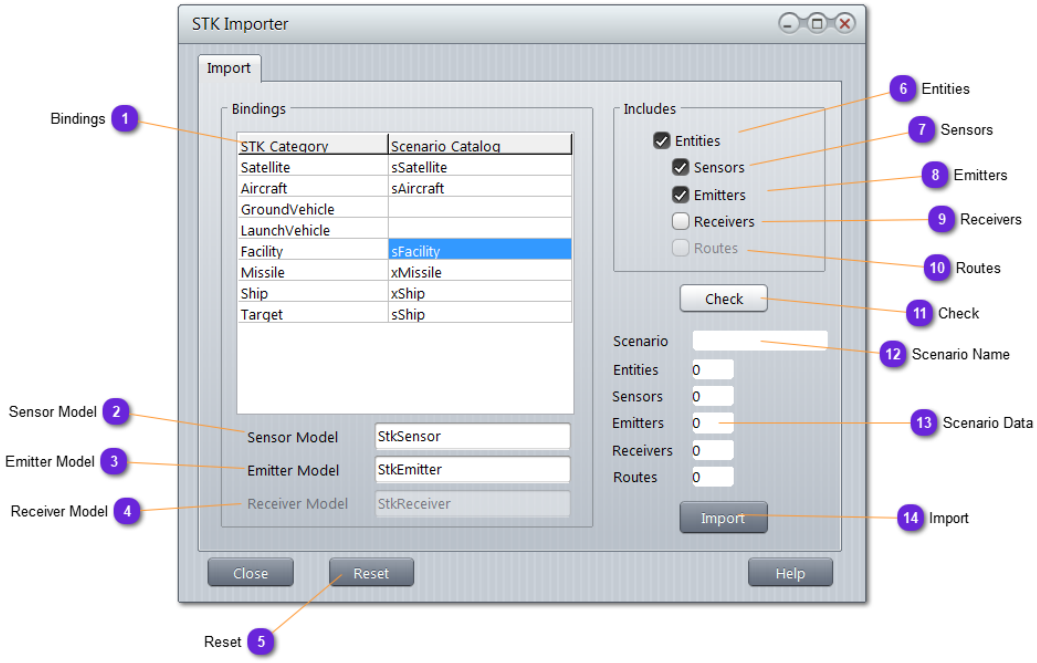
The Mesh importer is here a sample showing how to create from C code a new **Mesh** (or many) and to add them into the scenario.
See the file Import/Meshes/imp_mesh.cpp for explanations. The DLL imp-wind.dll must be rebuild using CodeGear C++ Builder.

## • Entities

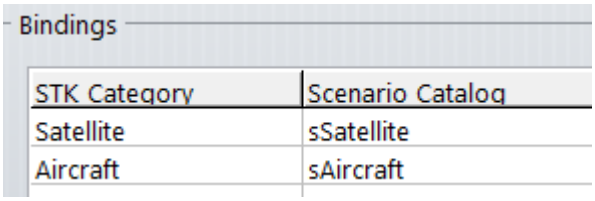
Imports into the current scenario all **Entities** defined in the Import/Entities/test.txt file, whose format has been predefined.
When the importer is called, the file selection dialog pops up to select the file to load.
If the format of the file differs from the sample one, imp_entities.cpp can be modified and the DLL imp-entities.dll must be rebuild using CodeGear C++ Builder.


*To remove one importer from the list, open the file Import/import.lst and change the first character 1 to 0 in the line containing the importer to remove.*
*To add a new Importer/Exporter, refer the the following chapter*

# STK Importer



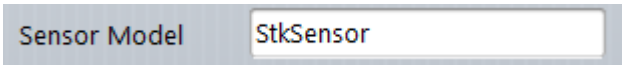

## 1 Bindings




User must pair for each STK Category the corresponding Catalog Entity in vsTASKER.
With this binding, whenever an STK object of this category will be found in the STK scenario, the importer will create a vsTASKER entity at the same location and of the same type, using the corresponding Catalog Entity.

When the STK category is not paired, the object will not be imported (and a warning message will be issued).

## 2 Sensor Model

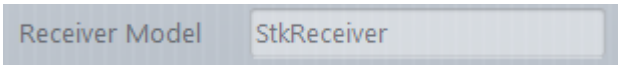



Name here the component that will be used as the interface between the STK Sensor and the vsTASKER environment.
vsTASKER provides built-in components for basic interfacing.

## 3 Emitter Model


Name here the component that will be used as the interface between the STK Emitter and the vsTASKER environment.
vsTASKER provides built-in components for basic interfacing.

## 4 Receiver Model

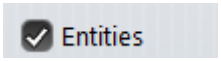



Name here the component that will be used as the interface between the STK Receiver and the vsTASKER environment.
vsTASKER provides built-in components for basic interfacing.

## 5 Reset

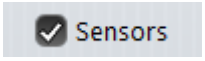

Tries to set default (factory) values to most of the window fields.

## 6 Entities



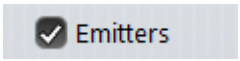

When this option is enabled, the Importer will look for all STK scenario entities.

## 7 Sensors



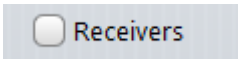

When this option is enabled, the Importer will add all STK Sensors belonging to each imported entity.
The Sensor Model (see 2) component will be used.

## 8 Emitters



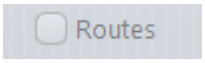

When this option is enabled, the Importer will add all STK Emitters belonging to each imported entity.
The Emitter Model (see 3) component will be used.

## 9 Receivers



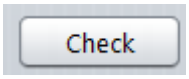

When this option is enabled, the Importer will add all STK Receivers belonging to each imported entity.
The Receiver Model (see 3) component will be used.

## 10 Routes




*Not available*

## 11 Check

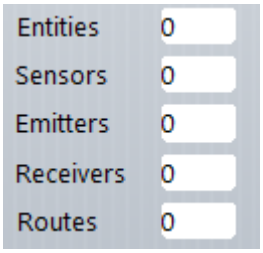



This button will run the importer with the current settings and will display the result of the queries in the text field below.

## 12 Scenario Name

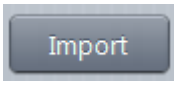

When Check (or Import) button is depressed, the Importer will query the running instance of STK and will display here the STK scenario name.

## 13 Scenario Data




When Check (or Import) button is depressed, the Importer will query the running instance of STK and will list here the total numbers of listed items found.

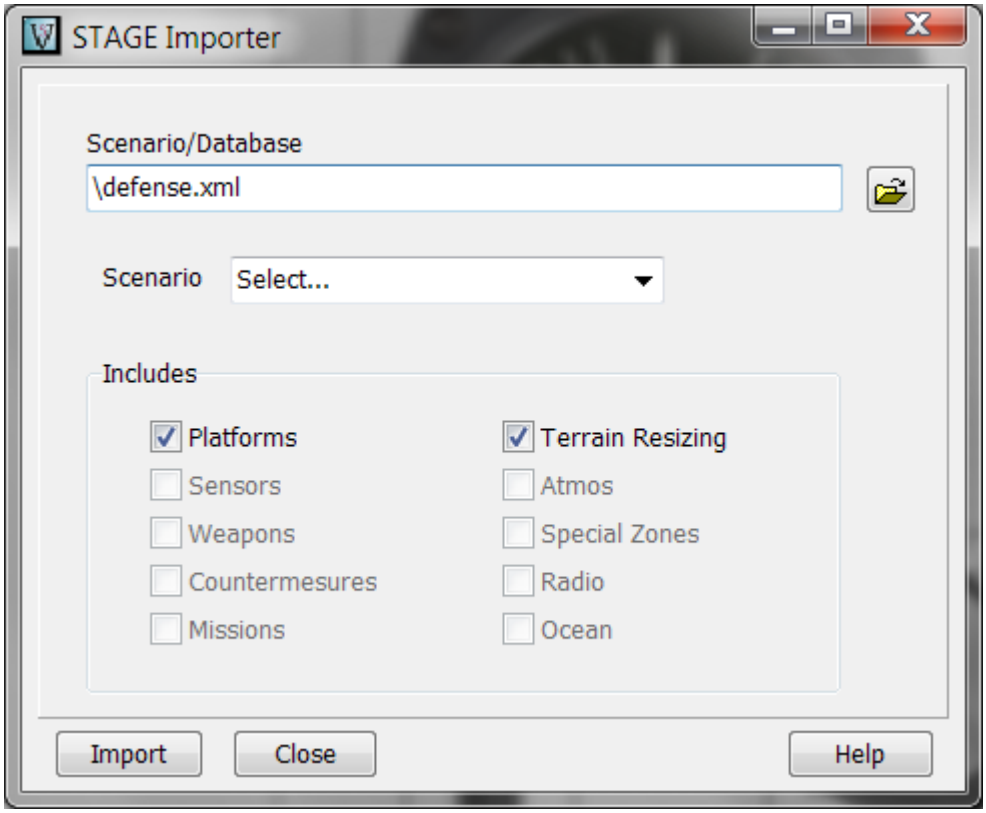
## 14 Import




This button will run the importer with the current settings and will add in the current vsTASKER scenario all STK available entities.

# STAGE Importer

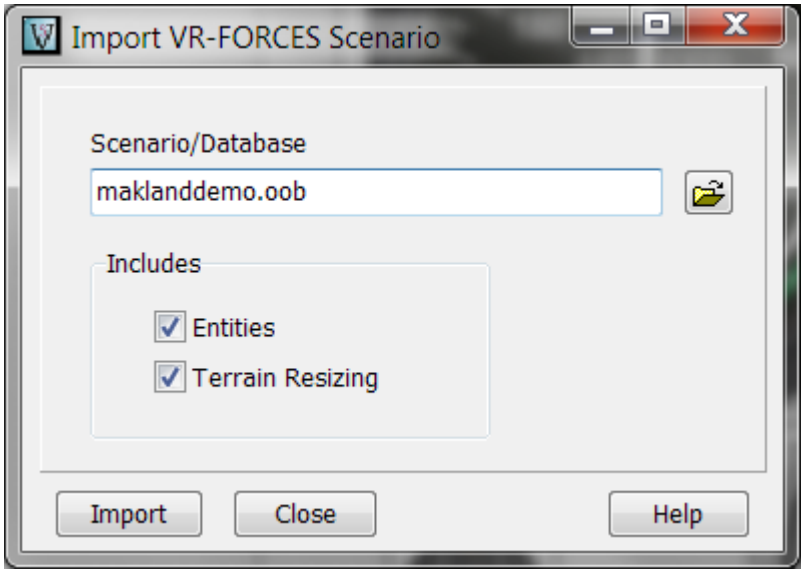
Specify the STAGE scenario database you want to import.
Import is only for Entities position, name and some basic attributes.




The importer loads the xml database and extracts all selected data in the Includes part.
After the import, user can start creating Logic and Behaviors for any imported Entity in order for vsTASKER to produce a STAGE compatible user-module.

# VR-FORCES Importer

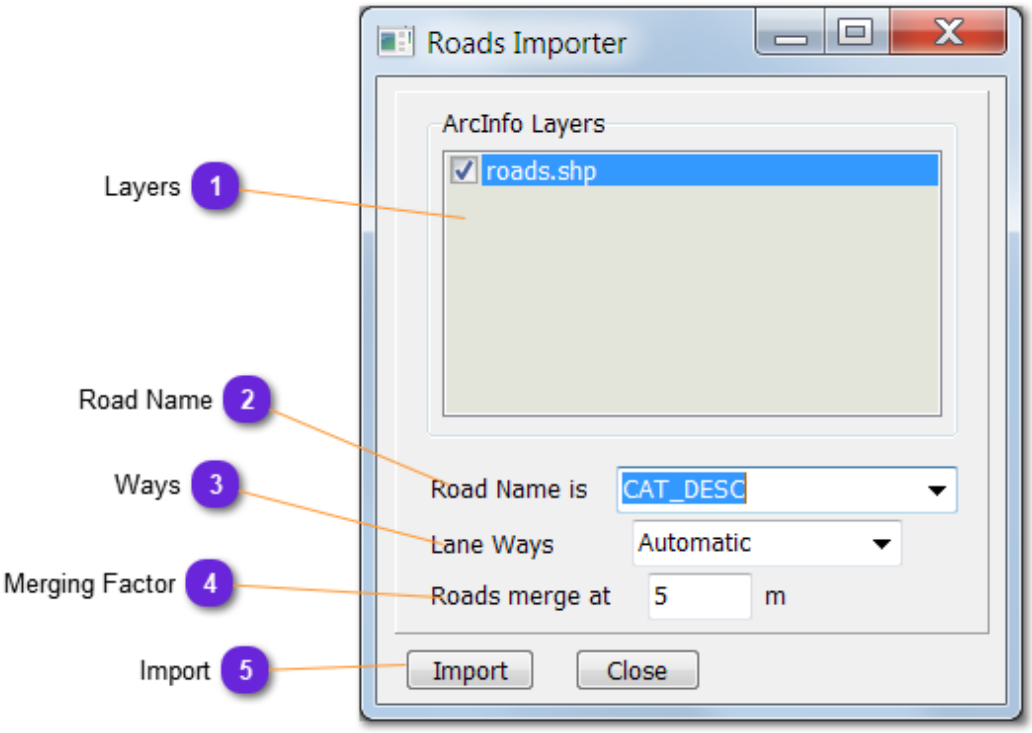
Specify the VR-FORCES scenario database you want to import.
Import is only for Entities position, name and some basic attributes.



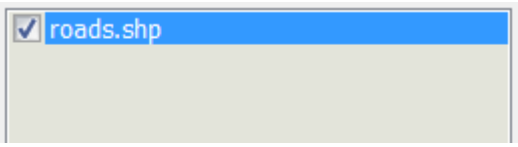

The importer loads the oob database and extract entity positions and terrain dimensions.
After the import, user can start creating Logic and Behaviors for any imported Entity in order for vsTASKER to produce a VR-FORCES compatible module.

# Roads Importer window

This importer need an Arc/Info database loaded in order to extract the roads from one or many layers and import them into vsTASKER.




## 1 Layers

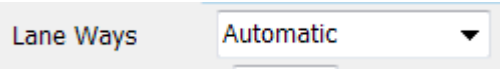



List all the layers of the ArcInfo database loaded. Empty if not loaded.

## 2 Road Name

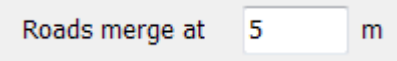

For the selected Layer (above), all the categories are listed into this list-down menu.
Select the one to be considered for naming the roads with the content of the category.
If nothing is selected (Automatic), roads will be named "Road #x" with x a number.

## 3 Ways




When Automatic is selected, the importer will try to guess the way for each segment found.
Otherwise, force each segment to be one-way or both-ways only.

## 4 Merging Factor



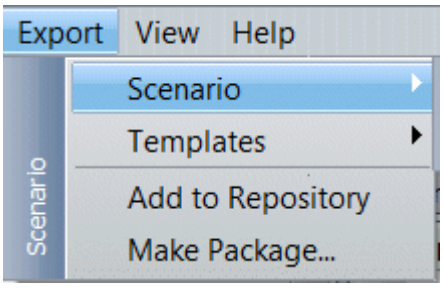

Specify the minimum distance between two ArcInfo points to make a road break.
The shorter the value, the more complex will be imported roads, meaning, more memory and more CPU consumption for path-finding, but more accuracy also.
The bigger the value, the lesser the accuracy.

## 5 Import


Click this button to import all roads based on the current setting.
Do not press it twice as all roads will be duplicated (with different names).


*It is mandatory to fix the database after import. Do Tools::Fix Database and then save.*

# Export



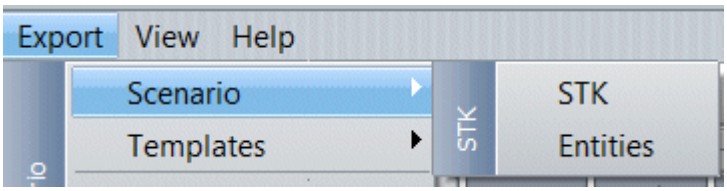

**Scenario**: see this topic

**Templates**: see the Template Exporter

**Add to Repository**: Move whatever is selected in the Diagram to the Repository Category, if possible and if not existing already.

**Make Package**: *not available yet*

# Scenario




**STK**: see the STK Exporter

**Entities**: see below

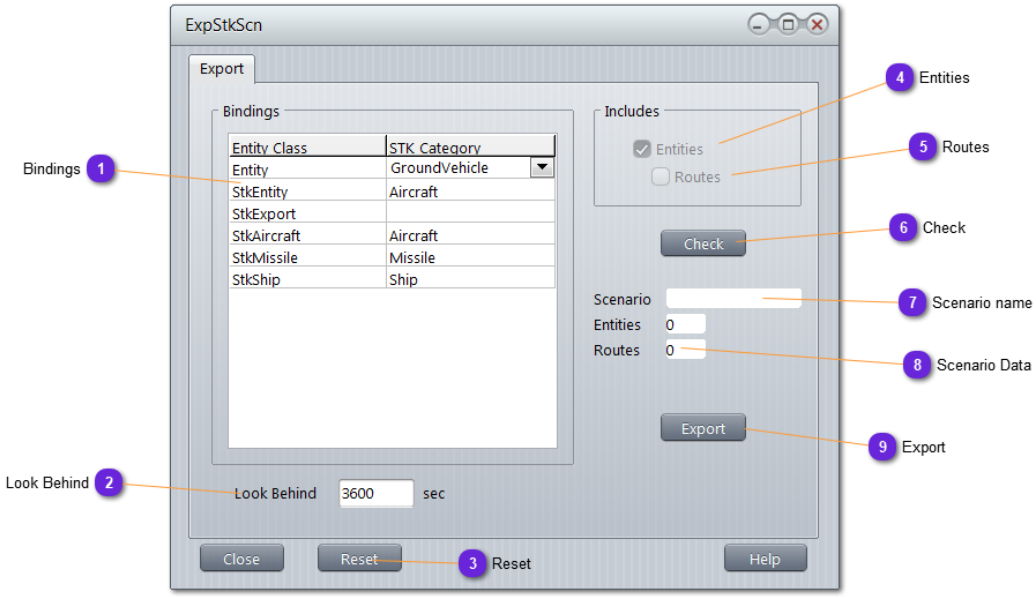
## Entities

Will export all the current (selected) Scenario **Entities** to a file (as entered by the user when the window pops up) according to the following format:

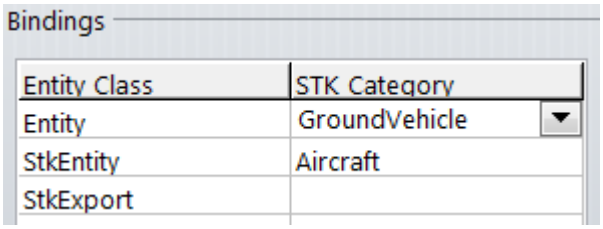
```
// name, catalog, lat/x, lon/y, alt, heading, speed
```

This format is also valid for importing.
If the format of the file differs from the sample one, imp_entities.cpp can be modified and the DLL imp-entities.dll must be rebuild using CodeGear C++ Builder.
Just refer to the function `DLL_EXPORT zoneImportFunc(ScnRep* sel_scen, IEOptions* options)`

# STK Exporter



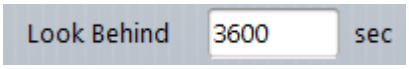

## 1 Bindings




User must pair for each vsTASKER Entity Class the corresponding STK Category.
With this binding, whenever a vsTASKER Entity will be found in the current scenario, the exporter will create an STK entity at the same location and of the same type as defined in the table.

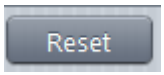
When the vsTASKER Entity Class is not paired, the Entity will not be exported.

## 2 Look Behind




Will set STK with this value for runtime use.

## 3 Reset

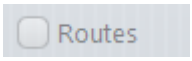



Tries to set default (factory) values to most of the window fields.

## 4 Entities

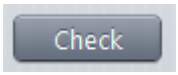

The importer will parse all Entities of the current scenario, and for each of them belonging to a Class bound with an STK Category, will create a corresponding STK Entity.

## 5 Routes




*Not available*

## 6 Check

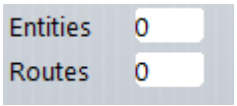



This button will run the exporter with the current settings and will display the result of the export in the text field below.

## 7 Scenario name

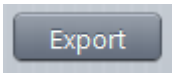

Name of the current scenario as it will be reflected in the STK running instance.

## 8 Scenario Data




Lists the total number of objects that will be exported according to the current settings.

## 9 Export

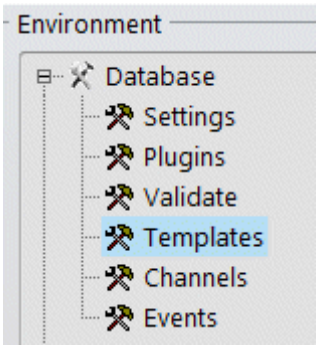



This button will run the exporter with the current settings and will add in the remote STK instance exportable entities.

# Templates

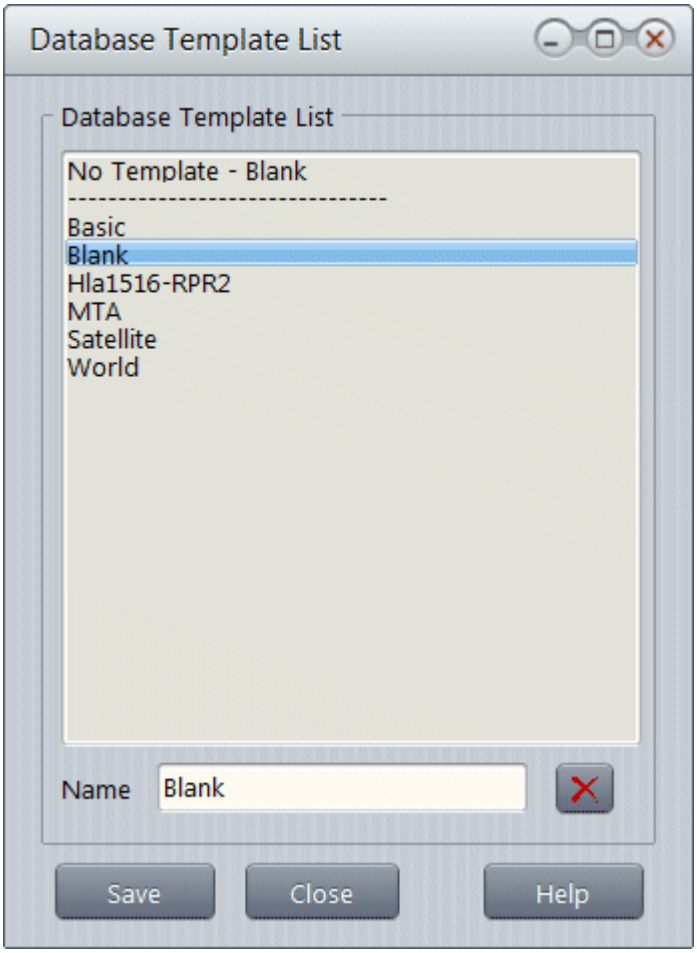
## • Config

Use this panel to remove some templates by selecting them in the list and using the  button.
Note that the template window is according to the level selected (see Templates chapter).
To manage the Database templates, select database in the Environment tree-list then use the Export::Templates::Config... menu.



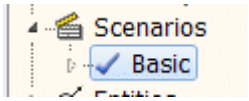

or from:


to get the Template window:

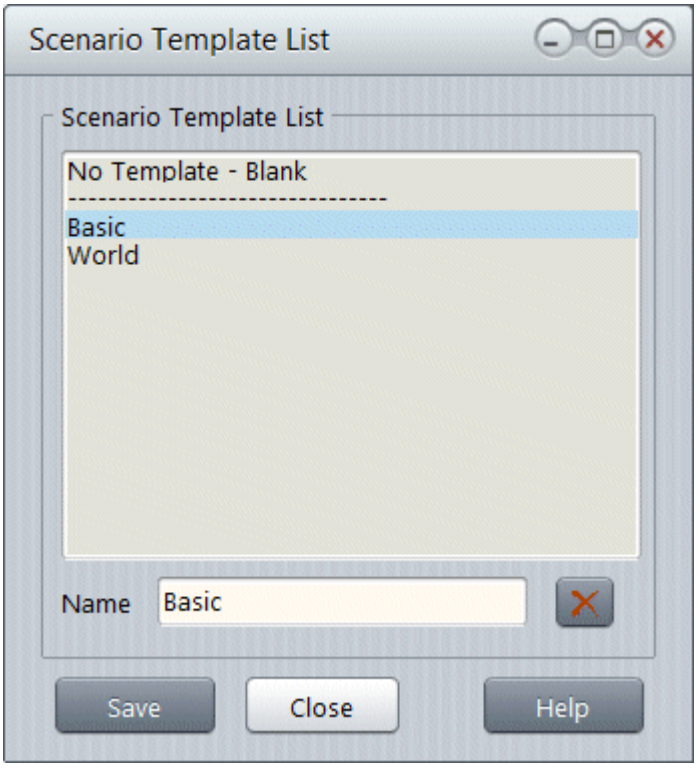



- **Save As**

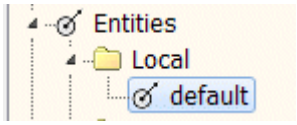
Use this menu to save the template of the current selected layer.
If you select the Database (see above), the Save As menu will allow you to save the database template or use another name.
If you select a Scenario (see below), the Save As menu will allow you to save the scenario template or use another name inside the active database template.

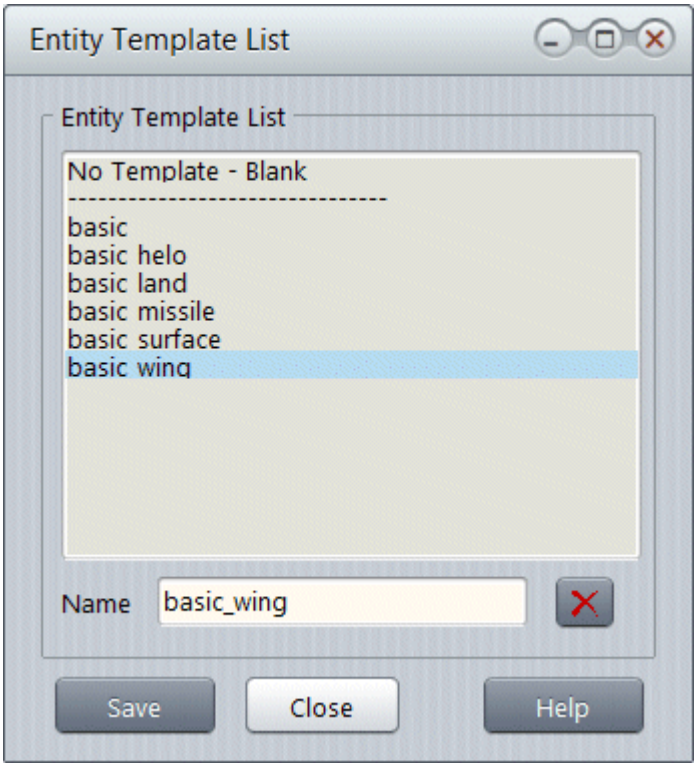


### Templates

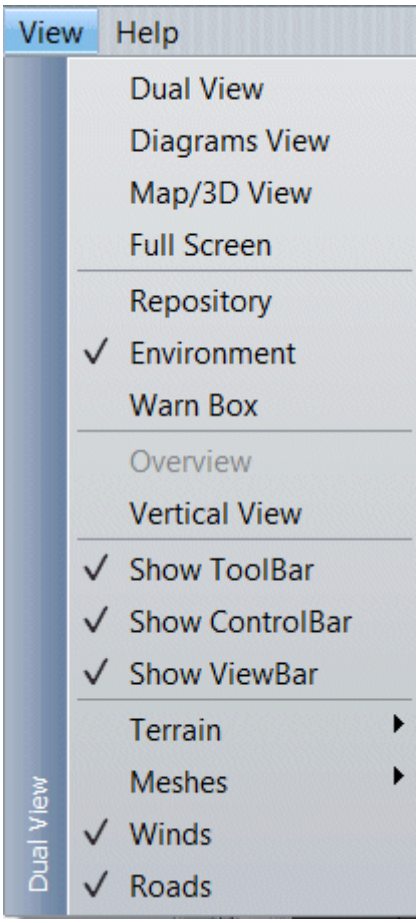



If you select an Entity (see below), the Save As menu will allow you to save the entity template or use another name inside the current scenario template of the active database.




Entity Template List
Entity Template List
No Template - Blank
basic
basic helo
basic land
basic missile
basic surface
basic wing
Name
basic_wing
Save
Close
Help

# View




**Dual View**: enable on the Display panel the Diagram (left) and the Terrain (right)

**Diagrams View**: enable on the Display panel only the Diagram view

**Map/3D View**: enable on the Display panel only the Terrain view

**Full Screen**: remove the Environment tree-list and the Message panel (bottom).

**Repository**: show/hide the Repository panel (right)

**Environment**: show/hide the Environment tree-list

**Warn Box**: show/hide the Message box (bottom)

**Overview**: *not available*

**Vertical View**: overlay on the Terrain map a vertical window transparent panel.

**Show Toolbar**: show/hide the Main application toolbar.

**Show Control bar**: show/hide the Runtime application toolbar.

**Show View bar**: show/hide the View application toolbar.

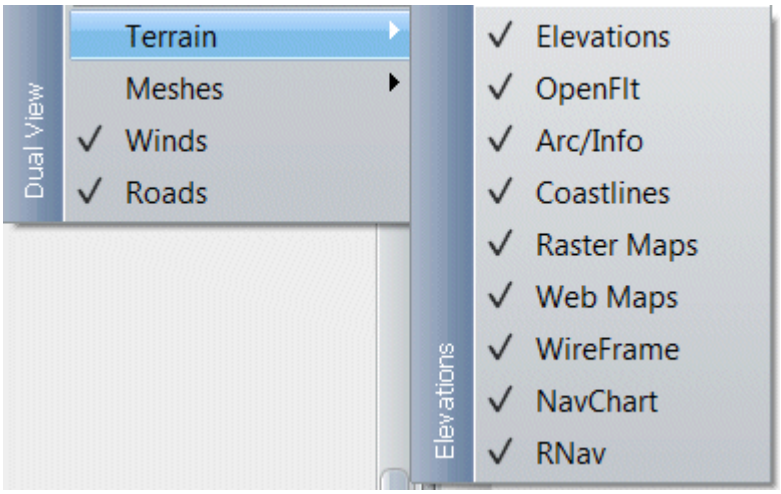
**Terrain**: see below

**Meshes**: see below

**Winds**: show/hide the Wind Areas (if any defined)

**Roads**: show/hide the Road Network (if any defined)

- **Terrain**




**Elevations**: show/hide Elevation/DTED layer.

**OpenFlt**: show/hide Open-Flight layer.

**Arc/Info**: show/hide Arc/Info layer.

**Coastlines**: show/hide Coastlines layer.

**Raster Maps**: show/hide bitmap and tiled maps layers.

**Web Maps**: show/hide imagery from web map services

**WireFrame**: show/hide wire-framed layers.

**Nav Charts**: show/hide Navigation chart (s57) layer.

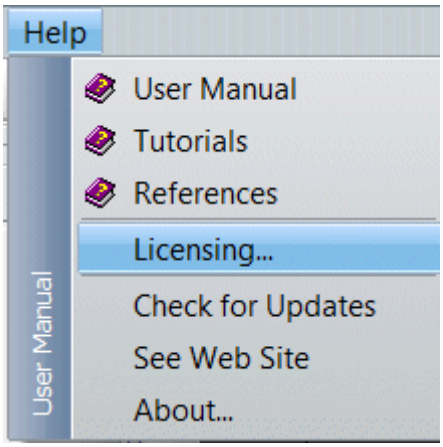
**RNav**: show/hide ARINC 424 RNav database layer.

## • Meshes


**Material**: show/hide Material based meshes.

**Proximity**: show/hide Proximity type meshes.

# Help




**User Manual**: show this document.

**Tutorials**: call outdated tutorial (from previous versions. Some are still useful)

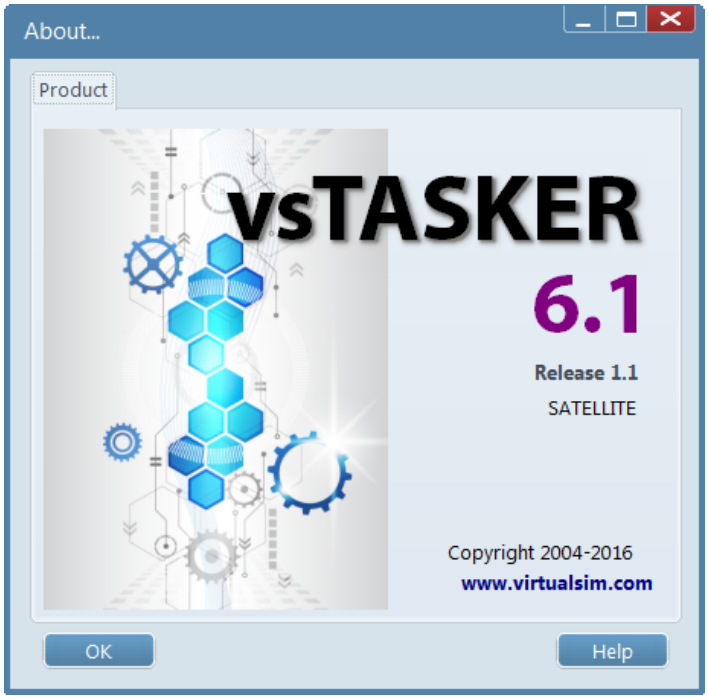
**References**: call the Reference Manual generated using Doxygen.

**Licensing**: display the licensing window. Refer to this section.

**Check for Updates**: tries to contact VirtualSim web-site to check if a new update or release is available for the product. Internet connection is mandatory.

**See Web Site**: will open the Internet browser on www.virtualsim.com

**About**: pops up a window with the current version details.



# Database

Databases are normally located on data/db for the one vsTASKER can work with, and in data/samples for the demo one (read-only, must be saved in data/db in order to be modified and saved).

Each database contains everything listed in Environment tree list of the Main Panel. A database represents the simulation environment. It holds all the classes, settings, models, objects, scenarios, etc. to generate code for a specific simulation target.

For example, let's say you want to develop a simulation system for an external device using a driver interface.

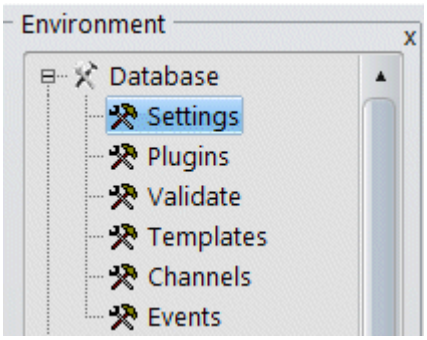
You must create a database with the name of this device or environment.
Then, you will specialize the different classes (Global, Scenario, Entity) and add as many model Components as requested, plus a Viewer (predefined or user-defined) in order to create scenarios.

Then, from inside the database, you will be able to create several scenarios. The code generated will encompass all database dependant code and classes.

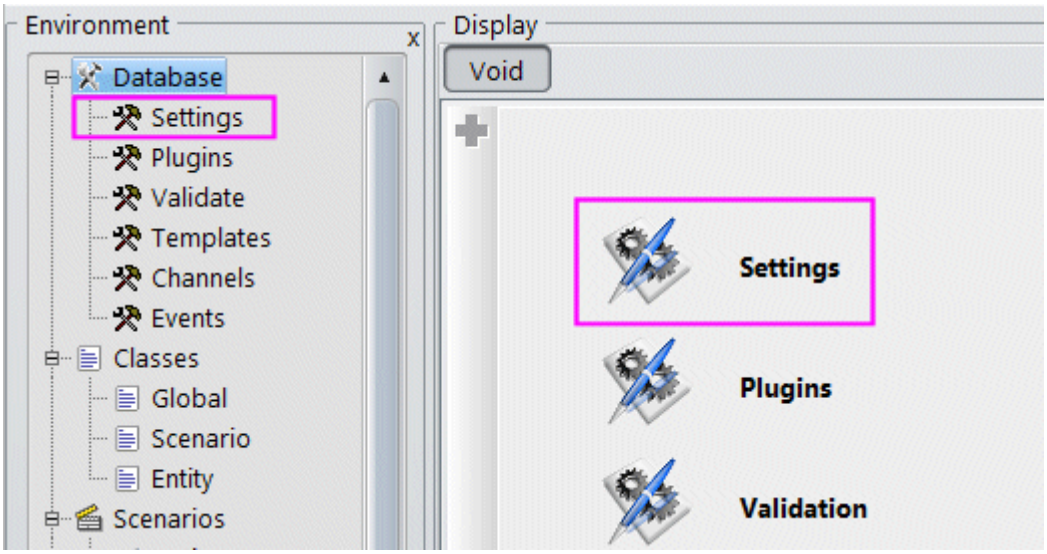
With database paradigm, there will be no dependency problems between disparate environments. Database can also generate Templates to facilitate new scenarios for the same environment.

# Database

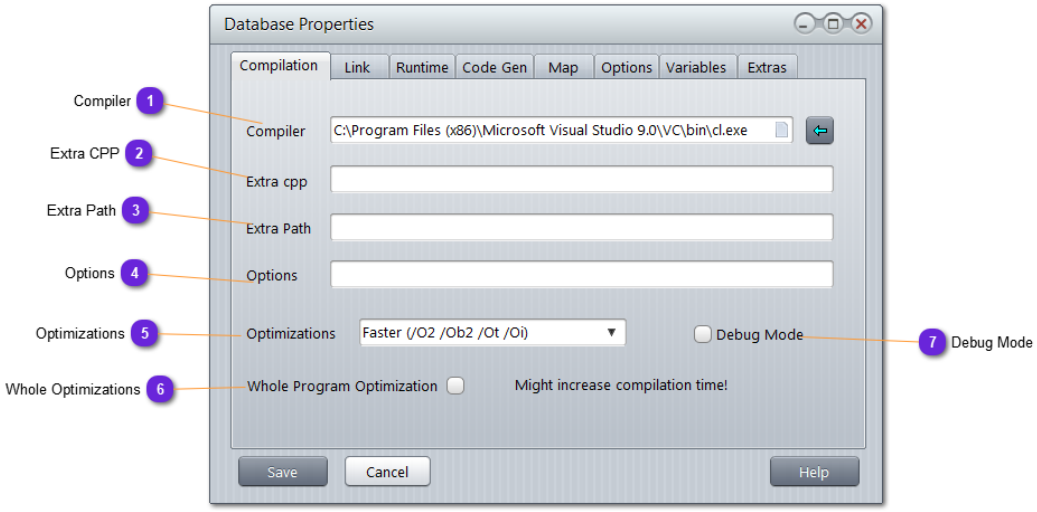
This window is used for the current database settings.
You can display it either from the Environment tree:



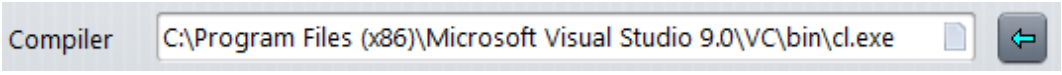

or from the Diagram panel is displayed:



# Compilation




## 1 Compiler




Use to set the Compiler to be used for this database. Use ... to select the cl.exe file.

Replaces the current compiler by the one specified in Options. The default value is taken from the Option setting but can be changed here if needed.

## 2 Extra CPP


If external C++ modules must be compiled and linked with the generated code, they must be listed there, separated by commas.
These C++ modules are typically glue modules with 3rd party applications or drivers.

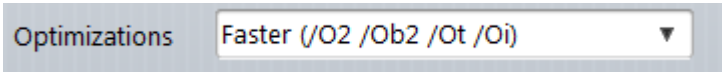
## 3 Extra Path


Paths used by the compiler to find the extra C++ modules listed above. If several, must be separated by commas.

## 4 Options


Compiler options added by the user.

## 5 Optimizations

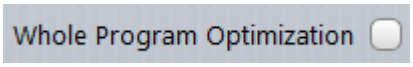



Select here the Optimizations you like for the target simulation engine.

**None:** only gets the inline function as specified by the user.
**Safe:** allow inline best candidate to be taken. speed is favor over size.
**Faster**: force speed optimizations.

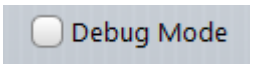
Sometimes, according to the libraries vsTASKER is linked against, Faster mode does not work and Safe or None must be chosen instead.

## 6 Whole Optimizations



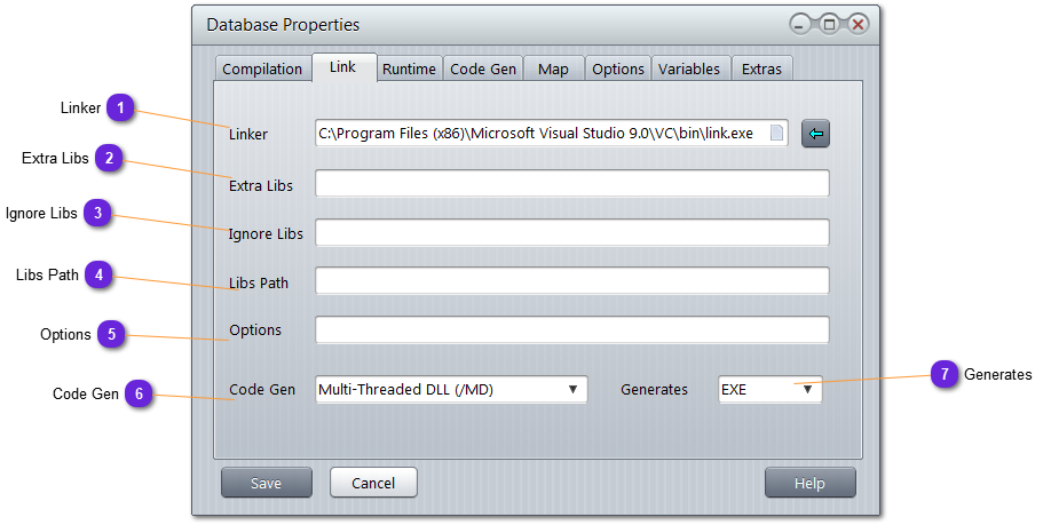

When selected, whole program optimization is forced, providing maybe a better performance although minimal in most cases.

## 7 Debug Mode

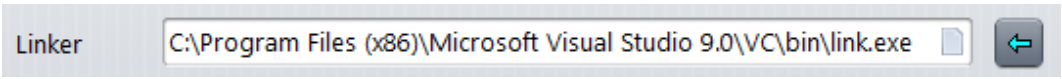



Target application will be linked in Debug mode, allowing debugging with Visual Studio is case of crash.

# Link




## 1 Linker




Replaces the Linker specified in Options. The default value is taken from the Option setting but can be changed here if needed.

## 2 Extra Libs


If 3rd party libraries are needed, must be specified here, separated by commas.

## 3 Ignore Libs


If some default libraries must be ignored because of conflicts with *Extra Libs*, all libraries to ignore by the linker must be specified here.

## 4 Libs Path


Paths used by the linker to find the extra library modules listed above. If several, must be separated by commas.

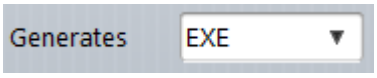
## 5 Options


Linker options added by the user.

## 6 Code Gen


Selects one of them according to the library used and the output type expected.

## 7 Generates

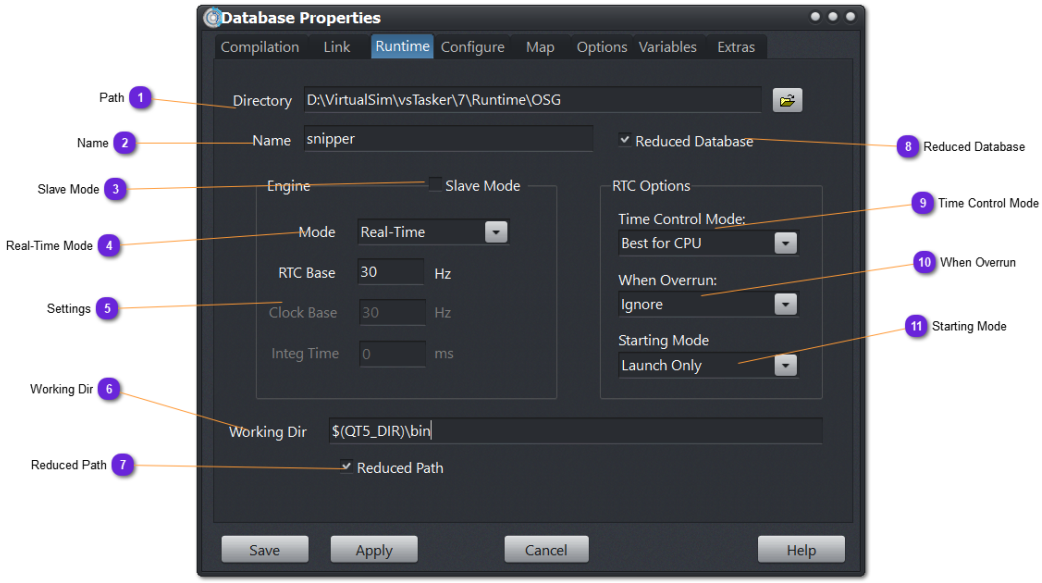



Selects if the output object must be an executable (EXE), a Dynamic Library (DLL) or a Component (COM)
See the Tutorial (Understanding Runtime) to learn how to use them.

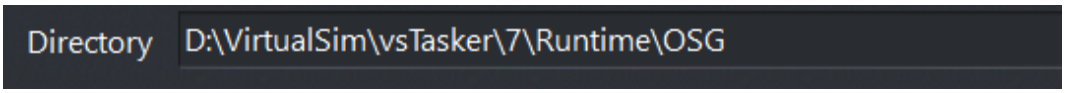
# Runtime

vsTASKER generates C++ code and produces an executable (or a dll depending on the settings).
If the GUI needs a database (.db) located normally in data/db directory, the runtime engine needs the corresponding compacted database containing everything needed by the engine for the simulation.

This database must be associated with the executable. It carries the same name as the executable with .rt extension. The runtime database is recreated every time the database is changed and saved from the GUI. You will find it in the same directory as the executable.

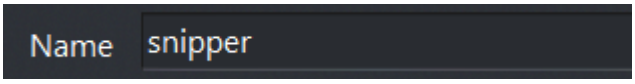



## 1 Path




Directory where the output (EXE/DLL/COM) is produced, the makefile is generated and the Runtime Database (.rt) is located.

## 2 Name

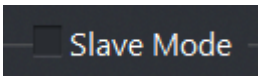



Name of the executable (without .exe or .dll) and the corresponding runtime database.

The extension depends on the output type selected in Link (*Generates*).

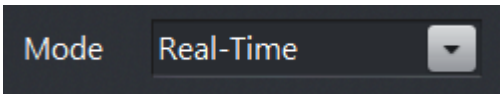
*To remember: For deployment of the simulation engine, in standalone mode, always include the runtime database (.rt). Standalone executable can be obtained from the Scenario property window.*

## 3 Slave Mode



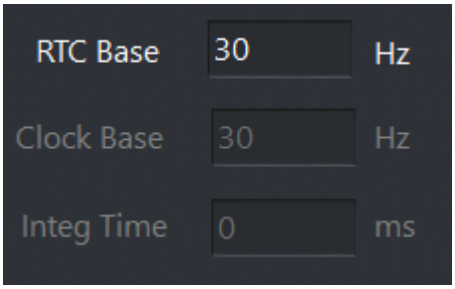

If checked, the RTC waits for an external TIC to advance of one cycle.

## 4 Real-Time Mode



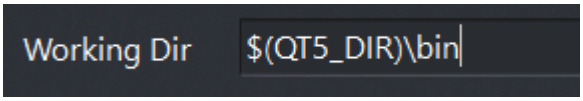

Specify if the RTC must run in **Real-Time** or **Asynchronous** mode. In **Real-Time** mode, The frequency is kept (if possible) through cycles. For example, if 30 Hz specified, then, every second (on the user clock), the RTC would have processed 30 cycles (one cycle every 33 milliseconds). In Asynchronous mode, the **Integration Time** is used to define the cycle length.

## 5 Settings



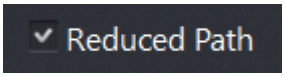

**RTC Base**: Value of the base frequency for the RTC in **Real-Time** mode.
**Clock Base**: Not used
**Integ Time**: In **Asynchronous** mode, value of the cycle in milliseconds the RTC must keep.

## 6 Working Dir



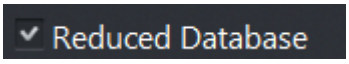

Add here the directories which will be added to the path when calling the Simulation Engine.
The paths added will not change the User or System path. It will only be set for the duration of the simulation run.

## 7 Reduced Path



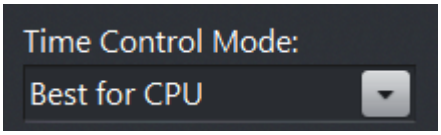

When checked, the runtime path will be cleaned and only contains the $(VSTASKER)\bin... directories plus the Working Dir ones (see above).
Use this option when the simulation engine crashes because of some others DLL found on the path instead of the correct ones.
When unchecked, the user/system path will be used as is.

## 8 Reduced Database



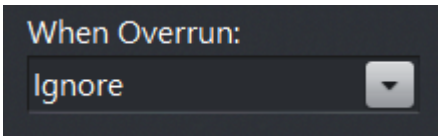

When selected, User (entered) code and help data are excluded from the Runtime Database (.rt), making it smaller in size and reducing the memory foot-print.
This is the default.

## 9 Time Control Mode



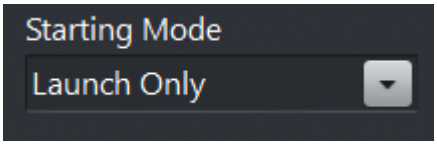

Choose between:
- ***Best for CPU*** release the CPU between cycle if time permits.
- ***Best for Accuracy*** keeps the load and wait for the integration time to be over before triggering the next cycle (or returning in case of an external call).

## 10 When Overrun




Choose between:
- ***Ignore*** will leave the internal time unchanged.
- ***Recup Time*** will try to keep the same number of cycles per second by reducing the wait time between cycles.

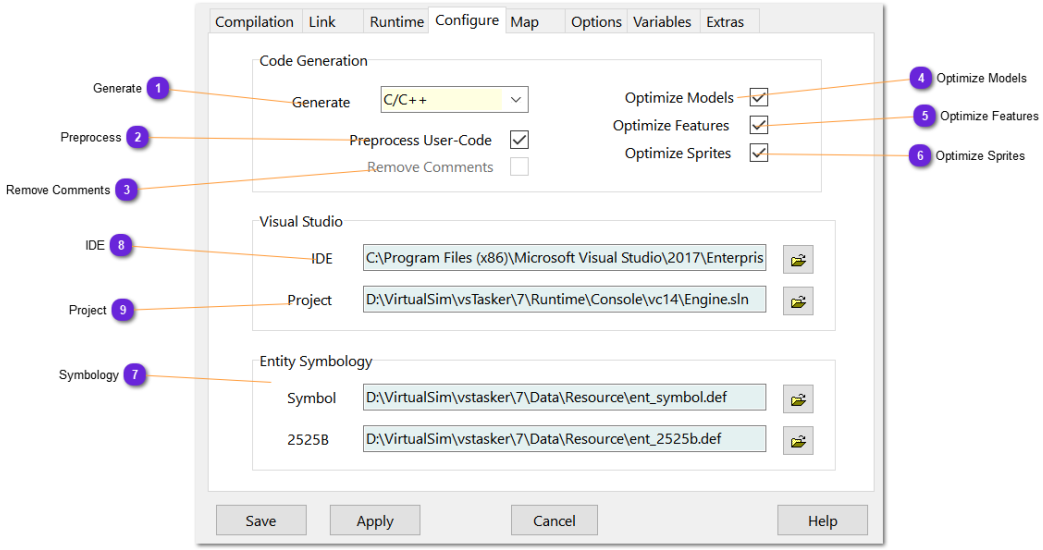
## 11 Starting Mode



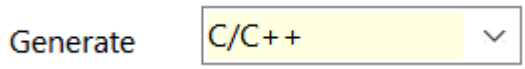

Choose between:
- **Launch Only** (default) does not start automatically (unless standalone mode). User must depress the Start button to start the simulation.
- **Start and Pause** will initiate a one cycle only start (without any time advance) to prepare the simulation. Pause mode is engaged. Press Start button to start the simulation.
- **Auto-Start** will start normally the simulation right after launch of the simulation engine. This mode should be selected for standalone executable, unless a START button is added into the HMI or if the start is provided by another mean (then *Start and Pause* shall be a better option).


*The **Start and Pause** mode works only once. When Stop is depressed, the restart will be operating normally (no pause).*

# Configure

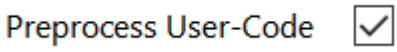



## 1 Generate




Specify what kind of code vsTASKER must generate for the simulation engine. For now, only C/C++ is supported.

## 2 Preprocess



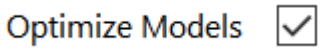

Call the preprocessor to replace the following macros: into

Change `E(myEntity):` into `((myEntity*)ent())->`
Change `P(myPlayer):` into `((myPlayer*)ply())->`
Change `S:` into `scen()->`
Change `P:` into `ply()->`
Change `E:` into `ent()->`
Change `L:` into `logk()->`
Change `K:` into `know()->`
Change `C:` into `ctx()->`
Change `G:` into `group()->`
Change `T:` into `scen()->terrain.db->`
Change `M:` into `scen()->meshes.db->`
Change `W:` into `scen()->winds.db->`
Change `V:` into `scen()->roads.db->`
Change `N:` into `vt_rtc->networks->`
Change `D:` into `vt_rtc->getDatabase()->`
Change `R:` into `vt_rtc->`
Change `A:` into `vt_amb->`

## 3 Remove Comments

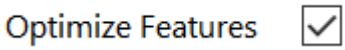

When checked, the generated code will not include the comments from the database.
The runtime database (.rt) does not include comments for security reasons, as the runtime database can be deployed.

## 4 Optimize Models



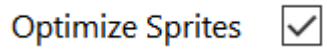

When checked, unused models (Components and Data-Models) are not code generated and available during runtime.
This is the default to reduce the compilation time and the executable memory footprint.

## 5 Optimize Features



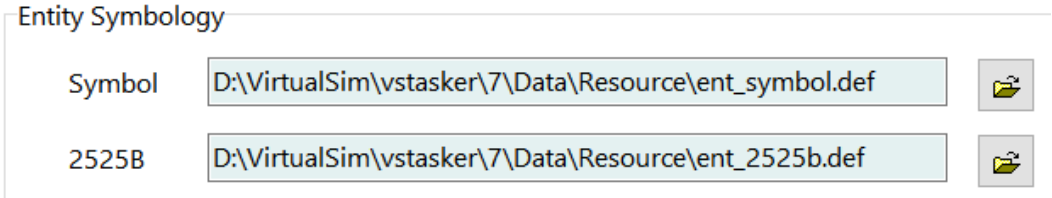

When checked, only used Feature *managers* are created.
This is the default to reduce the compilation time and the executable memory footprint.
Uncheck if **Features have to be created at runtime** (from code) because the *manager* will be required even if empty at simulation start.

## 6 Optimize Sprites



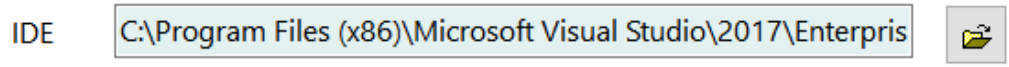

When checked, only used Sprite *managers* are created.
This is the default to reduce the compilation time and the executable memory footprint.

## 7 Symbology



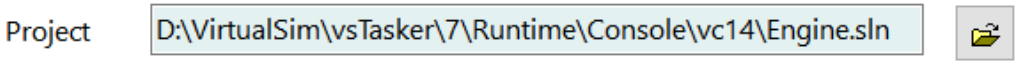

Redefine locally (for the current database) the general settings of Preferences for the Entity symbols.

## 8 IDE



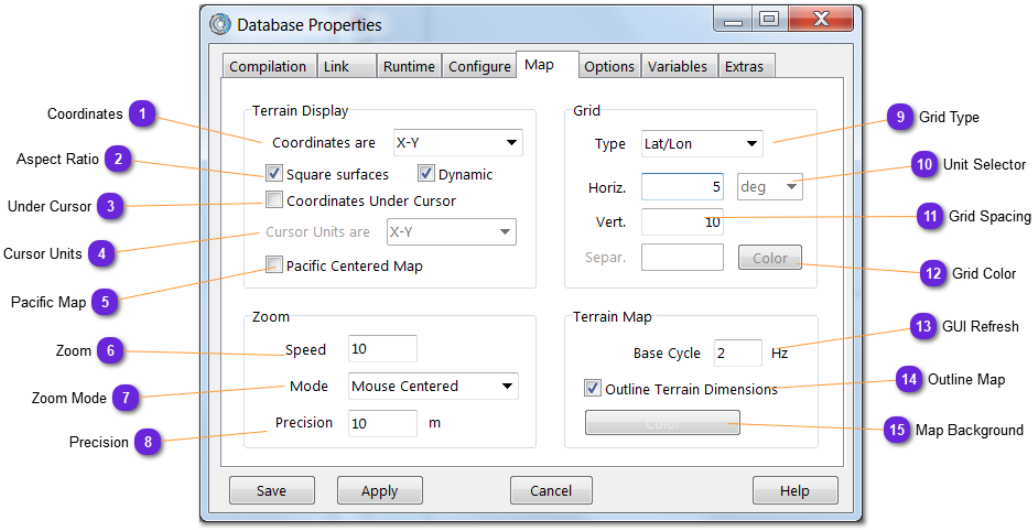

When using the **Visual Studio** menu option from any code editor of the GUI, the selected IDE will be used. If nothing is set, the option will not be available (grayed out).

# 9 Project

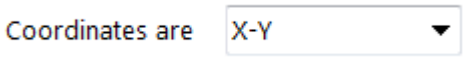



When calling Visual Studio to edit the selected code (from the GUI), a solution project can be used. It will enable the intellisense. Select the solution that is normally used for debugging, with the database generated code inserted in the project (otherwise, Intellisense will not be functional).

# Map




## 1 Coordinates

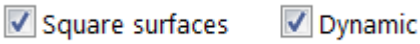



Select here the coordinates format to be used for display in the map terrain (mouse down, point click, selection area, grid...).

```
XY:           13279831.935686, 2888175.041949
Lat-Lon:      N026:11:21.3  E119:11:56.6
LL Decimal:   25.897061, 119.299798, 0
```


*When displaying WGS84 terrain data, set the coordinates to Lat-Lon so that to have real distance calculations.*
*The displayed coordinates will also be correct.*

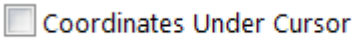
## 2 Aspect Ratio




vsTASKER is using flat earth projection meaning that on WGS84 maps, distortions are noticeable on high latitudes.
**Square surfaces** can be selected to avoid some vertical shrinking. The aspect ratio used is 1, meaning that drawing a circle will keep the distance on latitude and longitude over the map and its shape will not be oval. Select this mode when drawing 2D figures designed for flat earth projection (like Circle and Rectangle Special Zones)

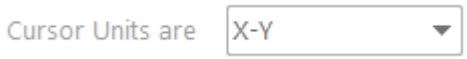
If **Dynamic** is selected, the aspect ratio will be computed according to the latitude of the center of the view. This mode can produce higher distortions on borders when viewing large areas but provides better accuracy on high zooms, around the center of the viewed zone.

## 3 Under Cursor



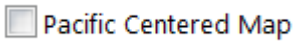

If checked, the mouse will carry the coordinates at the top of the mouse arrow.

## 4 Cursor Units

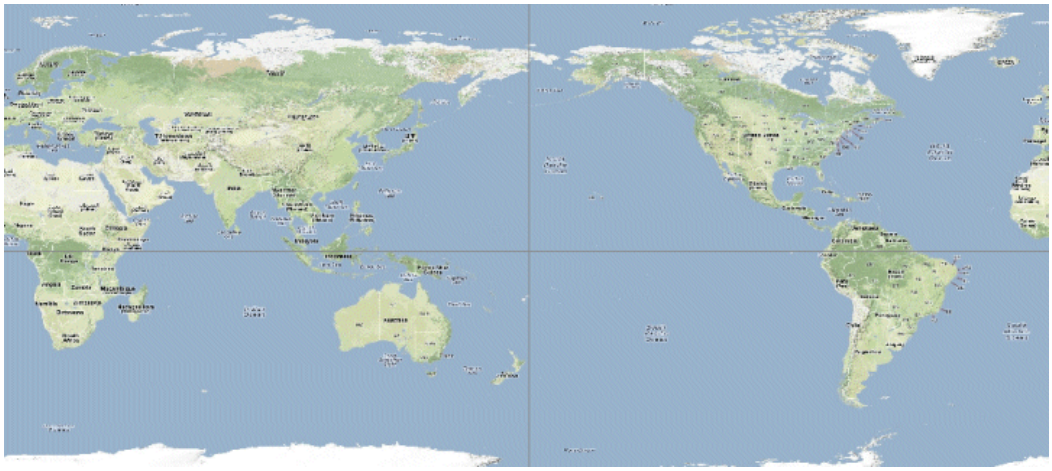



When the Under Cursor mode is selected, the coordinates unit is selected here. It can be different from the Map coordinates (above) for comparisons reasons or to display more usable value (Lat-Lon decimal).

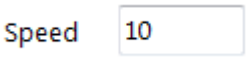
## 5 Pacific Map




Click this option when you are using a large map centered into the Pacific and having entities on both sides of the day change line.
It is suggested to use the Data/Terrains/Pacific map under such mode.
See here.

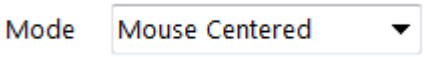


## 6 Zoom




Set here the zoom factor used when using the mouse wheel. The higher the value, the higher the zoom effect.

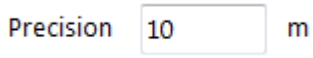
## 7 Zoom Mode




Select how the zoom must be performed when using the mouse zoom:

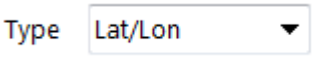
**Mouse Centered**: the window zooms around the location pointed by the mouse
**Window Centered**: the window zooms around the center of the displayed terrain map

## 8 Precision




This value specifies the accuracy of the zooming in Mouse Centered mode. The higher the value, the lesser the precision but the smoother the zooming.

## 9 Grid Type




Select the type of grid to display on top of the terrain map display:

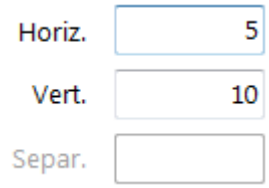
**None**: No grid displayed
**Cross**: Single cross with a square defining the boundaries of the terrain map
**Grid**: Vertical and Horizontal lines over all terrain but inside the square defining the boundaries of the terrain map
**Lat/Lon**: Latitude and Longitude lines are drawn according to the setting, in degrees
**Circular**: From the center of the map, circular rings separated by the radius value are drawn. According to the type of coordinates selected, flat earth or WGS 84 projections are used.


*The plugin DLL displaying the grid is located in /plugins/grid (GridDraw.cpp) and can be changed (or replaced) by the user.*
*In case of change, keep in mind that the successive product releases will overwrite the module. It is advisable to duplicate the module (myGridDraw.cpp), recompile it and use the generated myGrid.dll into the plugins definition window.*

## 10 Unit Selector

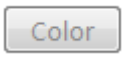

Unit selection for the grid.
**m** (meters), **km** (kilometers), **nmi** (nautical miles).
deg is only for Lat/Lon grid type.

## 11 Grid Spacing



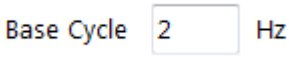

Setting values for the grid display, according to the type selection.
For **Grid**, horizontal and vertical spacing between lines. Unit is according to the selection (10) or in degrees for Lat/Lon grid.
**Separ** field is only used for Circular grid. It is the value separating two circles.

## 12 Grid Color




Set here the color used by the Grid module to draw the grid lines.

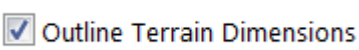
## 13 GUI Refresh




Set here the refresh rate of the Map/Terrain display at runtime.
The frequency shall be in accordance with the Export Component Model frequency. If the Export is running at 5hz, the Map Base Cycle shall be set up to 5hz. A higher frequency will increase the CPU load without any visible effect on the Map.


*A high frequency might impact the GUI response time.*

## 14 Outline Map




When selected, the terrain dimensions will be drawn (outlined). Deselect this option to hide it.
The terrain dimensions (width and length) are used to delimit the gaming area, inside a maybe much bigger map or terrain. It is useful to see the limits as much as to hide them.

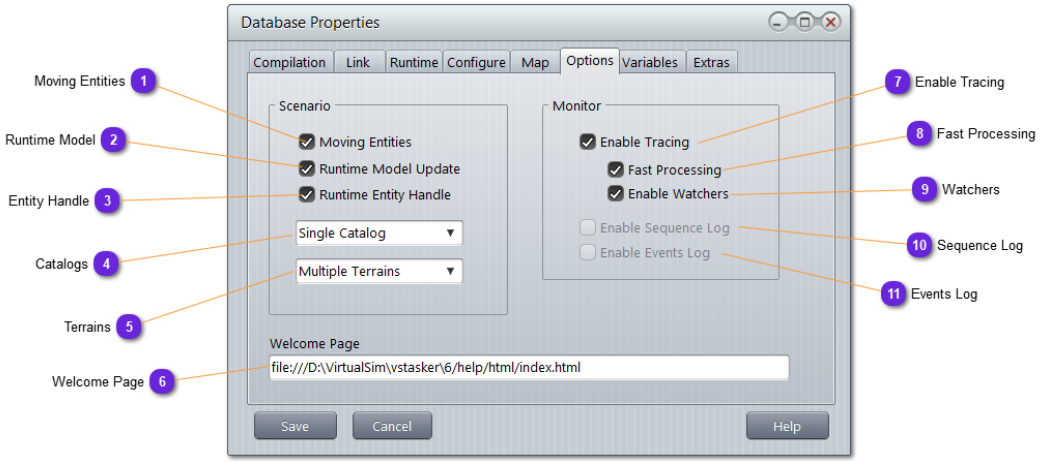
# 15 Map Background

Color

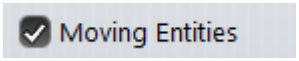
Set here the color requested for the Map terrain (OpenGL) background. The background is the visible area not covered with any layer or visible beneath vector layers.

*When two or more vsTASKER GUI must be displayed on the same computer (distributed simulation test), it is a good idea to change the color of the Map background to quickly differentiate each of them.*

# Options



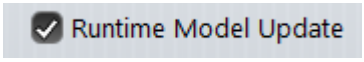

## 1 Moving Entities




When this option is checked, vsTASKER (GUI) will automatically update the position of the entities during runtime, by reading the shared-memory segment.
Export component must be attached to each entity.
Uncheck this option if the shared-memory is not used or if the GUI is not used either and to improve the simulation speed in case of numerous entities and high refresh rate.

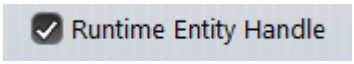

*By default, the Export component refresh rate is 1hz, which is quite low even with thousand entities.*

## 2 Runtime Model



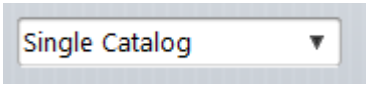

When checked, the Models (Components and Data-Models) are updating the public data (exported using //&&) whenever the Inspector window is opened during runtime.
If unchecked, the data is not updated from SIM to GUI and only the value changes from GUI to SIM are received.

## 3 Entity Handle




*Not used for now*

## 4 Catalogs

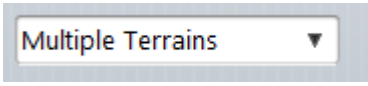



Select here which kind of catalog reference to use:

- Single: unique catalog for entities, allowing several scenarios to share the same list and definitions. This is the default.
- Multiple: each scenario hold their own catalog of entities. This used to be the default for versions older than 5.1.

*It is not advisable to change from Multiple to Single. It is better and simpler to have one unique catalog per database as catalog is preferably used for runtime entities. The only reason why each scenario shall have their own catalog is to restrict the runtime entity creation to a predefined set of entities that are unique for the current scenario.*

## 5 Terrains

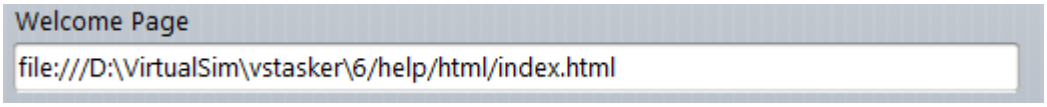



Select here if the terrain definition (including all layers and editor3D) is shared between all scenarios (in this case, it belongs to the database only and each scenario has a link to the database terrain data) or if any scenario must have its own terrain definition (in this case, the database has only a link to the current selected scenario terrain data).

- **Single**: unique terrain hold by the database. All scenarios will use the same terrain data definition. Memory efficient as the terrain will not be duplicated. The default mode.
- **Multiple**: each scenario hold their own terrain data definition. Must be used only if each scenario has different terrain layers.

*When only one terrain is used for several scenarios, use Single Terrain mode as it does save a lot of memory, mostly when the terrain is memory consuming.*

## 6 Welcome Page

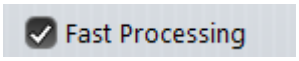



Put here the URL (local, starting with file:// or remote using http://) of any HTML pages or website you want, associated with the current database (or not).
By default, points to a blank page.

## 7 Enable Tracing

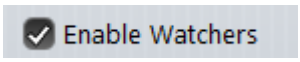

When checked, vsTASKER GUI will display in magenta the active objects of the selected (focused) entity. This is the default but might be a little time consuming for release simulation engines that request maximum performances. When unchecked, the shared-memory is not used.

## 8 Fast Processing



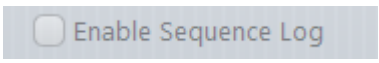

When checked, the incoming (shared-memory) tracing pile is processed continuously (by the GUI) until emptied. This insures that the actual tracing status is in synch with the Simulation Engine but the risk is a lower GUI reactivity in case of faster than real-time mode.

## 9 Watchers



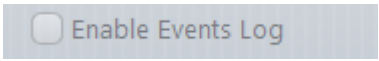

When checked, all the Watcher objects (in Logics) are working, meaning that the Simulation Engine will update the shared-memory for tracing in the GUI when the Logic (containing the Watchers) is opened during runtime. This is the default and might be a little CPU consuming.

## 10 Sequence Log



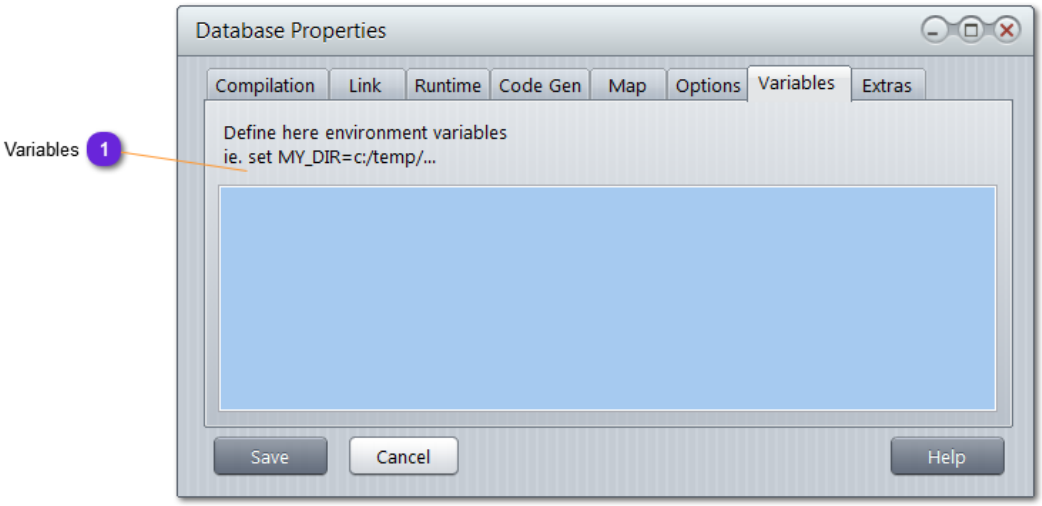

*Not available yet*

# 11 Events Log

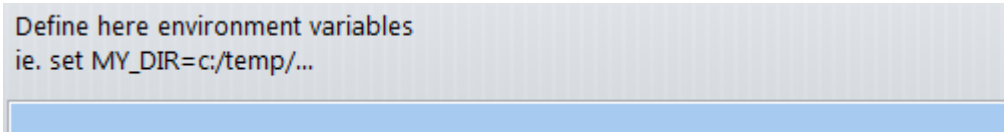



*Not available yet*

# Variables



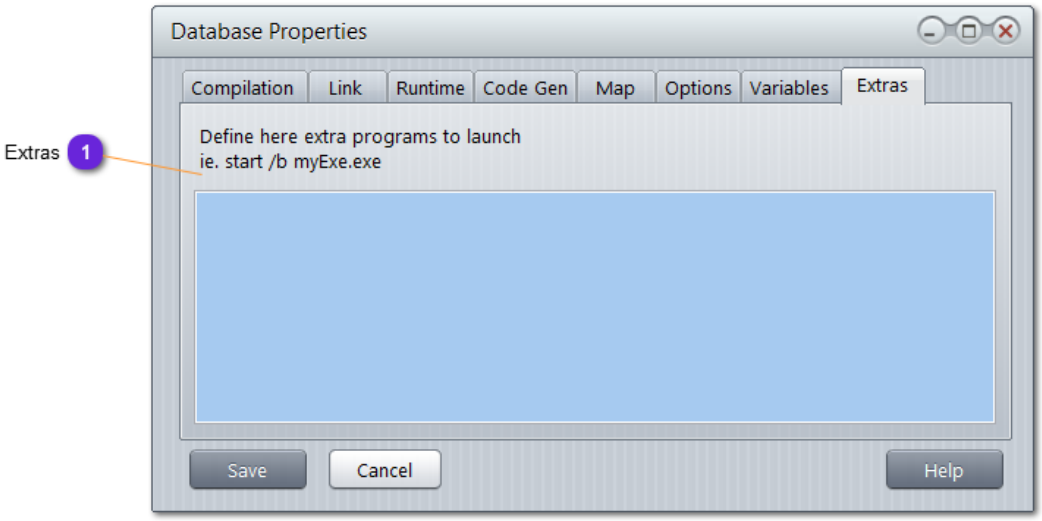

## 1 Variables



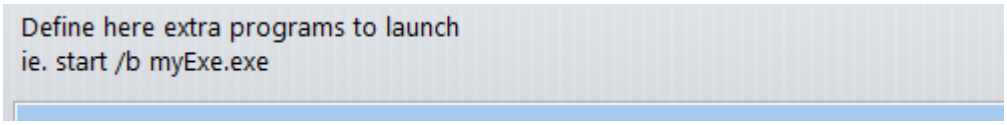

Add here all the environment variables that will be set **just before the nmake** (simulation engine compilation and link).
The lifespan of these variables will only be the one of the compilation.

*Works only for EXE and DLL targets.*

# Extras




## 1 Extras




Add here all the commands to be launched **after the simulation engine**. For example, if another viewer of application must be started after the simulation engine (a control panel, a driver, a logger...) then, it is a good idea to add them there.


*The started programs must be manually (or by code) closed. vsTASKER does not handle their ending, mostly when `start /b` is used.*

# Plugins

This window is used to change, add or remove plugins for the Scenario Map display.
Plugins are located in C:/virtualsim/plugin directory.

**[def_*]** are default plugins provided with vsTASKER.
**[usr_*]** are either samples or user defined plugins (DLL).

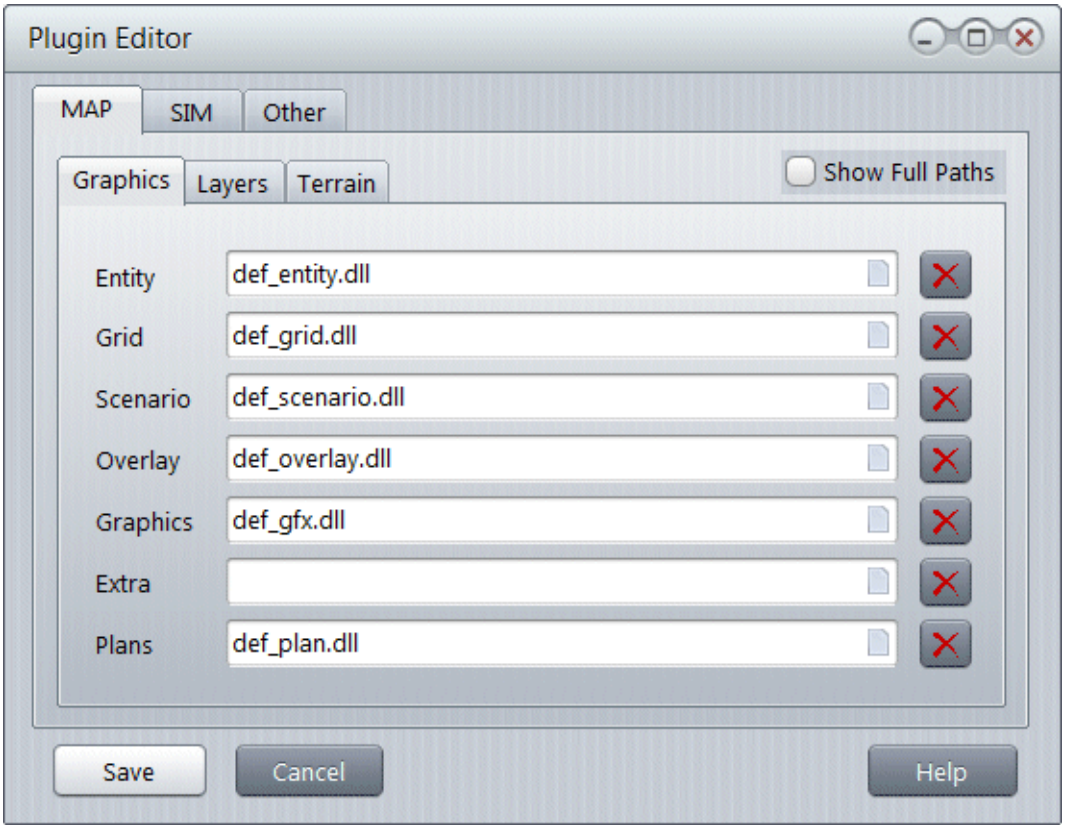
All these plugins are written in C++ using OpenGL code and compiled with Visual C++.
Sample projects are available in the plugin directory.

*Refer to the Developer Guide to learn more on the built-in plugins and how to create yours.*

## • MAP & SIM

Use the **MAP** for vsTASKER display on the Terrain panel; plugins are DLL and located in /plugins
Use the **SIM** for any OpenGL runtime display (standalone or using HMI or Qt map); plugins are LIB and located in /lib/vc90 or /lib/vc100; vsTASKER will automatically find the proper library according to the compiler setting (vc90 for Studio 2008, vc100 for Studio 2010).

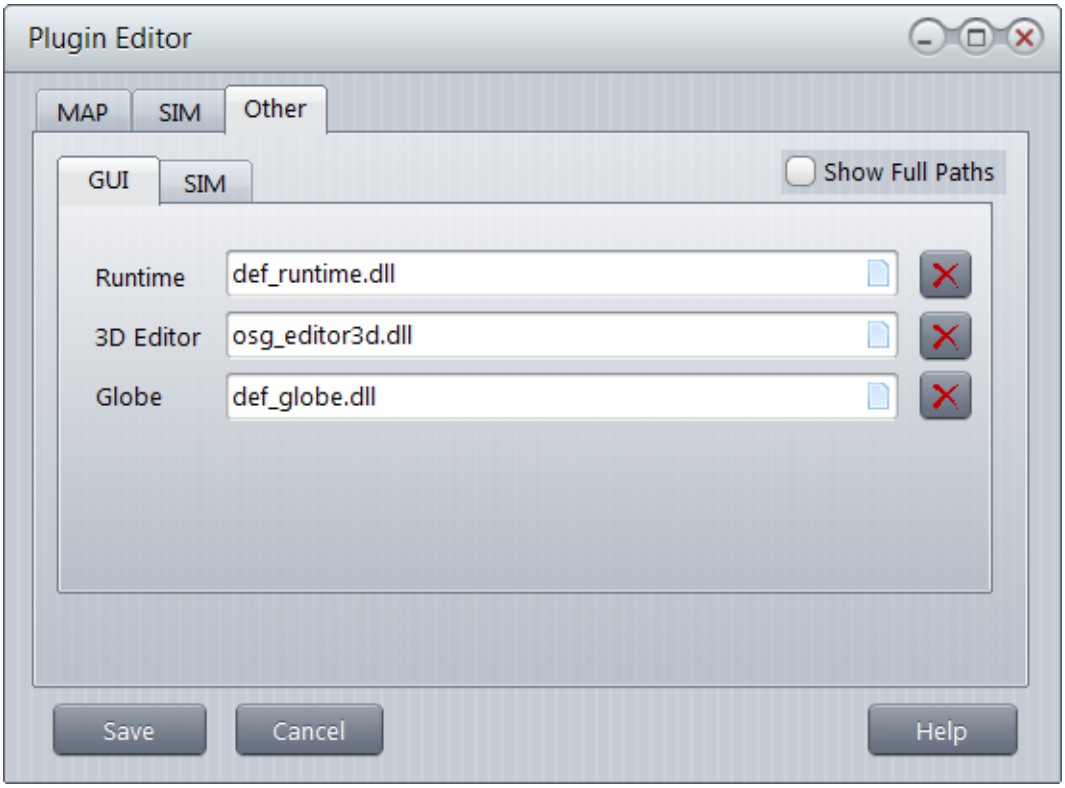
*If DLL or LIB are not loaded, the Graphics or Layers or Terrain will not be visible on the part where they are missing (vsTASKER GUI or Simulation engine view).*




- **Other**

Here are some special GUI only plugins.

### Plugins

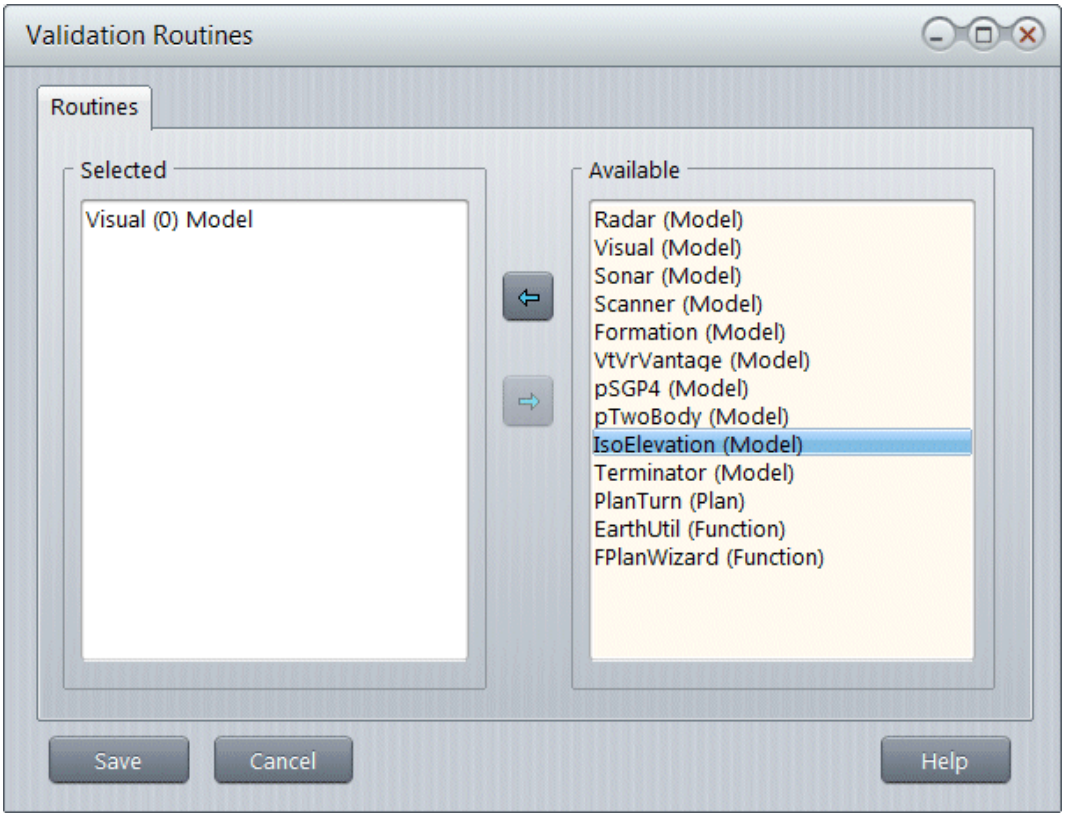



- **Runtime**: Select the plugin that maintains the runtime list in the Environment tree-list. Default is def_runtime.dll
- **3D Editor**: Select the plugin that displays the 3D editor on the Terrain map (of vsTASKER GUI). Default plugin is osg_editor3d.dll. A globe_editor3d.dll have also been developed, based on osgEarth, to provide a globe view of the earth with orbits and satellites objects.
- **Globe**: Select the plugin that will display the earth globe, mainly for Satellite views and orbits.

# Validation Routines

Validation routines are used to display on the Map relevant graphics about the Component or DataModel attached to a Model Behavior.
These routines are also used to validate the Component or DataModel Dyn-UI values.

*Refer to the Developer Guide to learn more on the built-in plugins and how to create yours.*




- **Selected**: You will find here all running routines (dll) with, between parenthesis, the number of Models attached to it. Double-click one to move it to *Available* list.
- **Available**: lists all pendant routines as specified in the /validation/validation.lst file. Routines in *Available* list are not running. They must be moved to *Selected* to run.

# Templates

Templates are reusable copy of database settings or parts.
It facilitates the design by providing predefined data instead of blank profiles.

When a database is saved as a template, it creates a directory in the data/template folder, with the name of the template.
In this folder, parts of the database might be stored as subpart templates. These subparts will also be considered as templates.

For example, if a database MySpecialDevice is saved as template, only Classes, Models, Curves, Logics, Knowledge, Viewers (...) are stored.
Scenarios and Entities templates must be saved under database template. This will facilitate scenario creation and population.

A Template is available for some vsTASKER items only:

- Database: This is the basic templates that will hold the embedded one. The database template will save the Sources, Settings and Plugins only.
- Scenario: Can only be saved if the database is a Template. The scenario template will save the internal Code and Terrain, but not the Entity, the Meshes and the Features. Scenario templates belong to the Database template.
- Entity: Can only be saved if the database is a template. The entity template will save the internal Code and Models, but not the Logics and the Knowledge. Entity templates belong to the Database template.


*To create a valid Database Template, first save the Database under a name, then save at least one Scenario inside this template.*

# Using Templates

Whenever a new Database is created, vsTASKER will propose to use one of the existing templates.

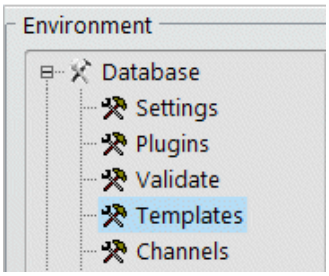
If one template is selected, whenever a new scenario is created, vsTASKER will propose to use one of the existing templates from the opened database template.

Same thing whenever a new entity is added into a scenario.

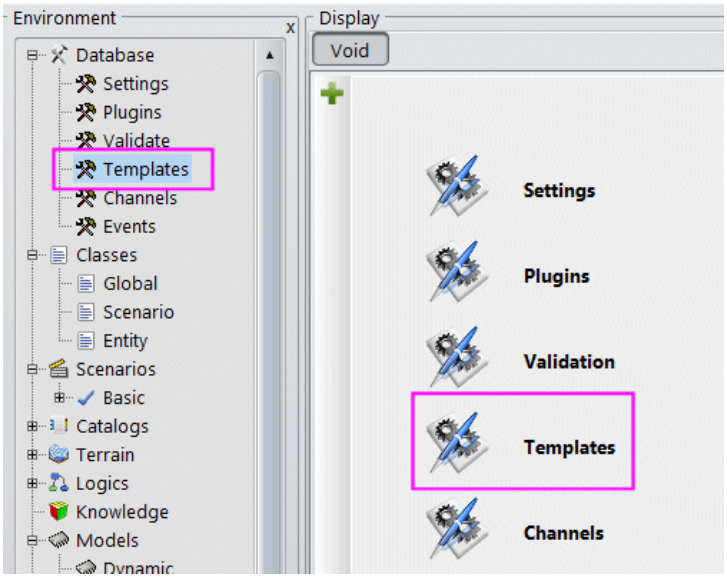
Scenario and entity templates must be stored inside a database template.
To store a scenario or an entity as a template, first, create the database template.

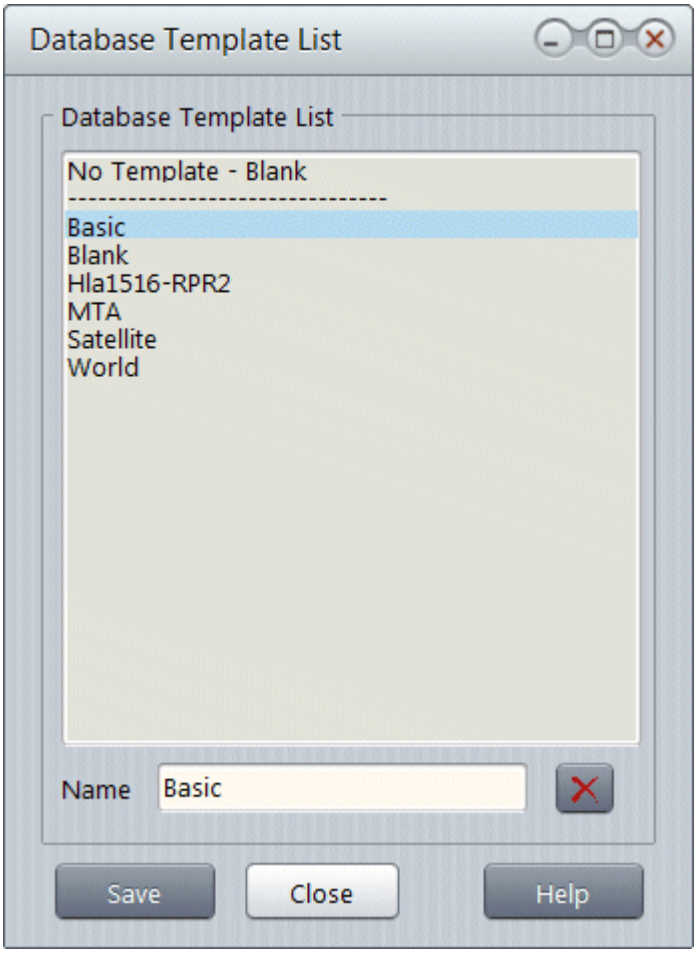
# Database Template

Once a database is setup, creating (or updating) a template can be done from the Database Item...




... then selecting Templates:



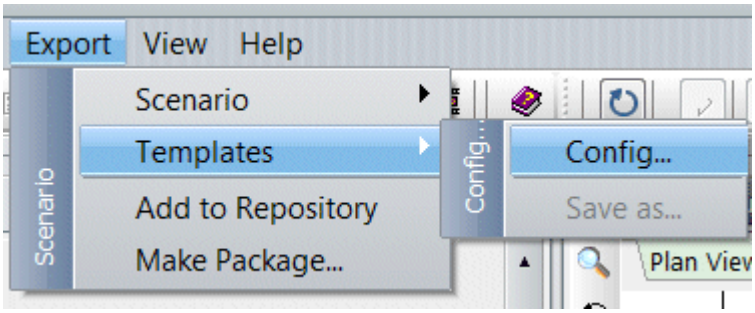



From this window, either select an existing Template to update it with the current Database settings, or enter a new Database name and press OK.

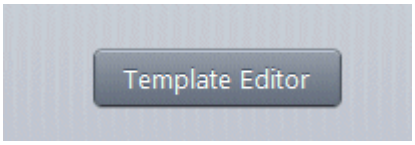
# Scenario Template

From inside a Database template, it is possible to create or update Scenario templates.
A scenario template will save everything stored into the Scenario Properties window. This includes the user-code, Terrain and Entities.

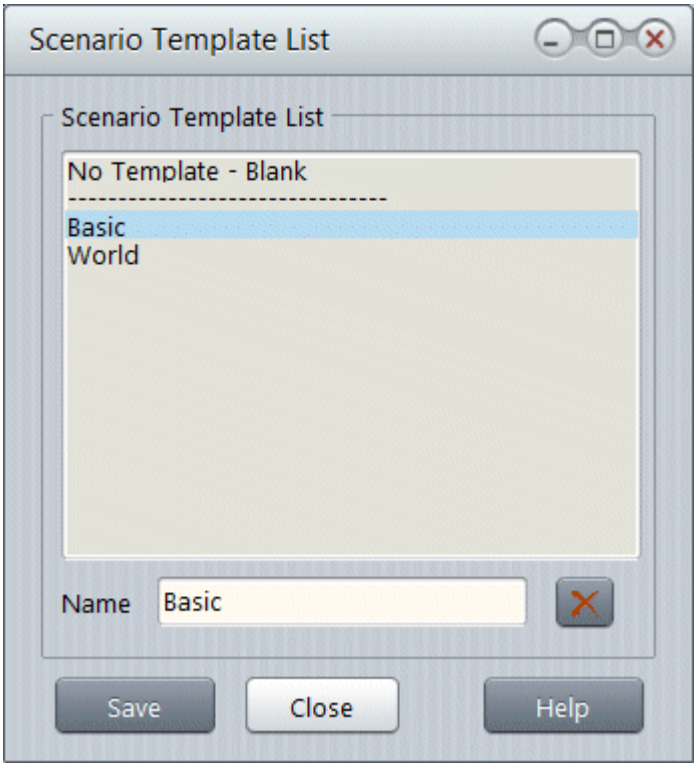
To create or update a scenario template, either select the scenario and use:




or use the:




of the Scenario Properties window.

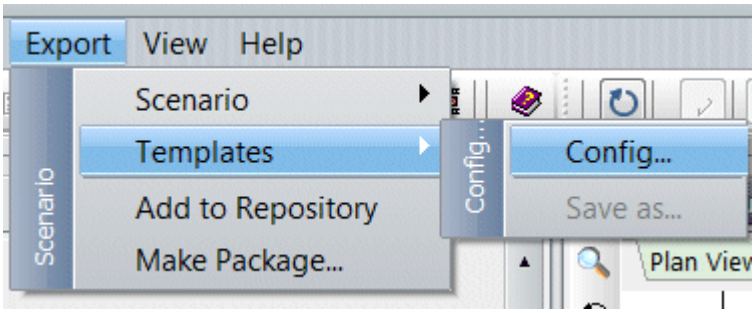



From this window, either select an existing Template to update it with the selected scenario settings, or enter a new scenario template name and press OK.

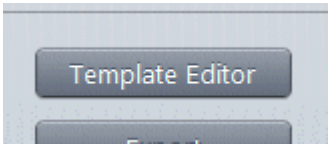
# Entity Template

From inside a Database template, it is possible to create or update Entity templates. An entity template will save everything stored into the Entity Properties window. This includes all Behaviors.

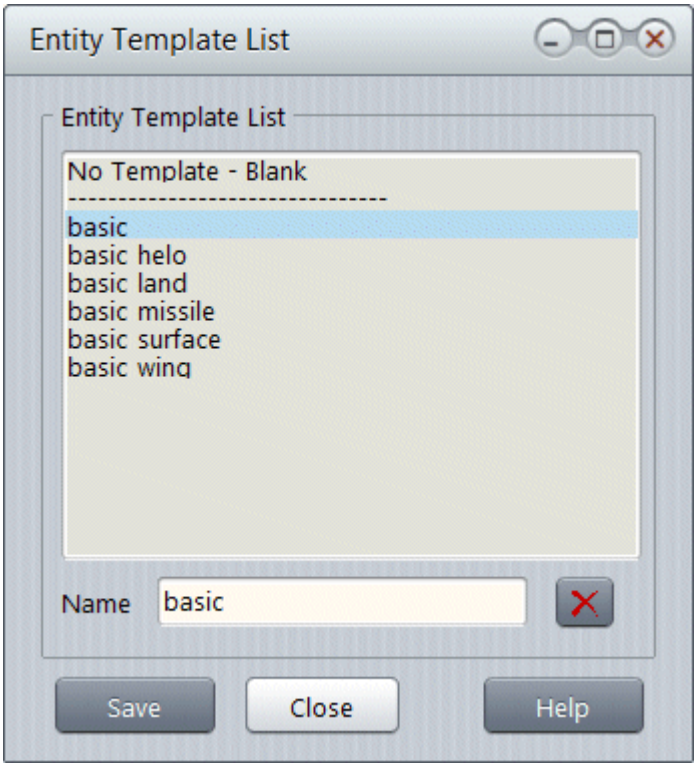
To create or update an entity template, either select the entity and use:




or use the:




of the Entity Properties window.

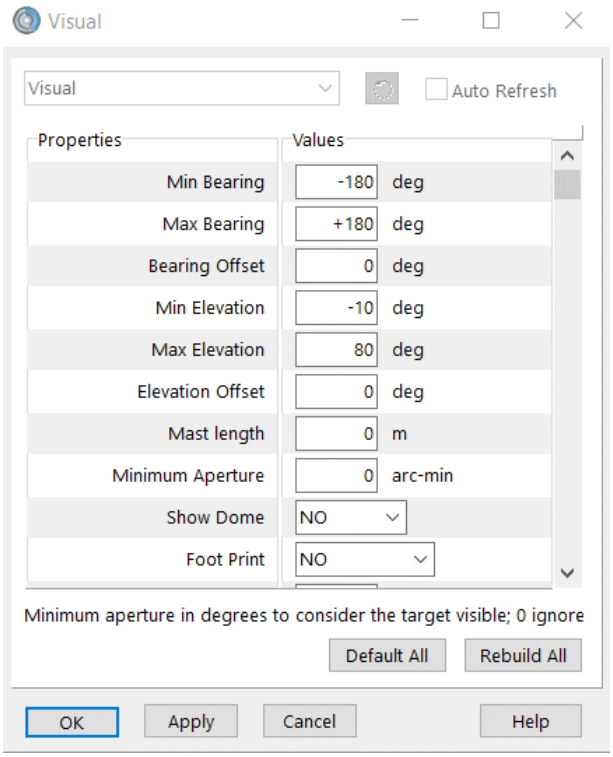



From this window, either select an existing Template to update it with the selected entity settings, or enter a new entity template name and press OK.

# Dynamic Interfaces

One of the key features of vsTASKER is its ability to automatically generate basic GUI interfaces for accessing parameter values of any object parameter.
With Dynamic Interfaces (let's call it Dyn-UI), user can define a variable belonging to a class object and make it modifiable for all instances, at Design and Runtime, using the generated GUI.

A Dyn-UI gathers all parameters specifically tagged in the object/class definition as: user modifiable.




In this example, the Sonar Component has defined all these variables as dynamic parameters.

In the Declaration part of a Logic, Knowledge, DataModel, Component (...), marking a variable for the Dyn-UI is done using the **//&&** sign after the declaration:

```
int my_private;
int my_public;  //&&  DEF[50]
```

In this example, my_public integer variable only will be added into the object Dyn-UI with the default value of 50.

The obvious advantage of Dyn-UI is that values it controls becomes available to the end user. He can modify any value of any interface and replace the default value. This can be done during design time or runtime (when simulation is stopped, design values will be retrieved).

# Using Dyn-UI

User can define C++ variables for automatic interface code generation.
With a simple formalism, vsTASKER provides for these variable GUI interface, default value mechanism and runtime user updates.

## • How to Use

In the Declaration panel, after the variable definition, the sign //&& or //[!# will enable the Dyn-UI for the declared variable.

ie:

```
int my_data;  //&&
```

If this variable is defined in a DataModel or Component, Logic or Knowledge, then, it will be accessible in the **Interface** part of the Behavior referring this Model.
User can put whatever value he wants at design time and this value will be put into the variable at runtime, during the initialization phase of the object instance.

If the Behavior Dyn-UI is used during runtime, the new value will be automatically sent to the object instance for update. All these mechanisms are automatically activated for any Dyn-UI variable defined by the user.

According to the above example, the following code will be automatically generated (and then, can be used from any vsTASKER code):

```
// ************************************************
inline void Cpt::setMyData(int data) { my_data = data; }

// ************************************************
void Cpt::setUserData(UsrInterface* setup)
{
   if (!setup) return;
   // Setup local
   CInterface* intfc = setup->get("Cpt");
   if (!intfc) {
      printf("Error Cpt::setUserData() with %s!\n", setup-
>getModuleName());
      return;
   }
   char* data;
   data = intfc->getValue("my_data"); if (GOTSTR(data))
setMyData(atoi(data));
```

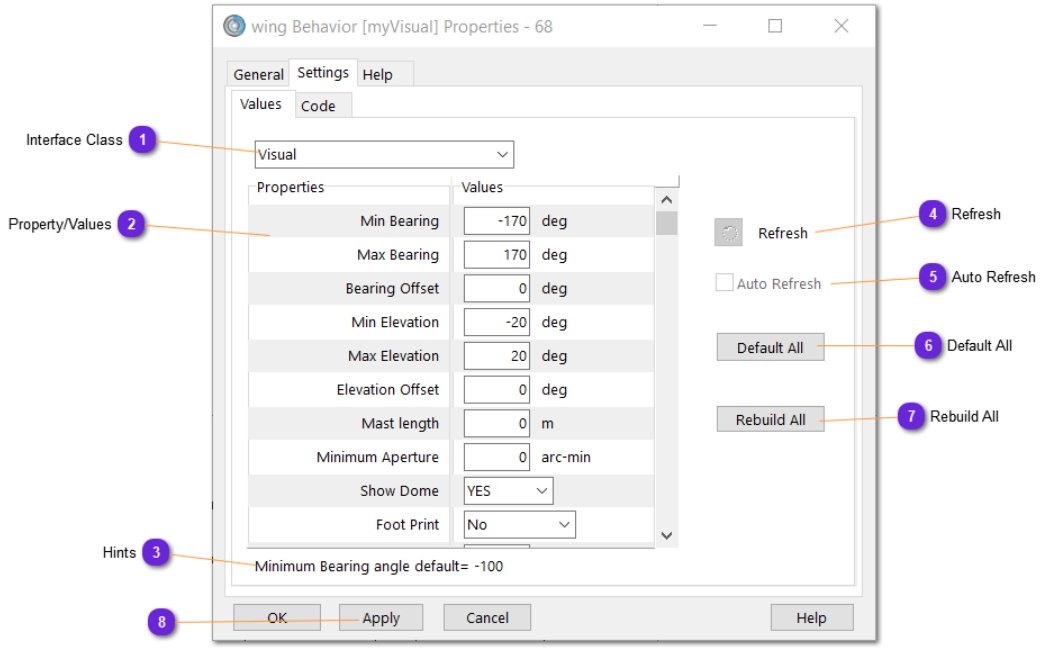
```
    // Put below user code to wrap-up settings
}
```

## • Where to Use

Following objects allow definitions for Dyn-UI variables:

- DataModel
- Component
- Logic
- Knowledge
- Federate Item
- Routines
- 4586 Messages
- CIGI Packets
- DDS Topics

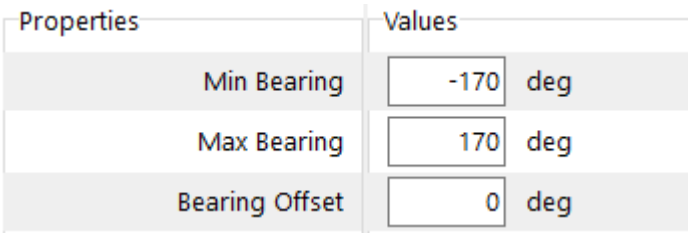
# Setting Values




## 1 Interface Class


For Components and Data Models only, list all the interfaces defined in the inherited classes, from the referenced object one down to the root class. All intermediate classes without interfaces are skipped.
Changing the Interface will also change the Property/Value list.

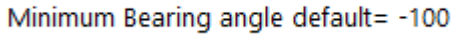
## 2 Property/Values




List of all parameters of the selected interface class.
Values are defaulted from the class definition itself (see here)
If some values are modified locally, they will be recorded at the local interface level.
Units are for display only. Can be removed but should not be altered.

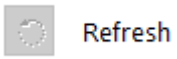

*At start, initial values are buffered and at simulation end, initial values are retrieved.*
*All modified values of all interfaces during runtime are lost.*

## 3 Hints




Display the description (if any) associated with the selected parameter (mouse in or mouse over).

## 4 Refresh




At runtime, click this button for request the simulation engine to send back the current values of this interface.

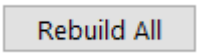
## 5 Auto Refresh


If checked, the update will be done automatically at 1hz frequency.

## 6 Default All

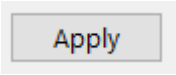

When pressed, all the user modified values are replaced by the interface default values.
Cannot undo.

## 7 Rebuild All




If the interface has been changed in the class (new parameters added or some removed), the Property/Value list must be updated (it is normally done automatically but in some case, manual rebuild must be done).
This button can also be used when the interface seems corrupted.

## 8 Send




At runtime (only), the Apply button changes to Send.
When depressed, all modified values of the current interface will be sent to the simulation engine for update.

# Reserved Keywords

These options follows the Dyn-UI declaration sign //&&
They can all be combined as long as separated by spaces and on the same line as the variable.

*Tags are case sensitive. Must all be capitalized.*

## • Review

**DEF[...]**: specifies the default value of the variable. The value in brackets must match the variable type (except for FileDef type which requires an integer)
*ie:*

```
int my_value; //&& DEF[-1]
```

This keyword can also be used to initialize variable of embedded interfaces. See INTF below.

---

**LBL[...]**: specifies the name to be used in the generated interface instead of the raw variable labeling.
*ie:*

```
int my_value; //&& LBL[My Value]
```

---

**KEY[...]**: uniq identifier representing the variable. Normally, identical to the code name. The only reason why to change the key using this tag is to keep the name but discard all default values attached to it in the various databases. As the Dyn-UI look for variable default values according to the key, if it is not found, then DEF[] value will be used.
*ie:*

```
int my_value; //&& KEY[new_value]
```

---

**SET[...]**: go here to learn how to use setters.

---

**DISABLED**: the variable, label and unit will appear disabled on the Interface. Useful to show something which is not available.

---

**READONLY**: the value will be displayed only but won't be modifiable (action button will be disabled). Similar to DISABLED but without the grayed out effect.

---

**MMX[...]**: boundary values constrain user entered value to remain in the specified min-max bounds. Only for number type.
*ie:*

```
int my_value; //&& TYPE[number] MMX[-10,10]
```

---

**UNIT[...]**: for display only. Can be whatever the user decides. The unit will be displayed at the right of the value text field
*ie:*

```
int my_value; //&& TYPE[number] UNIT[rad]
```

---

**HINT[...]**: text to be displayed in the GUI to explain the purpose or content of this variable
*ie:*

```
int my_value; //&& HINT[Put here the target temperature]
```

---

**LIST[...]**: If variable is an enumeration, puts here all the strings that should be displayed on a drop-list instead of their associated number (which will be stored in the variable). If numbers are not specified in the list, the first one will be 0 and others +1
*ie:*

```
int my_value; //&& LIST[cold=1,warm=2,hot=3,danger=5] DEF[warm]
```

*or:*

```
MyEnum my_value; //&& LIST[Cold, Warn, Hot, Danger] DEF[Cold]
```

---

**TYPE[...]**: Special type for special predefined Dialog Boxes. These types call special actions in the generated interface. Between parenthesis, the expected variable type.
*ie:*

```
int color; //&& TYPE[color]
```

List of all types, case insensitive:

| | |
|---|---|
| **String, Text** | Specify that the data is alphanumeric or that LIST[] values are strings and not integ |
| **Number** | When the data is numerical (short, int, long, float, double) |
| **Boolean, Bool** | When the variable is 1/0, true/false, good/bad, on/off... (bool) |
| **Color** | Display a color dialog box : (int) |
| **Chart** | List all defined Chart (string) |
| **Curves, Curve** | List all defined Curves (string) |
| **Formation** | List all defined Formation patterns (string) |

| | |
|---|---|
| **Point** | List all defined Points (feature) in the related scenario (string) |
| **Path** | List all defined Paths (feature) in the related scenario (string) |
| **Zones, Zone** | List all defined Special Zones (feature) in the related scenario (string) |
| **Road** | List all defined Roads in the related scenario (string) |
| **Mesh** | List all defined Meshes in the related scenario (string) |
| **Trajectory** | List all defined Trajectories (feature) in the related scenario (string) |
| **Flight-Plan** | List all defined Flight Plans (string) |
| **Orbit** | List all Orbits (string) |
| **Search-Pattern** | List all Seach Patterns (feature) in the related scenario (string) |
| **Entity** | List all defined entities (except current one) in the related scenario (string) |
| **Ent-X** | Synchronize with entity X value |
| **Ent-Y** | Synchronize with entity Y value |
| **Ent-Z** | Synchronize with entity Z value |
| **Ent-Speed** | Synchronize with entity speed value |
| **Ent-Heading** | Synchronize with entity heading value |
| **Ent-Logics** | List all entity behavior logics (string) |
| **Ent-Knowledge** | List all entity behavior knowledge (string) |
| **Ent-Components** | List all entity components (string) |
| **Ent-DataModels** | List all entity data-models (string) |
| **File** | Go here for a full description |
| **FileDef** | Go here for a full description |
| **Null** | Useful to set a defined pointer to NULL in setDefault. Will not appear in the genera |

---

**ENUM[...]**: go here to learn how to use this keyword

---

**INTF[...] or INTFC[...]**: go here to learn how to use this keyword

---

***GROUP[label,nb]***: to group (from the current one) the nb following fields (including this one) into the same background color. The label will be displayed on the left corner of the first group item.

---

# • Some Examples

```
     int   type;       //&& TYPE[FileDef] FILE[Entities.def] PATH[$(VSTASKER_DIR)/
Runtime/CIGI/Settings/Titan]
     int   categ;      //&& LBL[Category] DEF[Land]
LIST[Other=0,Land=1,Sea=2,Air=3]
     float min_dist;   //&& LBL[Update: Distance] DEF[0.5] UNIT[m]  HINT[Minimum
change for update] GROUP[minimums,3]
     float min_angle;  //&& LBL[Update: Angle] DEF[1] UNIT[deg]  HINT[Minimum
change for update]
     int   min_time;   //&& LBL[Update: Time] DEF[5] UNIT[s] HINT[Update every
seconds, 0 for always]
     int   on_focus;   //&& LBL[Update on focus] DEF[true] TYPE[boolean]
     int   fcs_dist;   //&& LBL[Close to focus] DEF[5] HINT[Distance to focused
entity to trigger update]
     int   state;      //&& LBL[Entity State] DEF[Active]
LIST[Standby=0,Active=1,Remove=2]
     char  parent[NS]; //&& LBL[Parent Entity] TYPE[Entity]
     int   collision;  //&& LBL[Collision Detection] DEF[Enable] LIST[Disable=0,
Enable=1]
     int   alpha_en;   //&& LBL[Inherit Alpha] DEF[false] TYPE[boolean] DISABLED
     int   anim_state; //&& DEF[Stop] LIST[Stop=0,Pause=1,Play=2,Continue=3]
DISABLED
     int   clamping;   //&& DEF[AltAttClamp]
LIST[NoClamp=0,AltClamp=1,AltAttClamp=2]
     int   smoothing;  //&& DEF[false] TYPE[boolean] DISABLED
```

# File Tag

When using **TYPE[File]**, the Dyn-UI will show a button to call the file selector, although the full file path could be entered manually.
With this type, the following tags are needed:

**PATH[...]** or **DIR[...]** : directory where the file must be found. If omitted, the file selector will start from the system root.

**FILE[...]** : default filename to be used (must exist in the directory mentioned above or provide here the full name including the path).

**EXT[...]** : extension filter, for the file selector only.

This type will need a character string long enough to store the full filename. Do not use `char*` nor `string` unless you define your own setter. By default, Dyn-UI uses `strcpy`.
`FS` is an alias for `FILENAME_SIZE` which worth `256`.

*ie:*

```
char ent_files[FS]; //&& TYPE[File] FILE[entities.def] DIR[$(VSTASKER_DIR)/
Runtime/CIGI/Settings] EXT[def]
```

# FileDef Tag

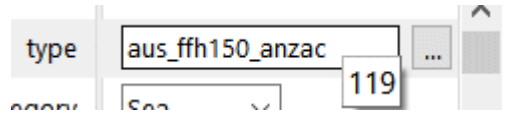
When using **TYPE[FileDef]**, the UI will show a button to call a special window accessing a formatted definition file made of triplets: **id, name, optional image**.

**PATH[...]** or **DIR[...]** : directory where the definition file must be found. If omitted, the file selector will start from the system root.

**FILE[...]** : default filename to be used (must exist in the directory mentioned above or provide here the full name including the path).

*ie:*

```
int type; //&& TYPE[FileDef] FILE[entities.def] DIR[$(VSTASKER_DIR)/Runtime/CIGI/Settings]
```




In the above image, the type (int) value stored is 119, but the name displayed is aus_ffh150_anzac (see file below). The Dyn-UI will automatically handle the triplet (id, name, image) from the definition file specified by the tags.

## • File Definition Format

The formatted file must be text based with the following structure:

```
# Entities
# Aircrafts
118=JTAC_FA18F_RAAF_NG-inventory:img/preview_jtac_fa18f_raaf_223.jpg
171=jtac_b1_lancer
234=MIG_29:img/preview_mig29.jpg
# UAV
199=MQ_1_Predator_UAV-JTAC
#--
#--
# Ships
119=aus_ffh150_anzac:img/aus_anzac.jpg
#--
```

Each line defined an entry, either a data or a folder definition (opening or closing a level).
A data entry is specified this way: id**=**name**:**image filename
**id**: a uniq value

**name**: what should be displayed in place of the id. Can contain any character except : and =
**image**: optional image which can go with the object. Can be a full path or relative to the file definition directory.

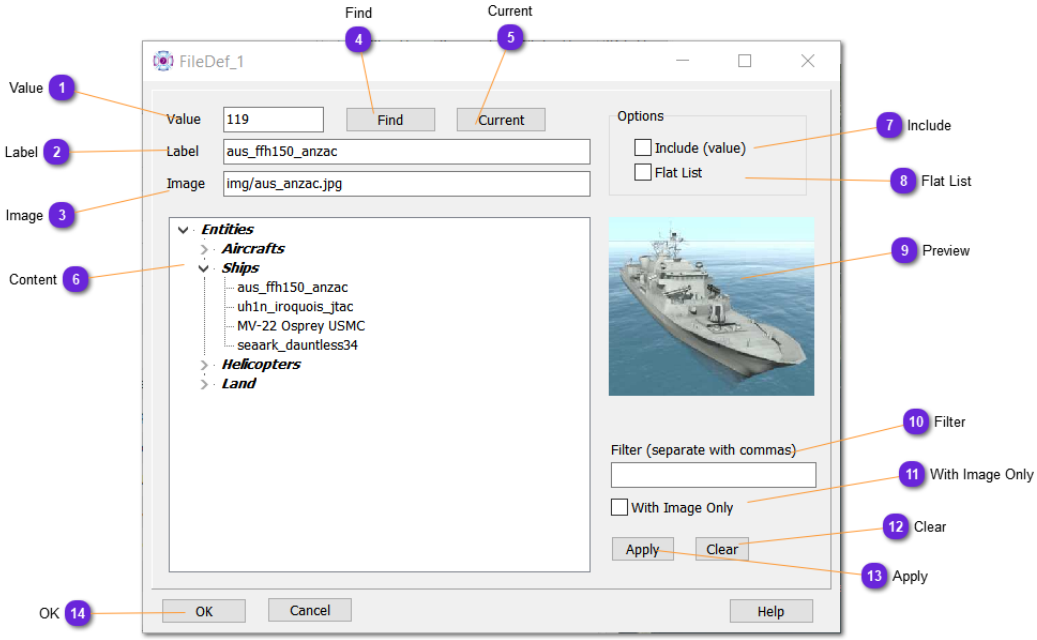
*id must be followed by a **=** (equal sign) and image filename, if any, must come after a **:** (colon)*

When a line starts with **# foo** it creates a new folder named foo, to store the following data entries.
When a line starts with **#--** it closes the latest opened folder.

## • File Definition Selector

See here.

# FileDef Selector




## 1 Value


The unique id which is normally handled by the software. The first element of the file data triplet.

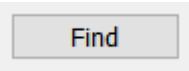
## 2 Label


Name associated with the id (value). Although it should be unique, it has only informative value, for the user.

## 3 Image

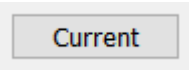

Optional filename (with full or relative path) of the associated preview image. Preferably square JPEG or PNG square image.

## 4 Find




Use this button to find in the file the id (value) on the left.

## 5 Current




Use this button to revert to the current (if any) data triplet.

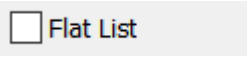
## 6 Content


Display of the file content, with the folder hierarchy as specified by # tags. Select any entry to fill the Value-Label-Image fields.

## 7 Include


Check this box if you want each data entry in Content (6) to mention the id before the label.

## 8 Flat List

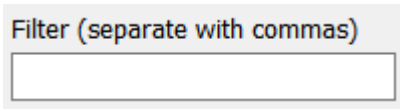



Check this box if you want the content to discard all folder membership.

## 9 Preview

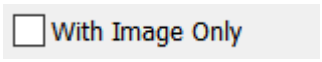
If the selected (or current) data entry has a preview image defined, will be displayed here.

## 10 Filter



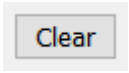

Enter here any motif searched in the file. Must belong to labels. Several motifs can be listed, separated with commas.

## 11 With Image Only



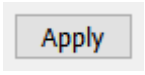

Check this box if you want the Content (filtered or not) to display only data entry with preview image.

## 12 Clear



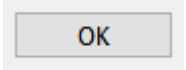

Clear the filtering and rebuild Content

## 13 Apply




Rebuilt Content according to the above filters.

## 14 OK




Keep the displayed data entry value id.

# Interface Tag

When the type is referring another Component or Data-Model class, specify here the name of the class with its own Dyn-UI.
The Interface will provide an Edit button for the variable to popup the remote Dyn-UI.
Use DSPL in the remote interface so that some values can be displayed in this one.

*ie:*

```
Complex my_value; //&& INTF[Complex]
```

For embedded interfaces, it is advised to set the default values of the interface variables locally than using the `setDefault` function.
For example, if a Component Foo uses a Data-Model Complex (above) defining two variables, sign and speed, then initializing these variables from Foo will be done this way:

```
Complex my_value; //&& INTF[Complex] DEF[sign(-1),speed(16.3)]
```

**DSPL** or **DISPLAY**: allow this variable value to be displayed on the interface, if the class is embedded into another interface.
For example, if a class foo (component or data-model or logic) contains 3 variables: x, y, z and that for each of them, `//&& DSPL` is
defined, then, if foo is used as a variable of another class, x,y,z values will be displayed between brackets.

*ie:*

```
int my_value; //&& DSPL
```

# Enum Tag

Similar to **LIST[]** but refering an existing `typedef enum` type defined in the code (and visible by the Code-Insight module)

*ie:*

```
long my_value; //&& ENUM[myValueEnum]
```

# Using Setters

**SET[...]**: specify the name given to the setter function for this variable. This setter will be used in the `setUserData` function. The setter code will be automatically added.

*ie:*

```
int my_value; //&& SET[setTemperature]
```

**NOSET**: setter function will not be generated and value will be directly assign in the `setUserData` function *(ie:* `foo = 17;`*)*
**USRSET** or **USERSET**: cancel automatic setter generation code to allow user to define its own.

*ie:*

```
int my_value; //&& SET[setTemperature] USRSET
```

# Keyboard

Keyboard can be used to accelerate some functions.
Here are the list of keys with the capability attached:

| | |
|---|---|
| Ctrl-O | Open a database |
| Ctrl-S | Save the database |
| Ctrl-X | Cut |
| Ctrl-C | Copy |
| Ctrl-V | Paste |
| Ctrl-D | Duplicate |
| Ctrl-F | Find |
| Ctrl-R | Replace |
| F2 | Scenario View |
| F4 | Refresh |
| F6 | Generate |
| F7 | Compile |
| Ctrl-F7 | Recompile All |
| F8 | Launch |
| F9 | Run |
| F10 | Pause |
| F11 | Stop |
| F12 | Hook Window |
| + | Increase Zoom |
| - | Decrease Zoom |
| Del | Delete selected |
| Backspace | Show category list |
| ESC | Close hint or window |
| Arrows | Move selected sprites |

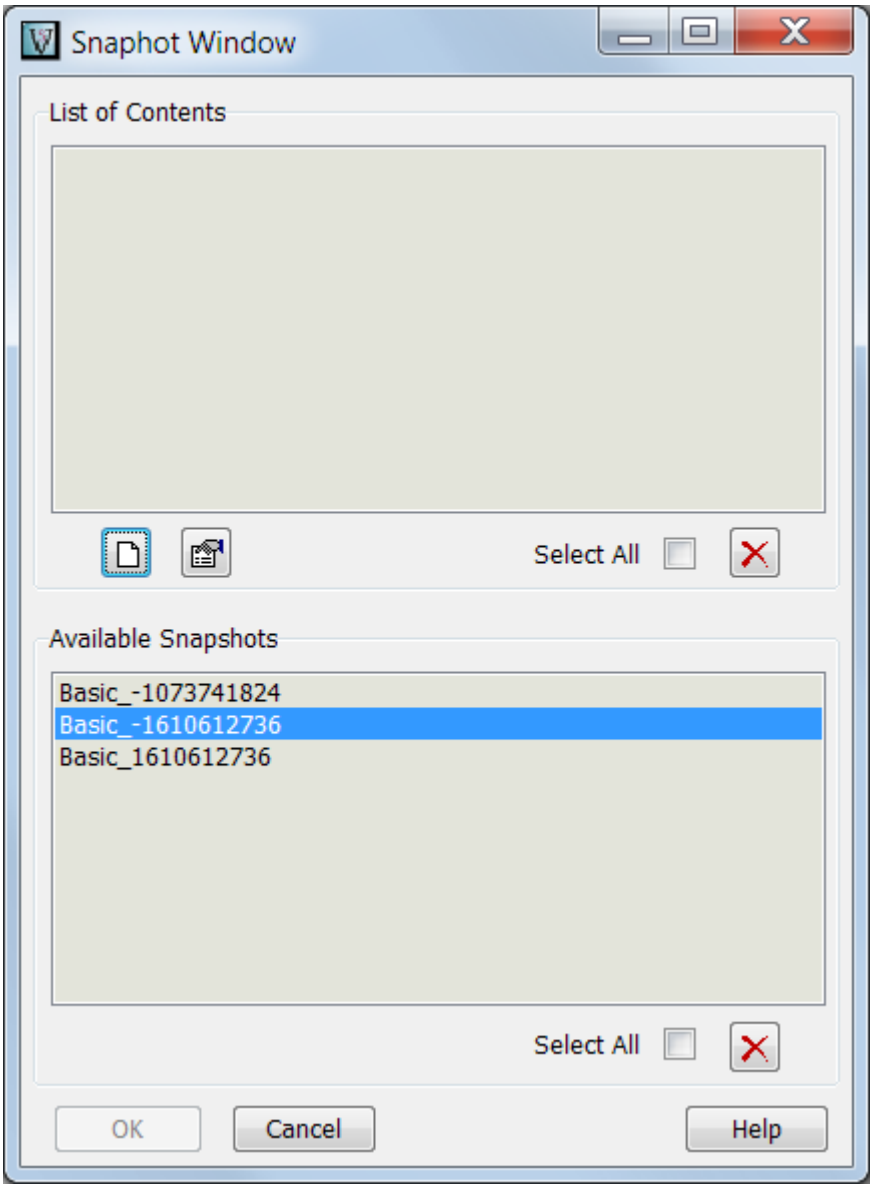
| | |
|---|---|
| | |
| f | fill/no fill toggle |
| Page Up | Map zoom in |
| Page Down | Map zoom out |
| Back Space | Go back |
| Space | Track Entity toggle |

# Snapshot and Restore

Snapshot/Restore is used to dump the simulation memory at a given point in time and restore (fly-out) from it at later time.

During the simulation run, use button to dump the memory into a file.

When the simulation is stopped, you can use the button to select from the list the Snapshot file to restore.




If Snapshot files are available for this database and current scenario, they will be listed in the **Available Snapshot List** panel.

Press the **Start** button then **pause** the simulation.

Select one of the snapshots in the list, then double-click to restore the simulation at this point in time.

Press **restart**.

# • Limitations

CS Edition only.

# • Common Errors

Snapshot/Restore is a complex mechanism that must be considered if user-modules or external modules are linked with the simulation engine.
These user-modules must be made compliant with the Snapshot/Restore mechanism in order for the simulation to restore properly.

**Misuse of this functionality can result in crash or memory corruption**.

# Command Line Options

vsTASKER can be started from a command line.
Below are the options:

```
/c db tp  // to create a db database base on template
/l db     // to load a given db database
/g        // to force generate code
/b        // to force build the engine
/s        // to load the associated engine
/ro       // to enable the read-only mode
/nosplash // start without splash window
/x        // to force exit
```

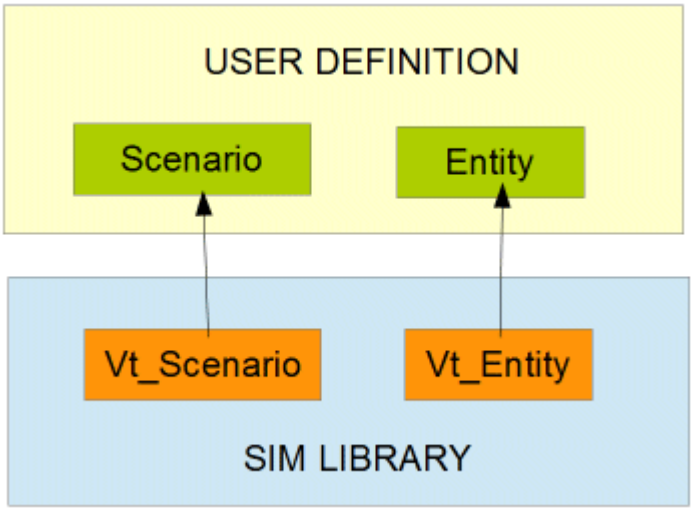
# General Overview

vsTASKER database hierarchy.

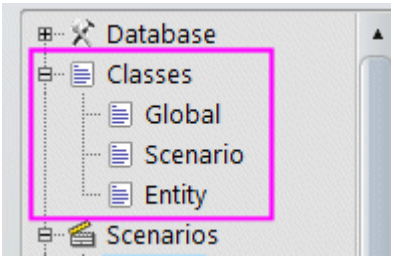
For a better understanding of the programming, refer to the Development Guide.

# Classes

Classes are used to expand a vsTASKER internal classes with user data and methods. This should be seen as a classic C++ class definition.



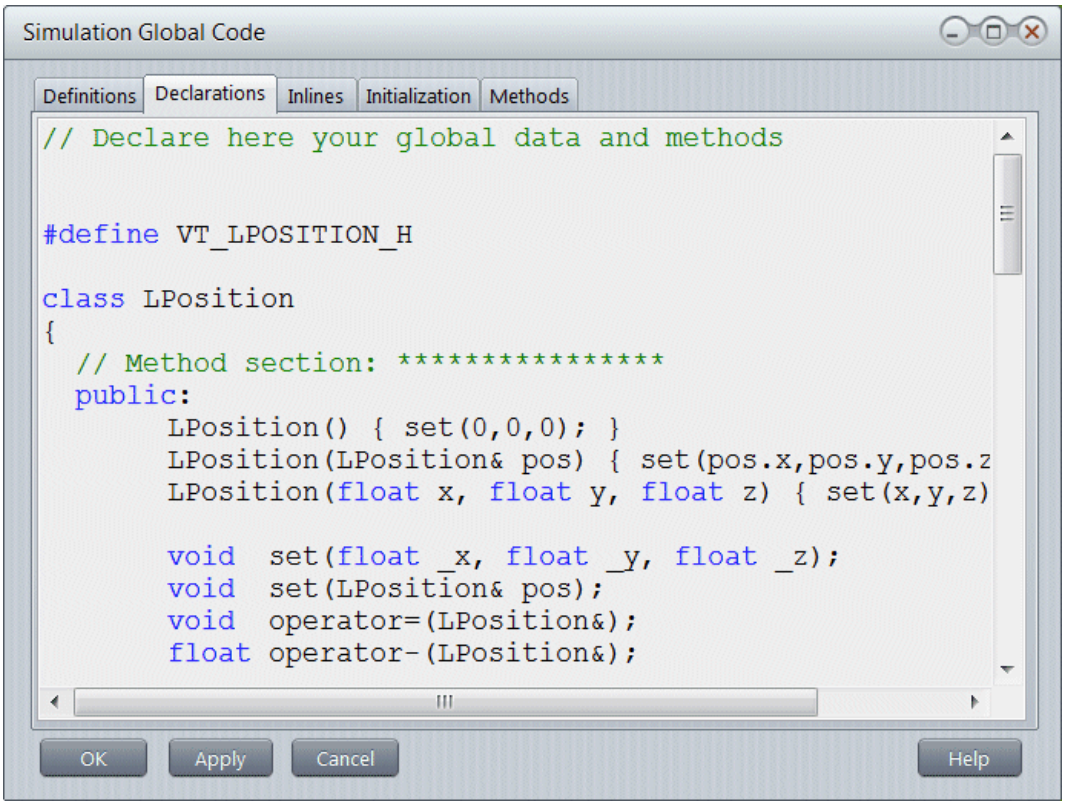

The **Sources** category (in Environment Tree-List and Diagrams) has three members:




Global
Scenario
Entity

# Source Code

As vsTASKER is a code generating tool, it allows the user to add its own.
To mimic the C++ class definitions, source code is organized with panels where a class definition is split.




## • Definitions

Put in this panel all `#include` (third party API), `#define` and `typedef` needed for the current Class.
*Definition* content is put without preprocessing in the beginning of the final header file, outside of any class definition.

## • Declarations

Put in this panel all **methods and data** you want to add in the current class (Scenario and Entity). For Global, use this section to list all static or global functions prototypes.

### Source Code

For Scenario and Entity, the *Declaration* part is added, without preprocessing, into the `Entity::` or `Scenario::` class definition in the header file. For Global, the *Declaration* content is put as is in the header file.

*No constructor nor destructor should be added here. Use instead the Initialization part.*

## • Inlines

Put in this panel all **inline methods** of the current Class (Scenario and Entity). There is a preprocessing of the *Inlines* content, so do not forget the keyword inline and the word `Scenario::` or `Entity::` before any method name.

For Global, specify here all inline methods of your user-defined classes (if any). Do not forget the inline keyword and the name of the class these methods belong to. For Global, the *Inlines* content is not preprocessed.

## • Initialization

This part is called automatically at initialization time (`INIT`), when simulation is run for the first time, every time the instance is reset (at restart) and finally at destruction (`CLEAN`) when the simulation is terminated. Stopping the simulation does not call `CLEAN`, only exiting the simulation calls `CLEAN`.

*Refer to the Developer Guide for more details on system phases/events.*

## • Methods

Put in this panel all **methods** of the current Class (Scenario and Entity). There is a preprocessing of the *Methods* content, so do not forget the word `Scenario::` or `Entity::` before any method name.

For Global, specify here all methods of your user-defined classes (if any). Do not forget the name of the class these methods belong to. You must also put into this panel all global or static functions you have defined earlier. You can also put all the `#define` and `#include` needed for the local global code only. For Global, the *Methods* content is not preprocessed.

# Global

**This is not a class**!

User should use the Global Code to add all his user classes, his global variables, functions and inlines or methods of his user-defined classes.

**Definition**, **Declaration** and **Inlines** contents are put without preprocessing in the final header file.

**Initialization** part can be seen as a function called according to phases (`INIT, RESET, CLEAN,` see here) and **Methods** part is added without preprocessing in the final source code, at the beginning.

# Scenario

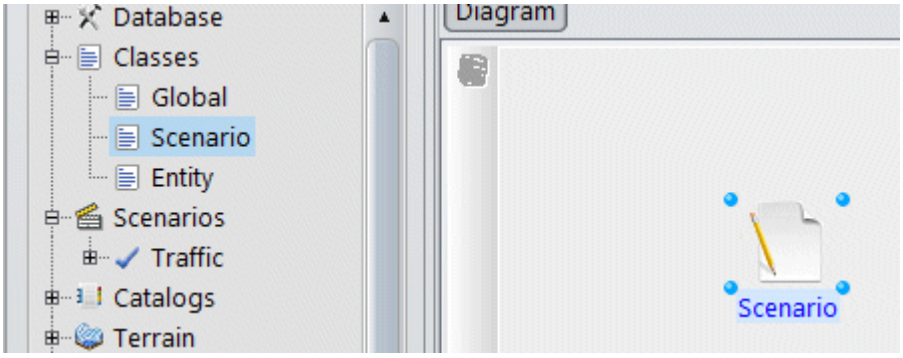
This defines the Scenario Class that inherits from Vt_Scenario vsTASKER base class included into the vstasker.lib library.

The Vt_Scenario expects the Scenario class to exist and this one is code generated by vsTASKER. Scenario class embeds all user definition that will be used in the database.

The Scenario class is also part of the Scenario Template.
This class will be instanciated for each scenario of the database.

Another solution than using this class would be to create either *Components* or *DataModels* and attach them to the Scenario **Player**.
It is faster and more convenient to directly add members and methods to the Scenario class and the choice to use one way or the other is let to the programmer.

Using this class is lower lever than using a Model (*Component* or a *DataModel*) but simpler. The main drawback is that members (variables) will not be dynamically modifiable by the user like using a Model.



# Entity

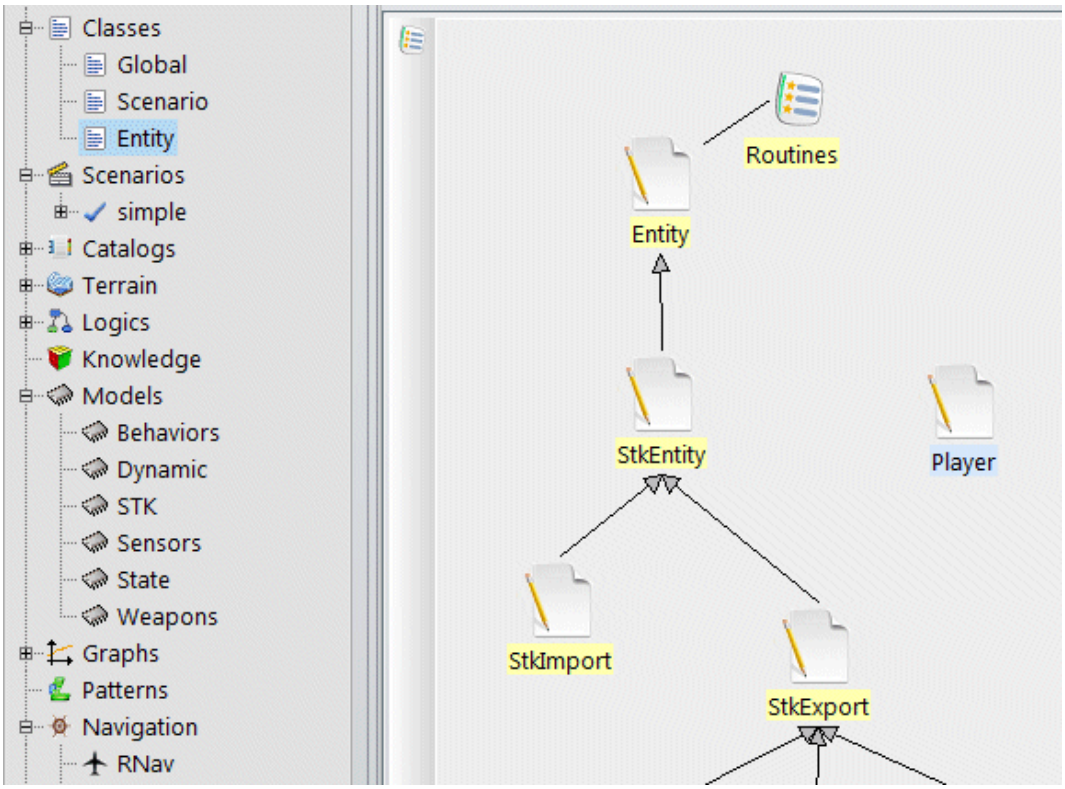
This defines the Entity class that inherits from the Vt_Entity vsTASKER base Class.

The Vt_Entity expects the Entity class to exist and this one is code generated by vsTASKER.
Entity class embeds all user definition that will be used in the database.

The Entity class is also part of the Entity Template.
This class will be instanciated for each entity of the scenario.

Another solution than using this class would be to create either *Components* or *DataModels* and attach them to the Entity **catalog**.
It is faster and more convenient to directly add *members* and *methods* to the Entity class and the choice to use one way or the other is let to the programmer.

Using this class is lower lever than using a Model (*Component* or a *DataModel*) but simpler. The main drawback is that members (variables) will not be dynamically modifiable by the user like using a Model.




- **Player**

The Player class is similar to the Entity class except that it will have no dynamic nor export component nor position.
Only one Player will be instanciated and attached to the scenario.

# • Inheritance

For both Entity and Player classes, user can specialize children classes and set any Entity (from the *Scenario* or the *Catalog*) to be linked with the children class.
Different classes can then be defined including their own data members and methods.

Because these child classes have their own Initialization phases (called after parent ones), specific values (or methods with specific parameters) can directly be called from there. Also, it is possible to set some Entity virtual methods to specialize them.

# • Routines

See here

# Events

An Event is a named signal that carries (or not) user defined values. An Event is initiated by an object and is propagated to other objects that can react to Events (observers).
Propagation is limited by the scope of the triggering. The propagation is also limited at one cycle.

*If a reaction to one event raises another event, the latest will be processed at the next cycle. This behavior can be overridden with the risk of infinite loop.*

## • Raising an Event

**From the GUI**, see here.

**From the code**, do the following:

```
<initiator>.raiseEvent("event name", { T_Ptr, pointer, Scope });
```

For example, if the entity tank must send the event Fire to all scenario, do the following:

```
Vt_Entity* tank = S:findEntity("tank");
tank->raiseEvent("Fire", SCENARIO);
```

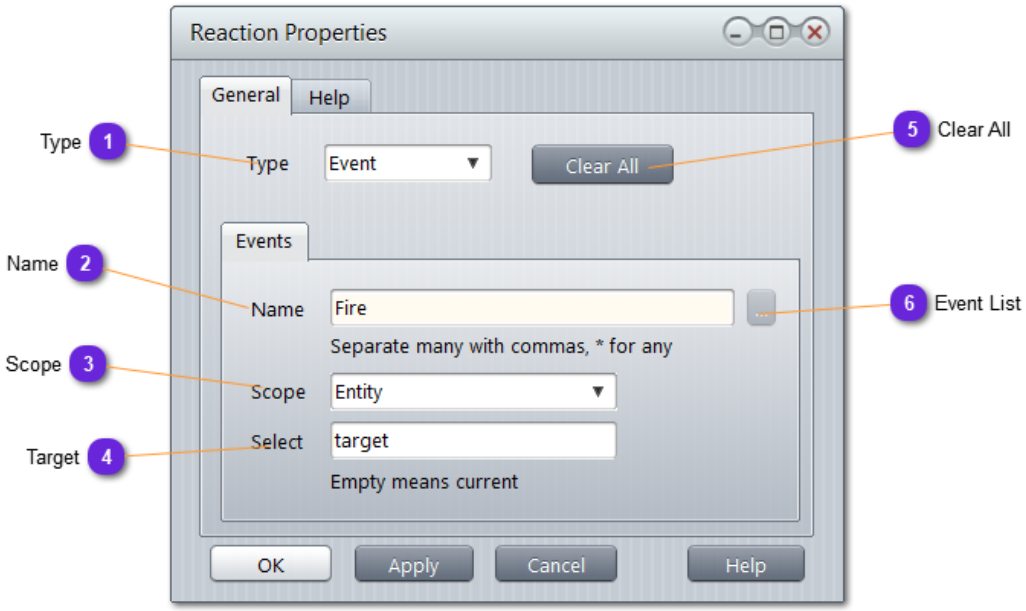
To specifically informs the entity target that it has been fired at by tank, do the following:

```
Vt_Entity* tank = S:findEntity("tank");
Vt_Entity* target = S:findEntity("target");
target->raiseEvent("Fire", T_Ptr, tank);
```

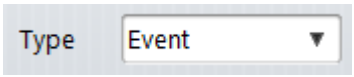
## • Observing an Event

For now, it is only possible to observe an Event from an Entry Point, under Reaction mode. Components and Data-Models can also react to events.

# Raising Event

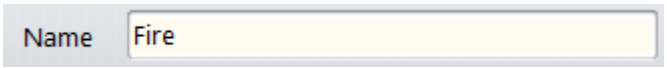



## 1 Type




Select here Event to activate the proper page below.

## 2 Name




Name of the event. As a string, spaces are not allowed.

## 3 Scope

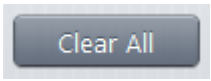

The scope defines the hierarchical regions where the event will be propagated.
You can get the enumeration ScopeType in include/vtypes.h

- KCONTEXT: propagated to all rules of the context
- LGROUP: propagated to all objects of the group
- LOGIC: propagated to all objects of the logic
- KNOWLEDGE: propagated to all contexts of the knowledge
- CMODEL: only to the selected data-model
- ENTITY: propagated to all behavior objects (logic, knowledge, model) of the entity
- SCENARIO: propagated to the player and all entities and their behavior objects
- DNETWORK: propagated to all distribution objects
- NSOCKET: only to the selected socket
- NFEDITEM: only to the selected federate item
- PLAYER: all player behavior objects
- COMPONENT: only the selected component
- MASTER: propagate to the master entity (from a unit member)
- MEMBERS: propagate to all the members of a unit (from a master)
- BATCH: only to the selected batch object
- NDISPDU : only to the selected PDU object
- DPATTERN: only to the search-pattern object
- FORMATION: to all members (including masters) of a formation
- NFPLAN: only to the selected flight-plan
- SPRITE: only to the selected sprite
- NSTNMSG: only to the selected STANAG message
- NCGIMSG: only to the selected CIGI message
- NORBIT: only to the selected Orbit
- NPACKET: Only to the selected Packet

## 4 Target


When a scope implies a specific target (entity, batch, socket...), enters the name of the object here.

## 5 Clear All




Reset all fields to default.

## 6 Event List

*Not available yet.*

# Facts

A Fact is a persistent Event.
A Fact must be stored into a Fact database.
Many objects have their own database.
Each scenario, entity, logic, knowledge, component (...) have their own.

## • Raising a Fact

From the GUI, see here.

From the code, do the following:

```
 <initiator>.raiseFact("fact name", duration);
 <initiator>.raiseFact("fact name", { T_Ptr, pointer, duration,
Scope });
```

For example, if the entity tank must store the fact Damaged to its own fact database, for 1 hour after a hit, do the following:

```
 Vt_Entity* tank = S:findEntity("tank");
 tank->sFact("Damaged", 3600);  // in seconds
```

## • Checking a Fact

Because a fact is persistent (longer than an event), it is possible to do it from the code or from the GUI.

**From the code**: To check if the entity tank has been damaged, do the following:

```
 Vt_Entity* tank = S:findEntity("tank");
 if (target->hFact("Damaged")) ...
```

**From the GUI**: see here

## • Deleting/Updating a Fact

To **update** a fact from a database, by changing its duration value, do the following:

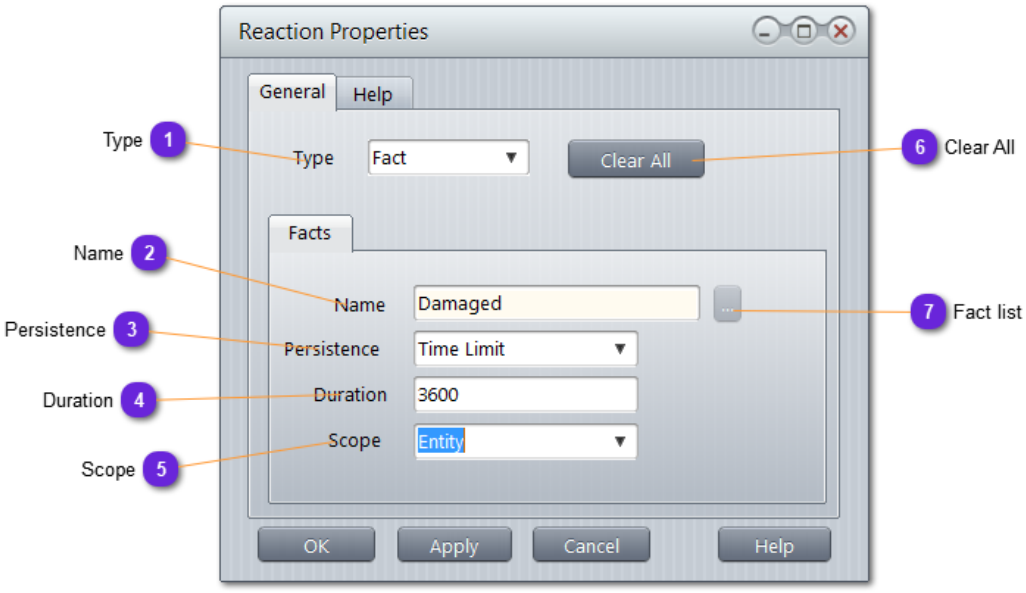
### Facts

```
Vt_Entity* tank = S:findEntity("tank");
tank->sFact("Damaged", 1000);
```

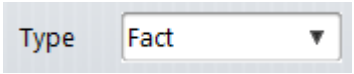
To **remove** a fact from a database, do the following:

```
Vt_Entity* tank = S:findEntity("tank");
tank->dFact("Damaged");
```

# Storing Fact

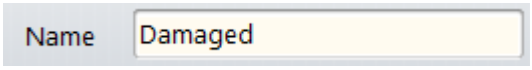



## 1 Type



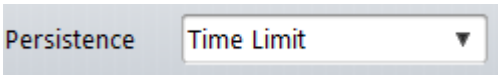

Select here Fact to activate the proper page below.

## 2 Name




Name of the fact. As a string, spaces are not allowed.

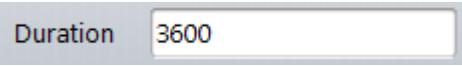
## 3 Persistence




A fact remains in the database for a specific time. It will automatically be removed when time is elapsed.

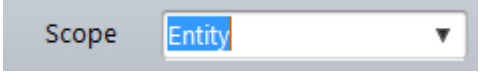
- Unlimited: fact will remain in the database until manual removal
- Time Limit: time must be specified

## 4 Duration



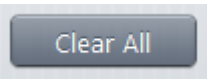

For Time Limited fact only. Specify here the life time of the fact in seconds.

## 5 Scope




Specify which database will be used to store the fact.
The database selected is always relative to the object raising the fact. For i.e., if the object raising the fact is a logic action, then selecting Logic for the scope will store the fact into the current logic (holding the object). If Entity is chosen, then the selected fact database will be the one of the entity holding the logic holding the object.

## 6 Clear All



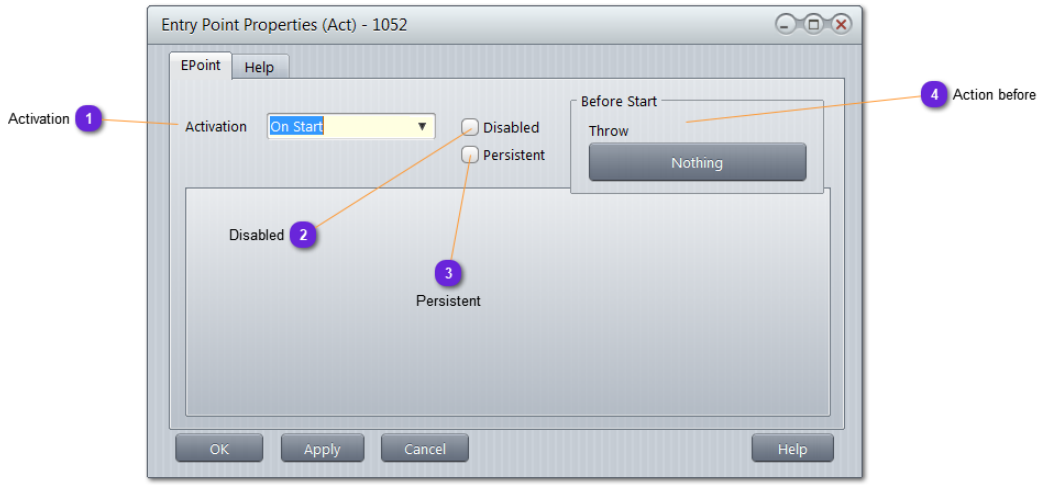

Reset all fields to default.

## 7 Fact list

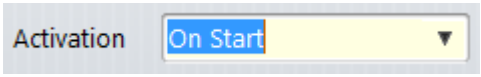
*Not available yet*

# Entry Point

An Entry point is needed to start a logical flux, a behavior or an iterative object on a diagram.
It is activated under specific conditions.
For now, only one condition per Entry Point can be defined



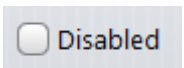

## 1 Activation




Select on which condition the Entry Point must activate or deactivate the object it is linked to:

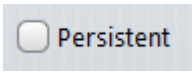
- On Start: At simulation start, as soon as possible after start (not available for Exit Point)
- On Time: See here
- On Reaction: See here
- On Condition: See here

## 2 Disabled



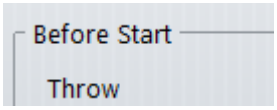

Check this button if you want this Entry Point to be switched off (temporary) without removing it from the diagram.

## 3 Persistent




By default, when an Entry Point is triggered, it will not be reactivated even if the same condition occurs.
If Persistent mode is selected, the Entry Point reactivate itself automatically after start and is ready at next cycle.
This might lead to unwanted behavior.

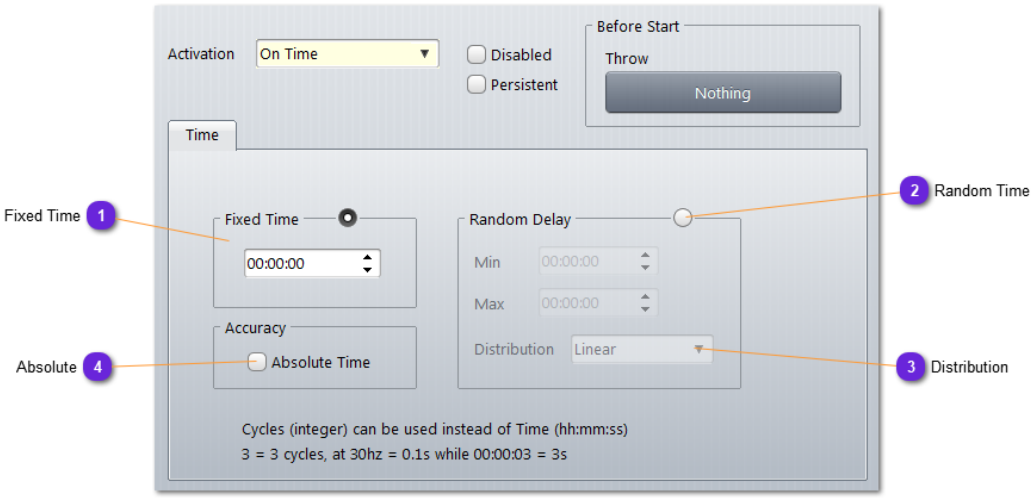
## 4 Action before



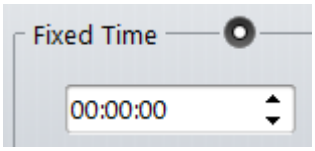

Specify here the Event or Fact that will be triggered before activation of the Entry Point (optional).

# On Time

When a specific or random time is elapsed or matches the current simulation time, the Point is triggered.

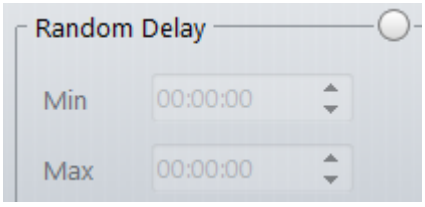



## 1 Fixed Time



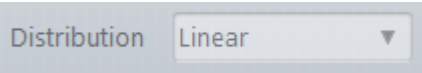

When Fixed Time is selected, as soon as the specified time is elapsed, the Point triggers.
For i.e: if the Point is inside a Logic with a Fixed Time set to 10 seconds. If the Logic is triggered 1 minute after the simulation start, then the Point will be triggered 1 minute 10 seconds after the simulation start.

## 2 Random Time



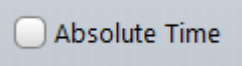

A random value (according to the Distribution) will be taken at Point initialization. This random value between Min and Max values will act as Fixed Time.

## 3 Distribution



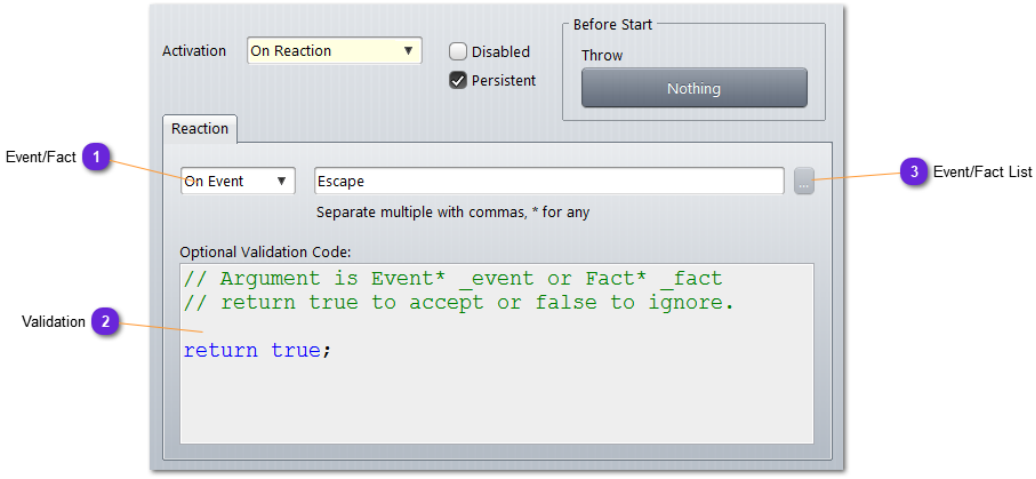

Linear: Random values are distributed equally between Min and Max
Gaussian: Random values are distribution according to a Gaussian curve between Min and Max. Mid-term has the highest probability of occurrence.

## 4 Absolute

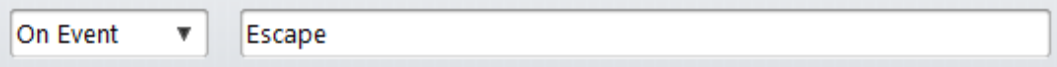



If this option is checked (for Fixed or Random Time), the specified Time is according to the simulation time.
Point triggers when the simulation time is greater than the given fixed or random time

# On Reaction




## 1 Event/Fact




Specify here the Event (or the Fact existence) that will initiate the triggering of the Point.
In case of an Event, the Point will register as an observer for events and for each one raised into the Point scope, triggering will happen.
If case of a Fact, the Point will continuously check the Logic fact database for existence of any Fact mentioned.

On the text field, write the Event or Fact (if many, separate them with commas) that will trigger the Point.

## 2 Validation

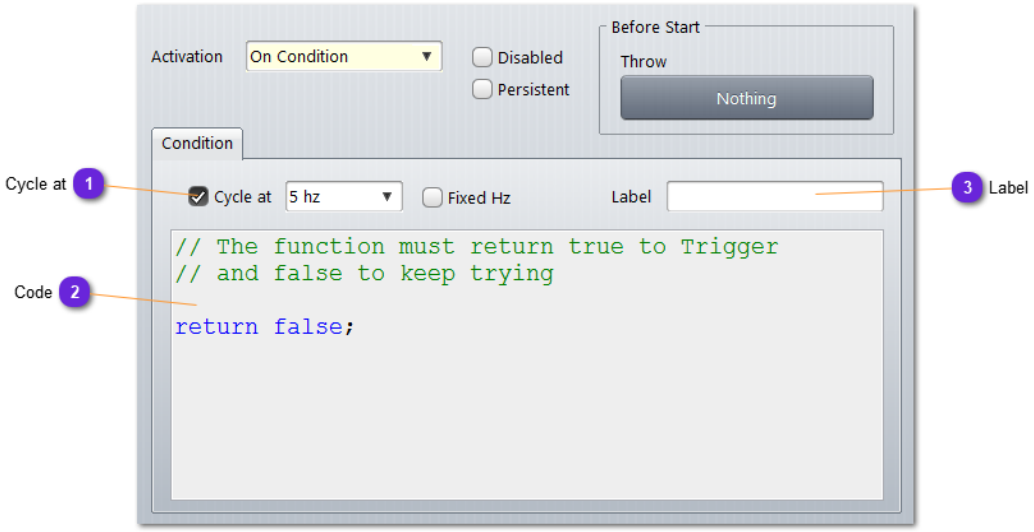
```
// Argument is Event* _event or Fact* _fact
// return true to accept or false to ignore.

return true;
```

Write here any code that will accept (return `true`) or not (return `false`) the catch of the Event or the existence of a Fact.
This can be any condition the user might decide, for ie.: *Cannot accept the take-off event before engines have run for minimum 7 minutes.*

## 3 Event/Fact List

*Not available yet*

# On Condition




## 1 Cycle at


A condition is a piece of code that will be evaluated (called) at a given frequency.
If Fixed Hz is selected, the frequency selected is the wall clock frequency and not the simulation engine clock.
If Cycle at is not selected, the Condition code will be evaluated only once.

## 2 Code

```
// The function must return true to Trigger
// and false to keep trying

return false;
```

Put here the condition code that will return either true (condition passes, triggering of the Point) or false (condition fails, no trigger)

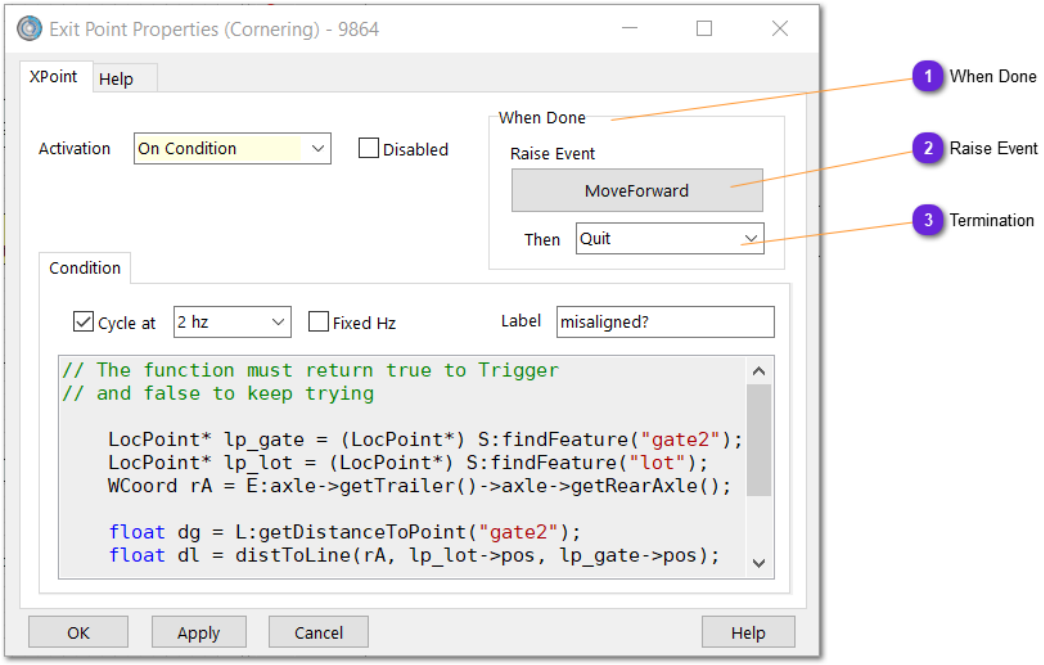
## 3 Label


Optional label that summarize the condition and will be used on the diagram panel.

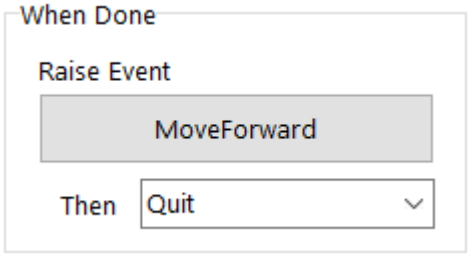
### On Condition

# Exit Point

An Exit point is needed to end the process of an iterative object.
It is activated under specific conditions.
For now, only one condition per Exit Point can be defined. See here for the condition list.

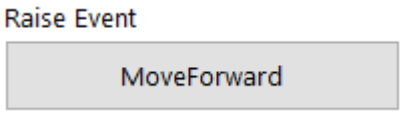



## 1 When Done




This part is active as soon as the Exit Point has triggered.

## 2 Raise Event




Specify here the Event or Fact that will be triggered before termination of the Exit Point (optional).

## 3 Termination


Select from the drop down list the termination mode to be applied to the object the Exit Point is attached to:

Suspend: the iterative object (task or group or behavior) will be **frozen** until next **resume**
Done: the iterative object will **finish** and its exit **arrow** will be **triggered**
Abort: same as with Done but the exit arrow will **not** be triggered
Quit: the environment belonging to the Exit Point will be **exited**. If inside a *Group*, the Group will be left (Done); if inside a *Logic*, this latest will be Done; if attached to a *Behavior*, all *Logic* behaviors will be terminated.

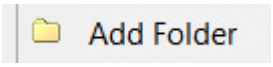
# Folders

For some Categories, when a lot of items has been created, the Diagram panel can become messy. This happens when a lot of Logic (but not only) have been defined. The user can create as many Folders as needed and there is no limit on the depth of the hierarchy (folders can hold folders).

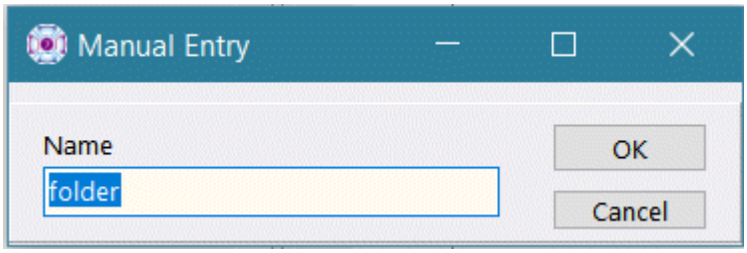
## • How to Use

### Create one folder

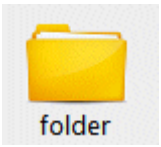
When this feature is available, use the icon  or the right-click popup **Add Folder** menu:




Then give it a name:



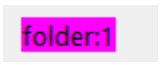

Then you'll have one:




You can now select the icon to move it around, right click to remove or rename it (make sure you do not have two folders of the same name in the same level).

### Opening a folder

Just double-click the icon. You know you are inside a folder because its name (the full path in case of nested folders) is mentioned in the top left of the panel:



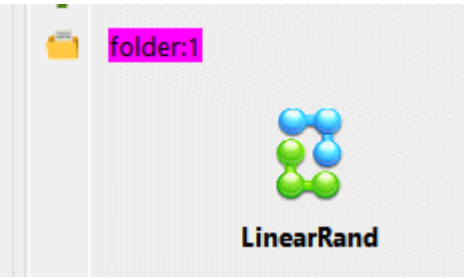
## Closing a folder

To move out or close a folder (and go up one level), just double-click the **background** of the panel.

## Moving items in

In order to put an item (or another folder) inside one, do like in Windows: select the item you want to move and drag it over the destination folder. When the folder image grows and opens, release the mouse button. Here is an example with a logic folder:

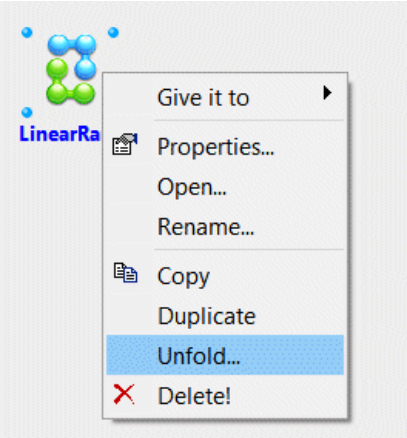

Now, you can double-click the folder to open it:




When inside a folder, you will notice the mention in the upper left part of the panel, in magenta. The number at the right of the folder name is the number of items the folder contains.

## Moving items out

Moving out a selected item is done using the right-click **Unfold...** menu:




## Deleting folder

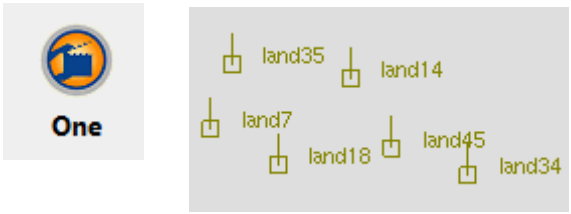
A folder cannot be deleted with item inside. It must be emptied before deletion. Use the Del key (when icon selected) or use the right-click popup **Delete!** menu option.

# Scenarios

A scenario is defined by a set of entities, terrain layers, plans, events and behaviors.
It also uses optional layers like features, meshes, roads, winds and RNav elements.
It will use some distribution items like sockets, DIS, HLA or STANAG 4586 messages.

A scenario holds a special entity called Player which has no real existence on the game play but can be seen as the master or operator.
it can have Models attached to it, Logics and Knowledge. Only Dynamic is forbidden as the Player has no position.

A scenario defines a situation at time t0.




Entities are represented in the Scenario environment (terrain or map) using symbols specified in the Entity Property Window. Each Entity can be selected using the mouse, and dragged somewhere else. If double clicked, the Entity property window is shown.

When an Entity is selected on the Map or Terrain, additional Behavior panes are available.
Can be used to delete one or a set of entities.

## • Environment

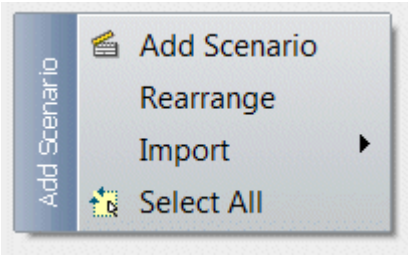

When a scenario is selected in the Environment tree list, it becomes active.
The selected scenario is marked with the sign and on the Diagram panel, it is shown plain color (inactive one is grayed out).

Clicking on a scenario name (Environment) or double clicking its symbol (Diagram) is enough to select and activate it.

The terrain map always display the selected scenario.

A database must always have at least one scenario.

Right click on the Diagram panel when a scenario is selected in Environment:




**Add Scenario**: create a new scenario and add it to the current database

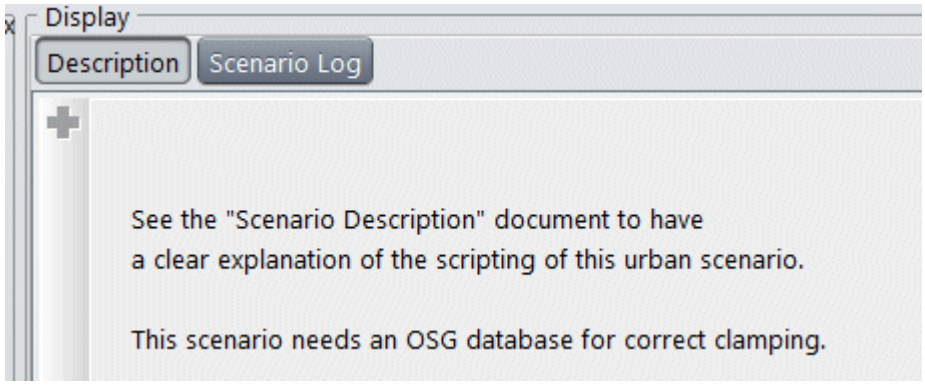
**Rearrange**: organize the symbols in the pane

**Import**: list all exported scenarios (/Data/Shared)

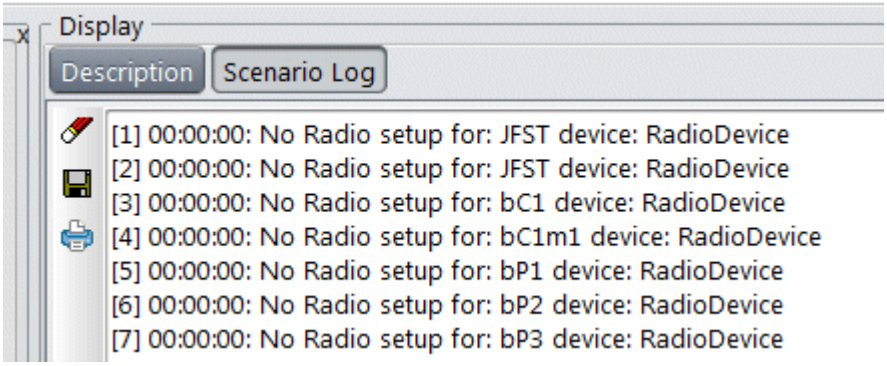
**Select All**: select all symbols (for move or deletion)

## • Diagram

When a scenario is selected into the Environment, the diagram shows the following panes:




Description: displays the content of the text Description. It is a good idea to display here the purpose of the scenario, how to run it and what to expect.



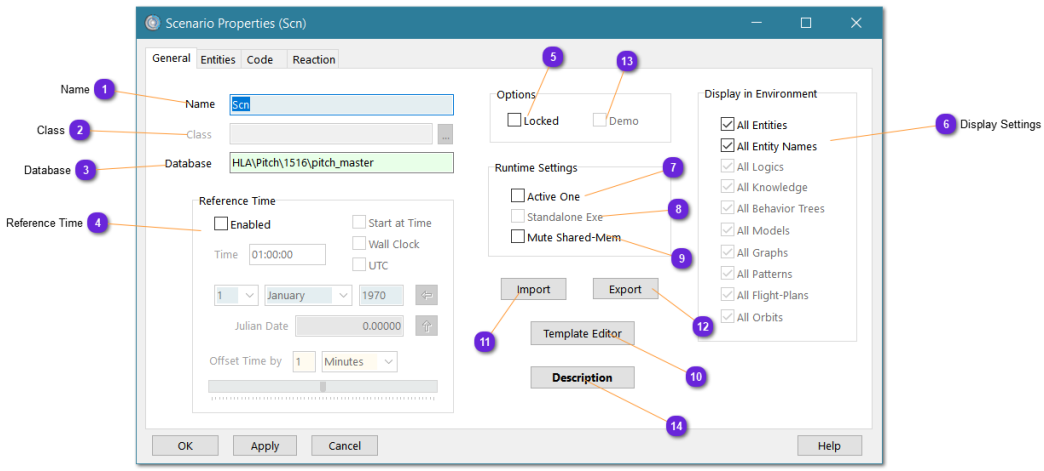

Scenario Log: displays all the Log entries (including the time) sent by the SIM engine using the `S:log("string");` function.

Erase the content of the Log (will be kept until the next simulation start)
Save the content of the Log into a file
Print the content of the Log

# Properties

Property window is called by double clicking on the terrain map background, in Terrain mode, or from the contextual menu (mouse right click) or even using the  button once the scenario is selected on the Environment list.

# General



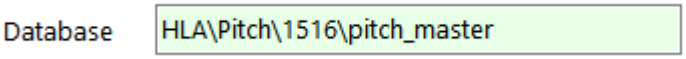

## 1 Name


Name of the scenario. Must be unique.

## 2 Class


In case the scenario inherit from another child class of the main Scenario class, specify that here using ...
User can change the class at any time; the class name will be use at the code generation stage.

## 3 Database

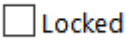



Read-only database name, including the path (under /data/db/) where it is stored.

## 4 Reference Time

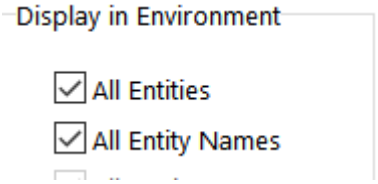

See here.

## 5 Locked




When selected, the scenario will not be modifiable (renaming, adding, moving or removing entities).

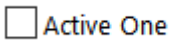
## 6 Display Settings




Select here what you want to see displayed on the Map or the Environment list.
Some scenarios might be set in such way that the Environment list is kept minimal or that terrain is left empty at design and runtime.


*Only the first two options are used for now.*

## 7 Active One

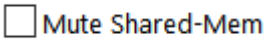



Specifies is the current scenario will be the first one to be called at simulation run. If only one scenario defined in the database or if the first scenario of the list must be the first to start, needless to set it true (harmless to do otherwise).

## 8 Standalone Exe

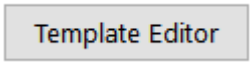

This option tells the code generator to produce a simulation engine that will not wait for the GUI to manually start the simulation (using the button). The Active scenario will be loaded and started as soon as the simulation engine will be run. If Active is not set, the first scenario of the database will be chosen.

## 9 Mute Shared-Mem



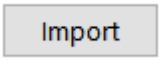

If selected, the shared-memory will not be filled by the simulation engine (although the segment will be created). This can accelerate the simulation and will avoid memory buffering if no GUI (reader) is connected to the shared-memory. If this option is selected, the GUI will not be able to monitor the activity.

## 10 Template Editor



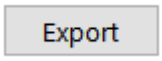

Call the editor in order to save the current scenario into the database Templat, as a template scenario.
Same as using the menu Export::Template::Save-as

## 11 Import




Call the importer to load from a file a saved scenario and replace the actual one (including the name). Data/Shared directory is used by default.

## 12 Export




Call the exporter to save the current scenario into an ASCII file (including some references) in order to share it with other players (when cut & paste cannot be used). Data/Shared directory is used by default.

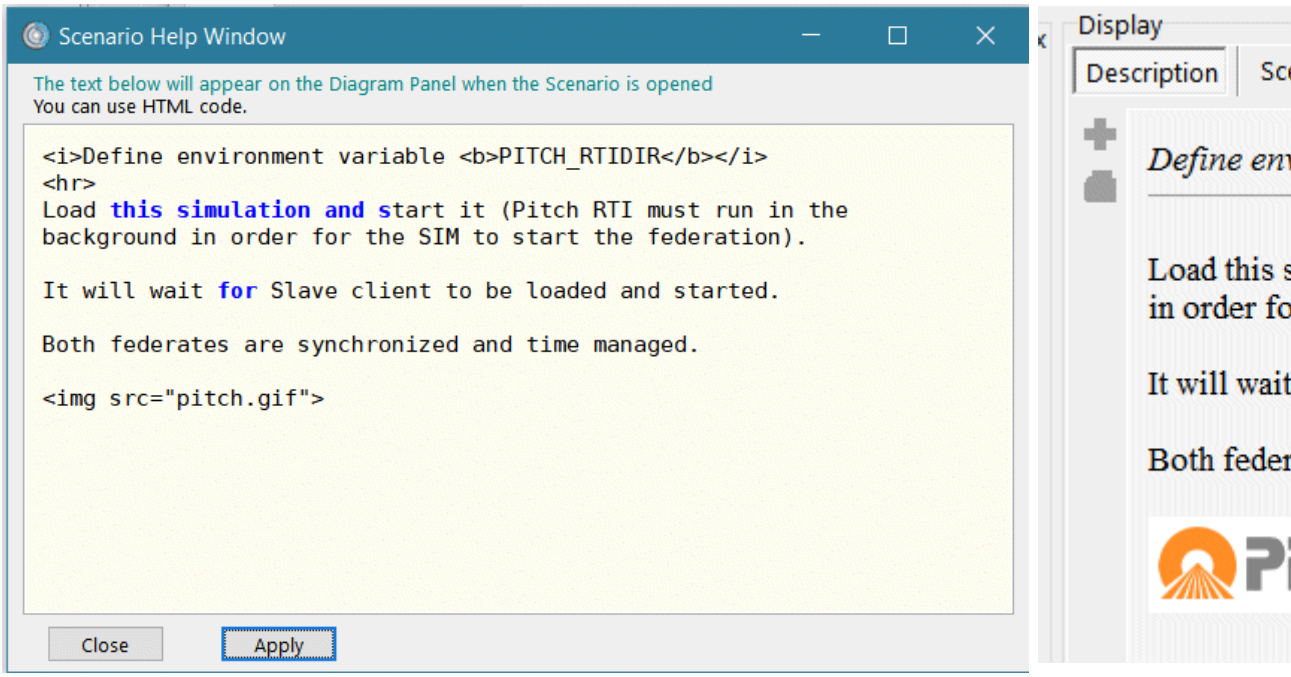
## 13 Demo


Activated and ticked only for demo only scenario, meaning it cannot be modified nor saved.

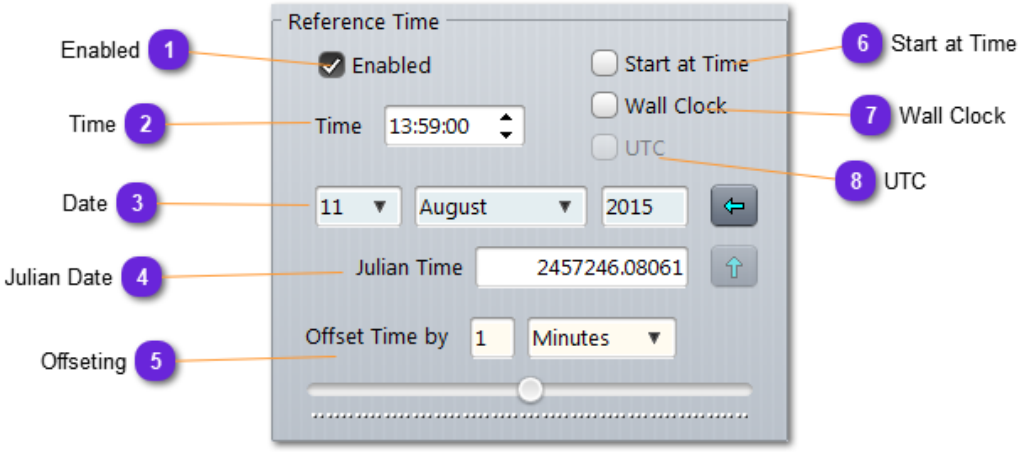
# 14 Description

**Description**

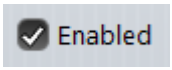
Pop up a new (modal) window for the user to write a description or user guide for the scenario. The result will be displayed on the side panel, as an HTML page, so, content of the description can include HTML tags or even, a copy paste of an HTML page designed in DreamWeaver (for example) :



# Reference Time



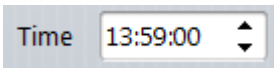

## 1 Enabled




When enabled, the reference time (date time) will be used instead of the default 00:00:00 time.
Simulation engine will add the reference time to the default time.

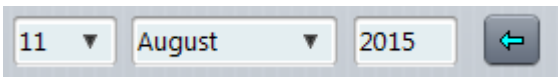

*Date time is not yet useable in the time related trigger.*

## 2 Time




Specify here the reference time for the calendar date

## 3 Date



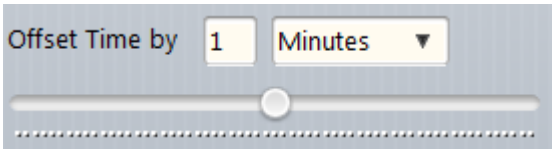

Specify here the reference calendar date
Use  to get the current computer date.

## 4 Julian Date


The above time + date gives a Julian date that is expressed in terms of days since noon Universal Time on January 1, 4713 BCE

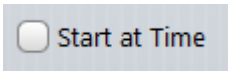
Use ⇧ to update the calendar time when the Julian date value is modified.

## 5 Offseting



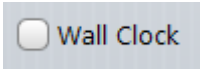

Specify here the offset value for the slider unit below. Each mark will then offset the above date from the value, positive on right and negative on left. Cancel button will revert to the initial date. Ok will valid the new offset date.

## 6 Start at Time



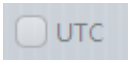

When checked, the simulation will be paused at start and wait for the given reference time to match the local computer time before resuming.

## 7 Wall Clock




When checked, the local computer date is used at each simulation start.

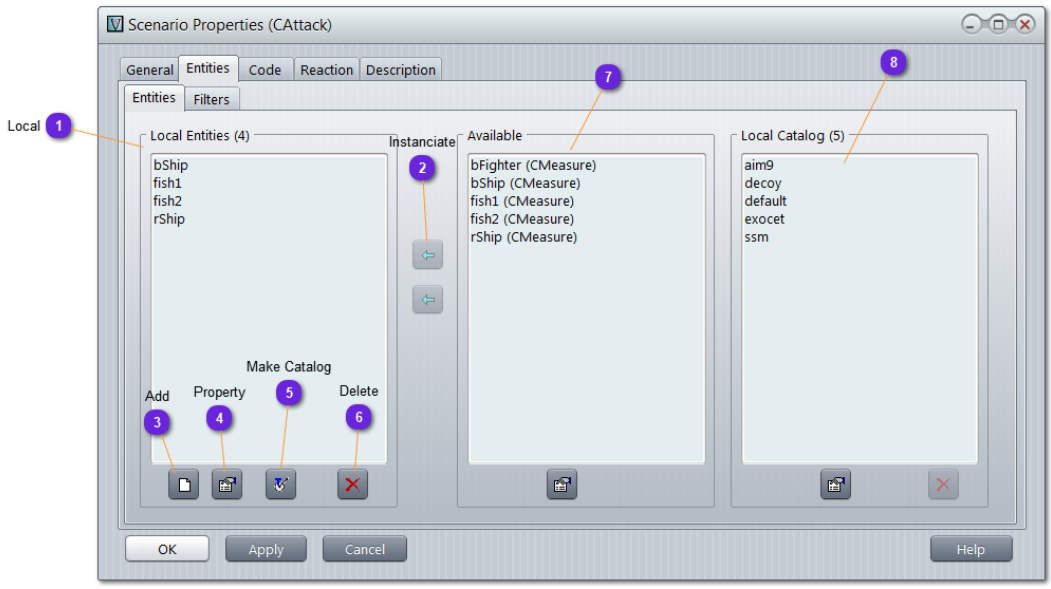
## 8 UTC



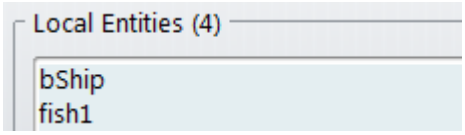

Check UTC if the UTC time must be used instead of the local computer date.

# Entities

See Entity definition here.




## 1 Local




List of all entities instanciated into the current scenario.
Multiple selection is available using keyboard Shift and Ctrl keys.

## 2 Instanciate

Copy/paste the selected entity from the Available or Catalog list to the current scenario.

## 3 Add

Add one entity from the template list.

## 4 Property

Call the property window of the list selected entity.

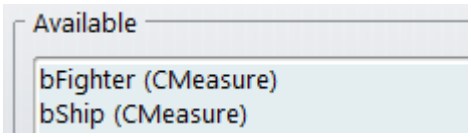
## 5 Make Catalog

Copy the selected entity/entities to the Catalog list.

## 6 Delete

Delete the selected entity/entities.

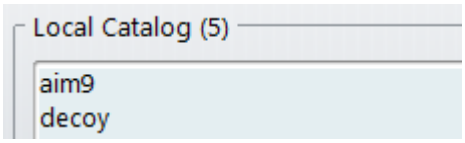
## 7 Available




List of all entities available in the other scenarios (in parenthesis) that can be copied into this one.

Select any of them and use the ⇦ button.

## 8 Catalog

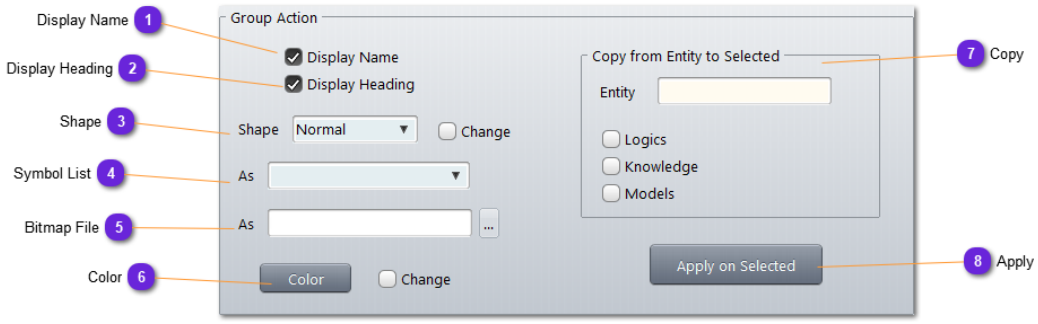



List of all entities available in the Catalog that can be copied into this one.

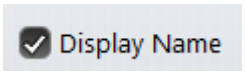
Select any of them and use the ⇦ button.

# Filters

This panel works only for selected entities. Use the map to select them first.

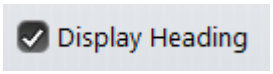



## 1 Display Name



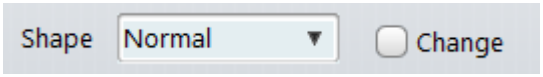

Show/Hide the name on the map for selected entities.

## 2 Display Heading



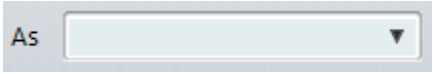

Show/Hide the heading arrow on the map for selected entities.

## 3 Shape




Set the 2D shape to the selected entities.
If Change is checked, the setting will be applied.

## 4 Symbol List




If shape is referring symbol, use this list to select the one to use.

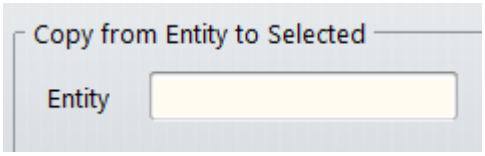
## 5 Bitmap File


If shape is referring a raster/bitmap file, use this selector to select which.

## 6 Color

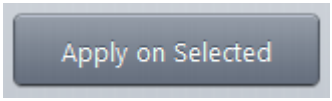

Set the color of the selected entities.
If Change is checked, the selected color will be applied.

## 7 Copy




Give here the name of the source entity you want to use to overwrite the selected one.
If Logics is checked, all source entity logics will be copied.
If Knowledge is checked, all source entity knowledge will be copied.
If Models is checked, all source entity models/components will be copied.

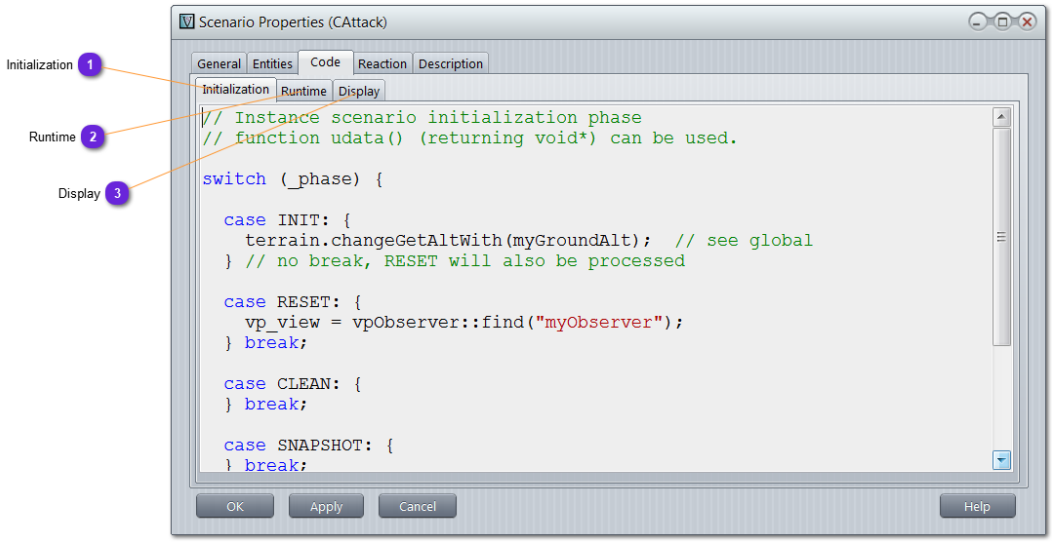
## 8 Apply



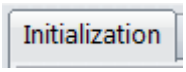

Once all the settings are done, use this button to apply.
Cannot be undo.

# Code

This code part will belong to the scenario only and will be called after the scenario class code.
See here.



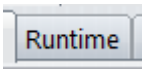

## 1 Initialization




This code is called automatically at system events.


*Refer to the Developer Guide for more details on system phases/events.*

## 2 Runtime

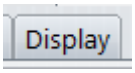



This panel contains the Scenario runtime code. This code is called at the RTC frequency specified in Database (Runtime) but is not called in Freeze mode.
The Scenario implementation code is a function that must return a value:

- **AGAIN**: the Scenario is called again at next cycle.
- **DONE**: the Scenario finishes normally. All Entities are terminated. Simulation stops but can be restarted.
- **EXIT**: the Scenario finishes normally, simulation stops and exits.


*The user can put whatever C/C++ code he wants there. This code can access all vsTASKER API, all included third party APIs and all user designed scenario and model data of the current database.*

## 3 Display



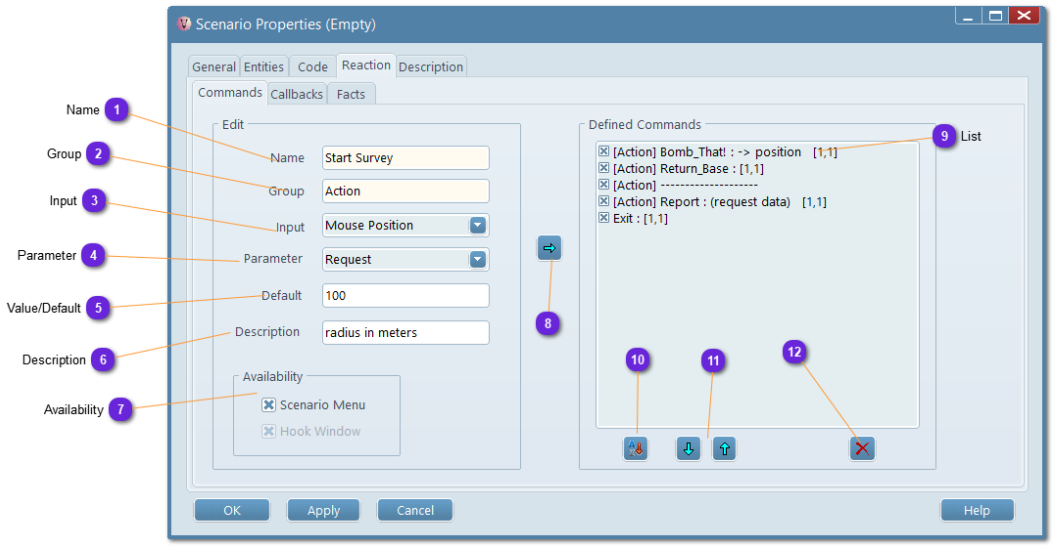

This panel contains the Scenario display code. This code is called two (2) times at the RTC frequency specified in Database (Runtime), even during the Freeze mode. It is a good idea to put here the Scenario visual representation on the OpenGL window (default) or the 3rd party application viewer.

- **BEFORE_DRAW**: Called before Entity Draw functions are called.
- **AFTER_DRAW**: Called once after Entity Draw functions are called.

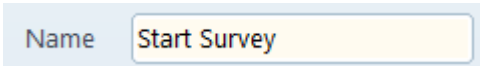
# Commands

A Command is a user-defined Event that will encapsulate a code (Callback).
Using commands is a simple and efficient way to create a user specific menu with all commands listed.

*For now, there is no way to show/hide a command nor to display them hierarchically. This feature might come in next versions.*



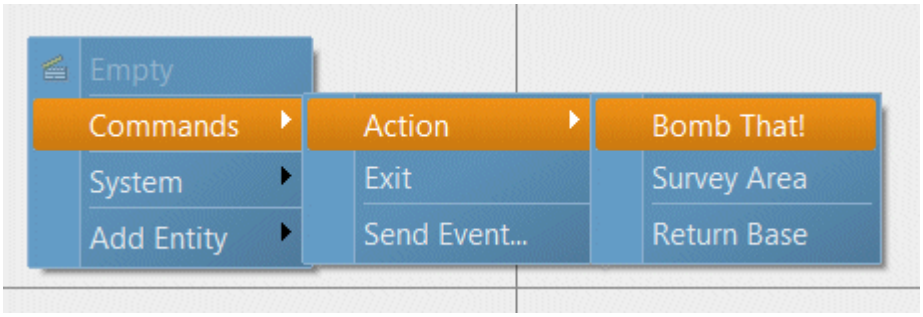

## 1 Name




Name of the Command (and name of the associated Event).
To add a **separator** in the generated menus (or submenu), just enter **-** (minus sign)

## 2 Group

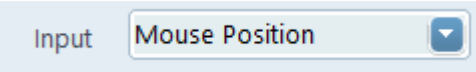

In order to group similar commands into the same submenu, use this field to create a submenu in the runtime command menu (see image below). Separators can be added using the **-** name with the specified group.





*For the moment, multiple submenus are not possible, only one level. Names must be unique, even if belonging to different groups.*

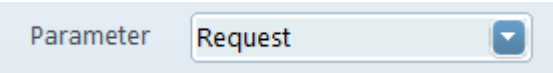
## 3 Input




Specify here if a selected (object) must be attached to the command:

- Nothing: no input;
- Mouse position: only the coordinate will be attached;
- Mouse selection: only the selected object will be attached (if any).

## 4 Parameter



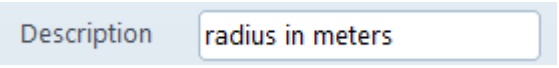

Specify if parameter(s) must be used after the command is called from the menu.

- None: no parameter;
- Static: use only the parameter line in the Value field below;
- Request: use the Default parameter line and tell the user to validate or change it.

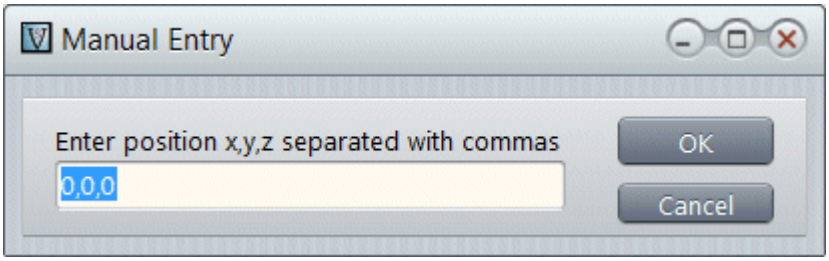
## 5 Value/Default


Enter in this field the parameter(s) of the command. Use commas to separate values of a list.
If Request is selected, the optional Default line is used as a first suggestion to the user.

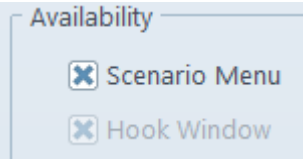
## 6 Description




If Request is selected, the Description line will be use to explain the kind of parameters to enter in the data line:
For ie, putting in the Default field: **"0,0,0"** and in the Description field: **"Enter position x,y,z separated with commas"** will give that window:



## 7 Availability




The Commands defined in the scenario can be triggered from the code (by raising the corresponding event) but most of the time, the purpose of defining a Command is to let the user select it from the GUI. vsTASKER offers two ways for using these commands:

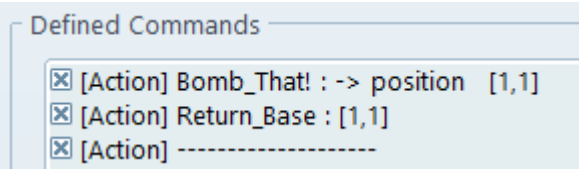
- Scenario menu: At runtime (only), right clicking on the scenario map will display the Commands menu with all defined commands listed in the same order as in the Defined list.


- Hook window: *Not available yet.*

## 8 Update/Create

Whenever an Edit field has been changed, use this button to create the command or update the selected command on the Defined list.
Failure to use this button will not create nor update the command.

## 9 List




This lists all defined commands.
Left of the column sign is the name of the associated event (spaces are replaced with underscore _)
Right summarize the input and parameter settings.
Use the check mark for deletion.

## 10 Sorting

Use this button to sort alphabetically the list of commands.

## 11 Rearrange

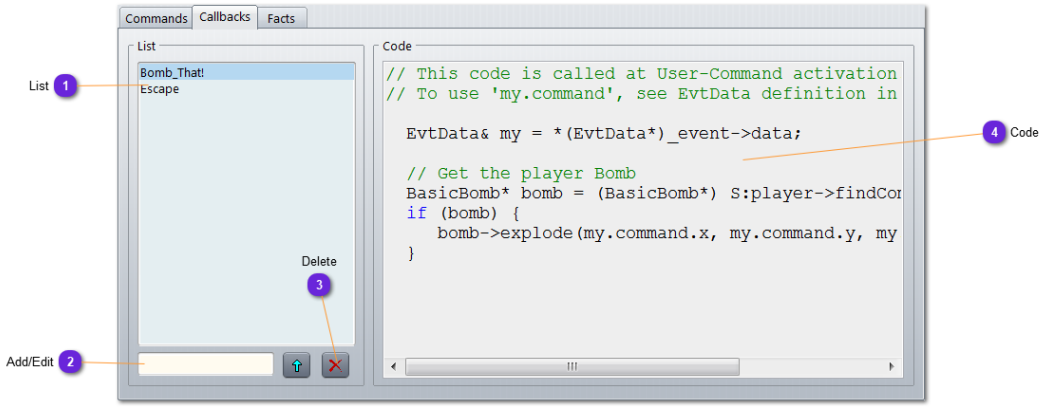
Move up or down in the list the selected command.

## 12 Delete

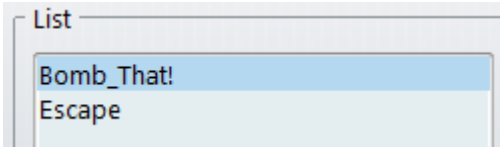
Use this button to delete all marked commands (or the selected one if none).

# Callbacks

Callbacks are the code associated with the Command (event).
The code is called after the event (command name) is propagated to the Scenario.




## 1 List




Lists all events defined at the scenario level. For each event, a C++ function code is associated.

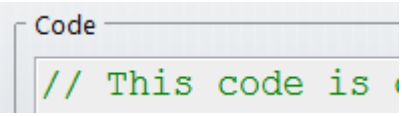
## 2 Add/Edit

To add a new event, just type its name in the text field below the list and use the  button.

## 3 Delete

To delete an event, select it and use this button.

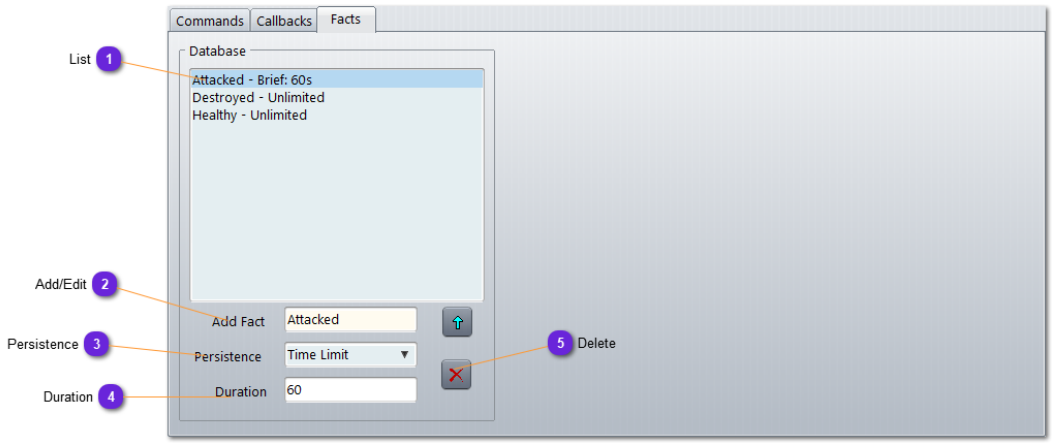
## 4 Code



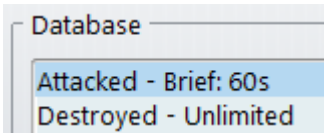

To specify the reacting code for an event, select it from the Event List and use this text area to put whatever C++ code you need.
If the event is carrying data, `_event` is not `NULL` and can be used locally, once converted into `EvtData` (see definition in runtime.h)

# Facts

Use this window to fill the Scenario Fact database at startup.

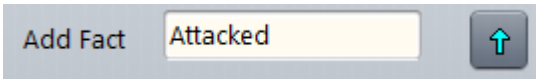



## 1 List




List of all facts defined in the current scenario.

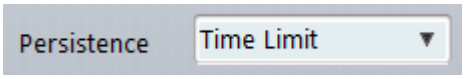
## 2 Add/Edit




Give the name of the fact to Add into the list.

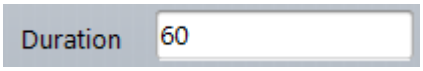
Use the  to Add a new fact or Update a selected one with a new Persistence or Duration.

## 3 Persistence




A fact can be Unlimited, meaning that it remains in the fact database until explicitly removed, or Brief, meaning it remains in the fact database only for a specified number of seconds.

## 4 Duration

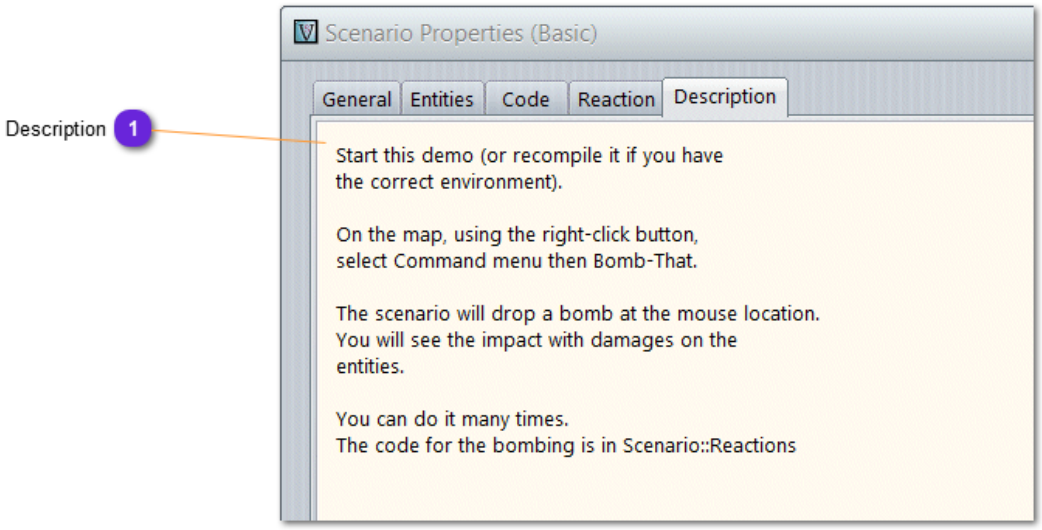



For Time Limited facts (see Persistence), specify here in seconds the total time the fact remains in the fact database before being automatically removed.

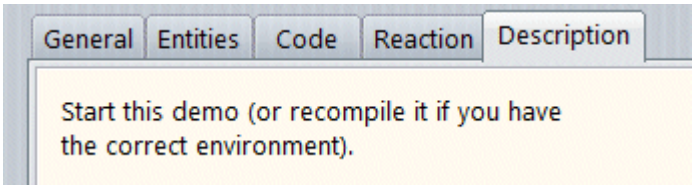
## 5 Delete

Select a fact in the list then use this button to delete it.
Cannot be undo.

# Description




## 1 Description

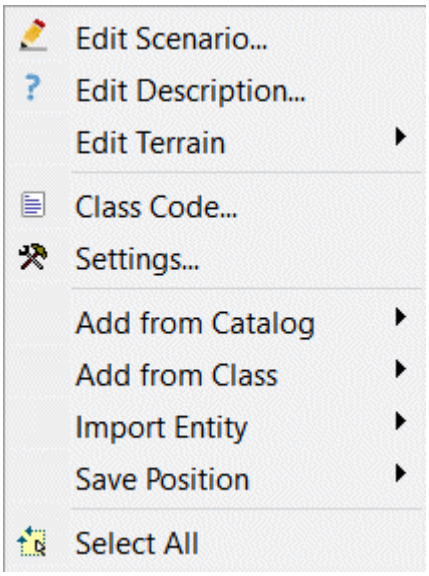



Write here the help/description of the scenario.
This text will be visible and readable in the Display panel when the scenario is selected in the Environment tree.
Description is a kind of read me.

# Popup Menus

Scenario popup menus can be displayed when using the right mouse click on Terrain map mode, over an empty area (no entity nor plan nor anything but terrain below the mouse)

## • At Design




**Edit Scenario**: call the property window

**Edit Description**: see property window, **description** button

**Edit Terrain**: see below

**Class Code**: call the Class editor window

**Settings**: call the database settings window

**Add from Catalog**: add one entity in the scenario from the Catalog list

**Add from Template**: add one entity from the list belonging to the same template

**Import Entity**: add one entity from exported entities to data/shared/entities

**Copy Position**: copy to clipboard (in ASCII format) the actual mouse position

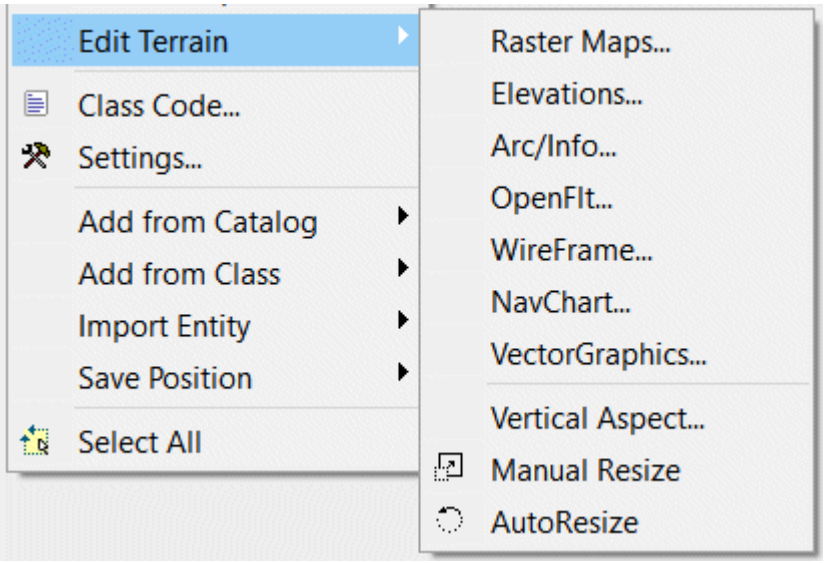
**Select All**: Select all entities

**Copy**: Copy the selected entities

**Duplicate**: Duplicate all the selected entities

**Delete**: Delete all the selected entities

### Popup Menu




**RasterMap:** see here

**Elevations**: see here

**Arc/Info**: see here

**OpenFlt**: see here

**WireFrame**: see here

**NavChart**: see here

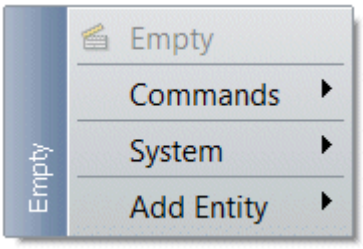
**VectorGraphics:** see here

**Vertical Aspect**: see here

**Manual Resize**: use the mouse to select the terrain area. Cursor change to + for selection (top-left, bottom-right)

**Auto Resize**: automatically reset the terrain area according to all object positions

## • At Runtime

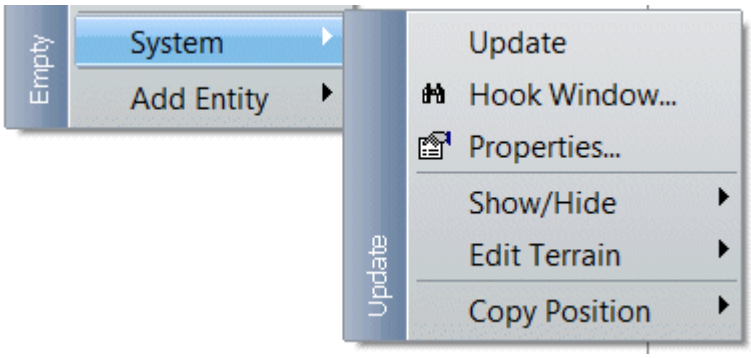



**Empty**: name of the current scenario (grayed out)

**Commands**: list all user defined commands

**System**: see below

**Add Entity**: list of the database catalog entities ready to be added




**Update**: ask the simulation engine to refresh the shared memory. If an entity is focused, to refresh the entity data.

**Hook Window**: call the scenario hook window

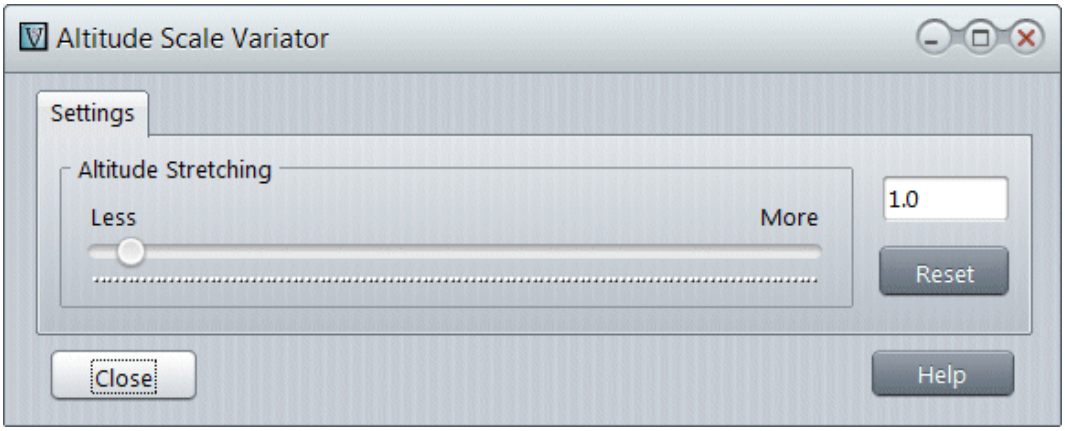
**Properties**: call the scenario design property window (in runtime, double click displays the hook window)

**Show/Hide**: display or hide entity symbols or entity names, on the map. ✓ means Show.

**Edit Terrain**: see above

**Copy Position**: see above

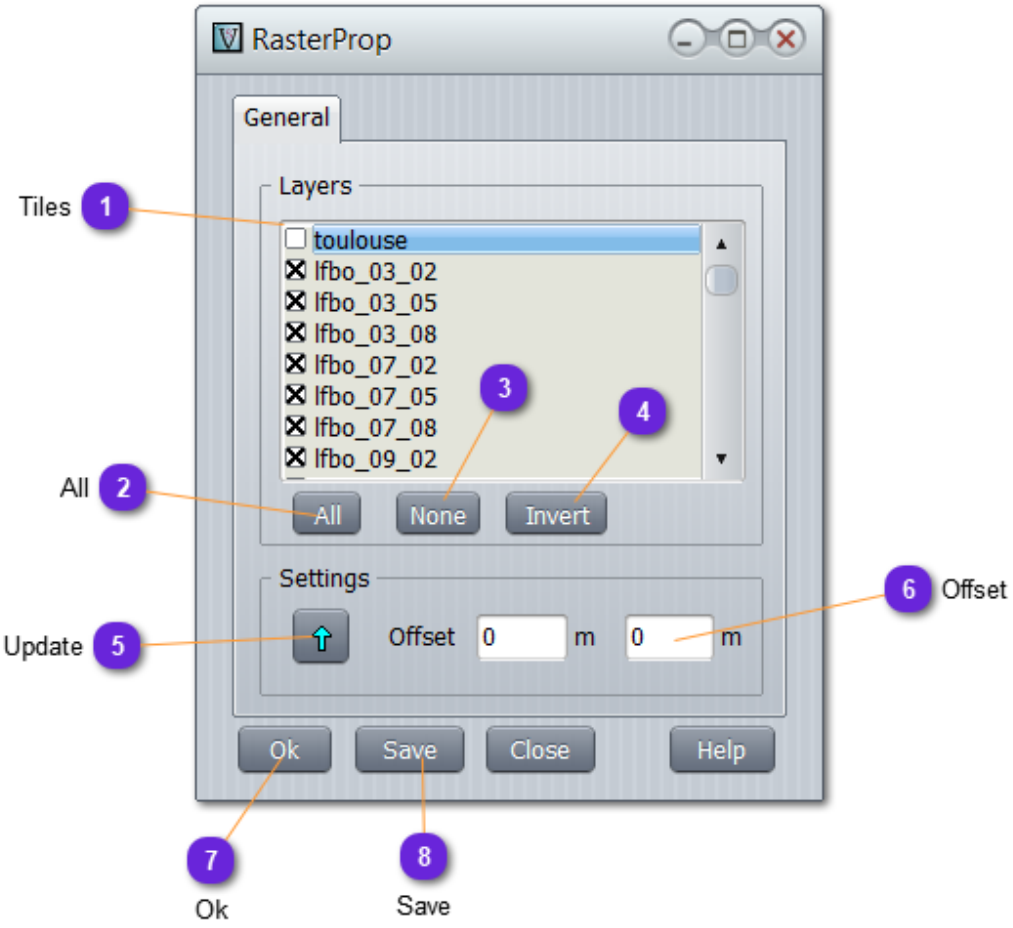
# Vertical Scaling



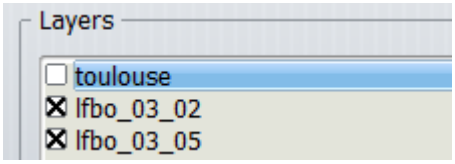

Change the following parameters to visually exaggerate the altitudes of 3D terrain:

- **Altitude Stretching**: Increase the altitude ratio regarding to the terrain size. Scaling is more uniform.
- **Reset**: revert the factor to 1

# RasterMap




## 1 Tiles

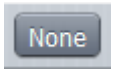



List of **all** the **tiles** belonging to the raster map file.
When a tile is checked (x), it will be drawn on the map (visible). When unchecked, it will be hidden.

## 2 All

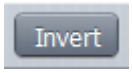

Check all tiles of the list.

## 3 None




Uncheck all tiles of the list.

## 4 Invert




Check all unchecked tiles of the list and uncheck all checked ones.

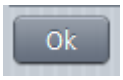
## 5 Update

**Apply** the offset (6) on **all checked** tiles.

## 6 Offset


Moves the checked tiles by (x,y) meters.

## 7 Ok




Close the window and keep the changes.

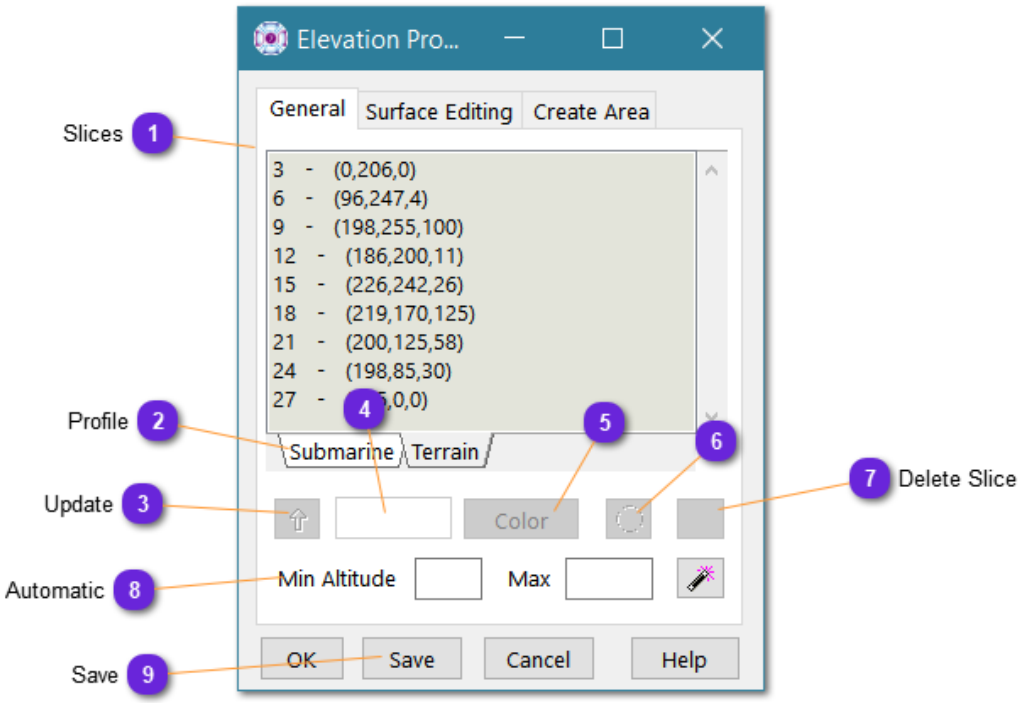

*Will not be recorded, so will be lost at database close.*

## 8 Save

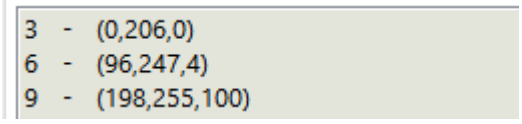

Will save the file raster map database with the changes.

# Elevations

This panel is defining the colorization of the elevation terrain, per altitude slices.

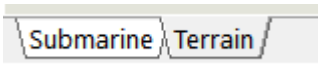



## 1 Slices




List of terrain slices with the associated color.
For i.e., in the list above, the terrain height from [0..50[ meters is colored in RGB as (0,128,0)

## 2 Profile




Two types of coloring is reserved: above see level and submarine terrain (negative values).


*Below sea level terrain cannot mix with normal coast terrain as only one color layering is covering the whole elevation database.*

## 3 Update

Use this button to **update** the **selected** slice with entered text and color values.

## 4 Low Altitude



Specify the low altitude (in meters) of the selected terrain slice (high altitude is the next slice low altitude -1)
You can change this value and the slice will be automatically updated and the list sorted on altitude.

## 5 Slice Color


Specify the color to apply to all terrain pixels belonging to the selected slice (from this slice altitude to the next slice altitude)

## 6 Add Slice

Will add a new slice at the altitude specified in the text field, with default coloring.

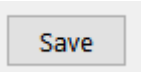
## 7 Delete Slice

Will remove the selected slice.

## 8 Automatic


Using the wizard button will automatically distribute colors and slice between the minimum and maximum altitudes defined in the text field.
Automatic coloring will depend on the profile.

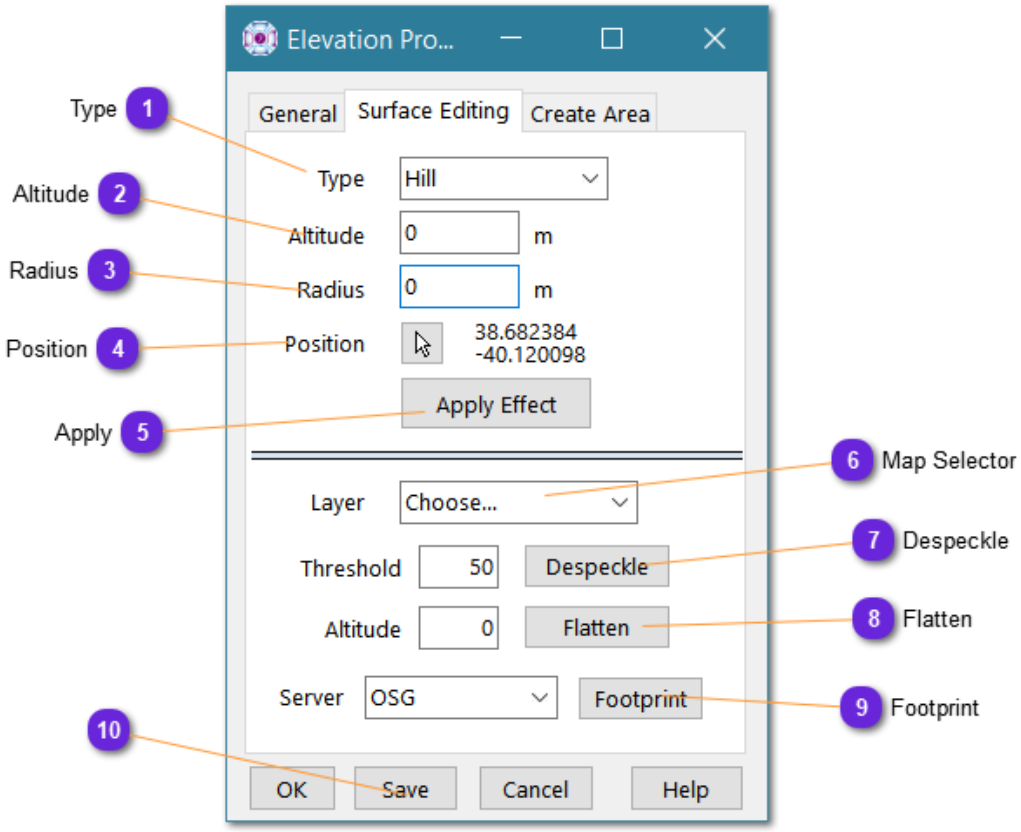
## 9 Save




**Save** (overwrite) this color setting to the elevation file itself (instead of just the database).
By doing so, the next database that will use this elevation file will get this setting.

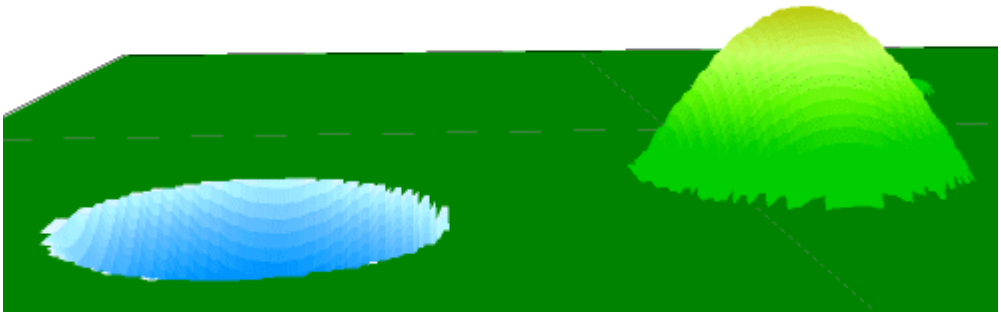
# Surface Editing

Use this panel to modify the elevation terrain profile.
The alteration added is either a smoothed **hill** or **cavity.**

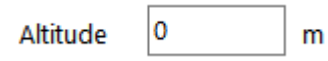



## 1 Type


Select here the type of alteration you want to add on the terrain (Hill or Cavity, see image below)

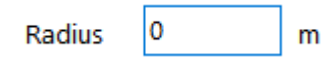


## 2 Altitude




Set here the **altitude** (amplitude) of the alteration, meaning, the highest (or lowest) point of the added hill (or cavity)

## 3 Radius

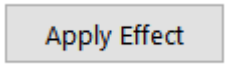



**Width** of the base of the alteration.

## 4 Position


Use the arrow button to select on the terrain map the center of the alteration that must be added on the actual terrain.
Mouse will change to a + for position selection on the map.

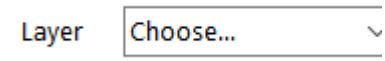
## 5 Apply




Apply the alteration (hill or cavity) on the terrain, at the specified position.

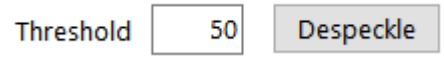

*Cannot undo !*

## 6 Map Selector



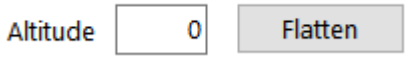

Select from the list of elevation layers the map to work with.

## 7 Despeckle




Use this option to remove artifacts that might appears after conversion or import from raw data source.
The value is the minimum burst or delta altitude between three consecutive plots (horizontal than vertical scan).
For i.e, a-b-c, three consecutive horizontal altitudes (plots). If a and c are almost identical (+/- one meter) and if |b-a| > threshold, then b will be computed as mean between a and c.

## 8 Flatten

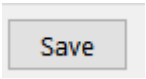



Force all altitudes in selected map to be equal to the left value.

## 9 Footprint


Will build the elevation layer using the terrain server selected on the left.
When CIGI is selected, the IG must be running and the CIGI setting must have been set prior to use this function.
See CIGI Design chapter.

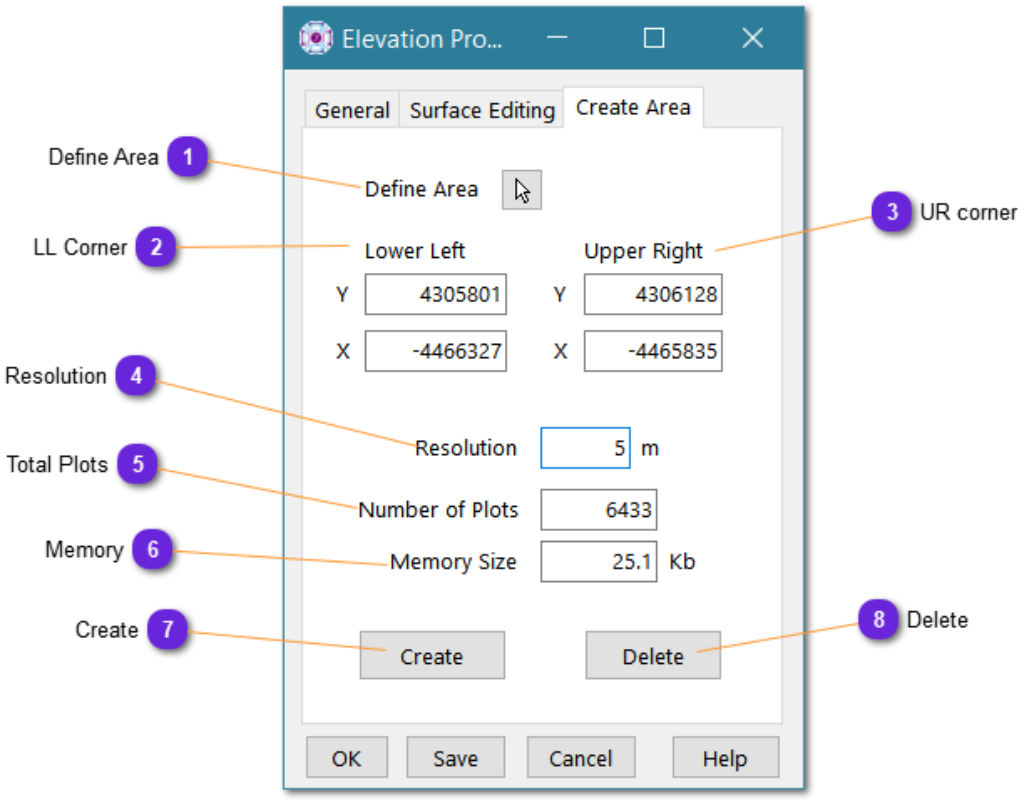
## 10 Save



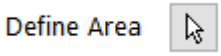

Save the altered elevation terrain file (otherwise, alterations will be lost as they are not saved with the database)

# Create Area

Use this panel to create **new map** layers for the terrain elevations.

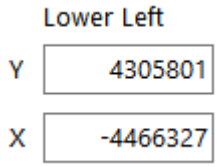



## 1 Define Area




Select the arrow button to draw on the map the layer area to create.
The definition is a rectangle from top left to bottom right.
When selected, the window disappear and the cursor changes to a double arrow, waiting for the user to draw the area. When done, the window comes back.

## 2 LL Corner

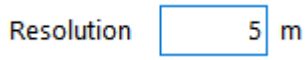



Lower left corner coordinates of the layer area. Can be modified.

## 3 UR corner

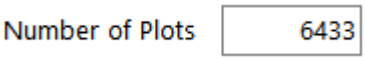

Upper right corner of the area. Can be modified.

## 4 Resolution




Specify here the width of a layer plot (square). The smaller the plot, the higher the accuracy but the heavier in memory. As several layers can be used and mixed inside an elevation terrain, it can be a good idea to create a low level area with big plots and several smaller layers with small size plots for better accuracy, on area of interest.

## 5 Total Plots

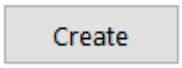



According to the size of the terrain and the resolution specified above, this is the number of plots that will be created.

## 6 Memory

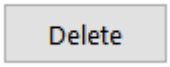

According to the number of plots computed above, this is the memory needed by the layer.

## 7 Create



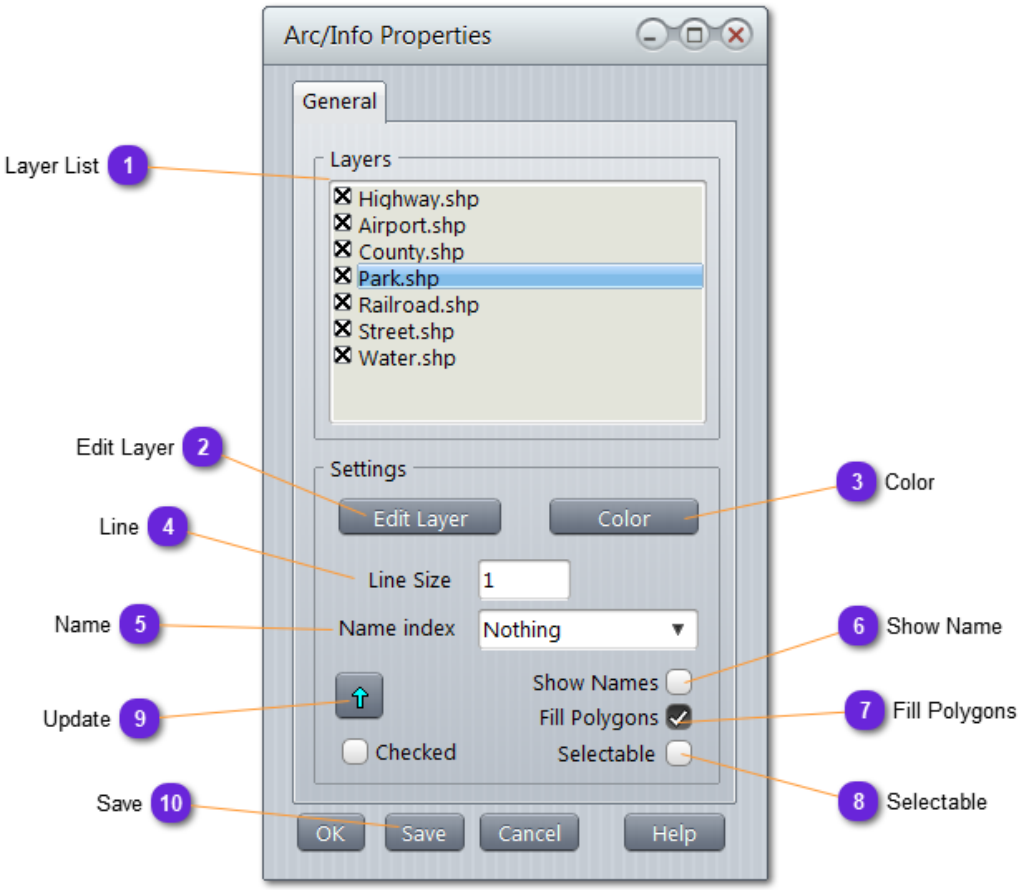

If the setting is satisfactory, use this button to create this layer and add it to the terrain elevation database.
The name used is defaulted. generated, then generated_1, generated_2, etc.

## 8 Delete

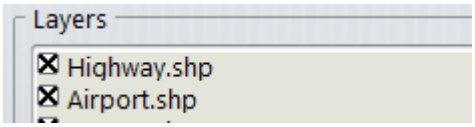



Delete (suppress) the latest generated layer (bottom of the list).

# Arc/Info

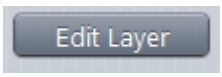



## 1 Layer List



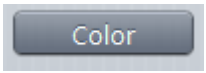

List all layers imported from the original Arc/Info file.
Select one layer in order to activate the Settings block below.
When a layer is checked (**x**), it will be drawn on the map (visible). When unchecked, it will be hidden.

## 2 Edit Layer



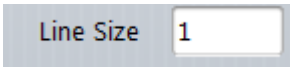

Call the layer editor. See here.

## 3 Color



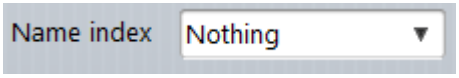

Set the color to use to draw all lines of the selected layer.

## 4 Line



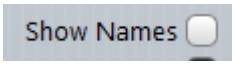

Use the text field to define the thickness of all lines of the selected layer.

## 5 Name



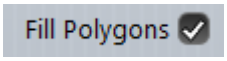

Select from the list of attributes defined in the layer, which one to use as a label to display on the map.

## 6 Show Name



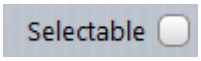

If checked, names (5) are displayed on the map.

## 7 Fill Polygons



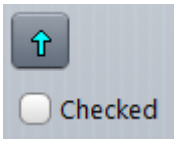

If checked, all polygons will be filled, otherwise, will be outlined.

## 8 Selectable




If checked, the layer will be selectable with the mouse.
When a layer is selected, it will be drawn with double thickness.

## 9 Update



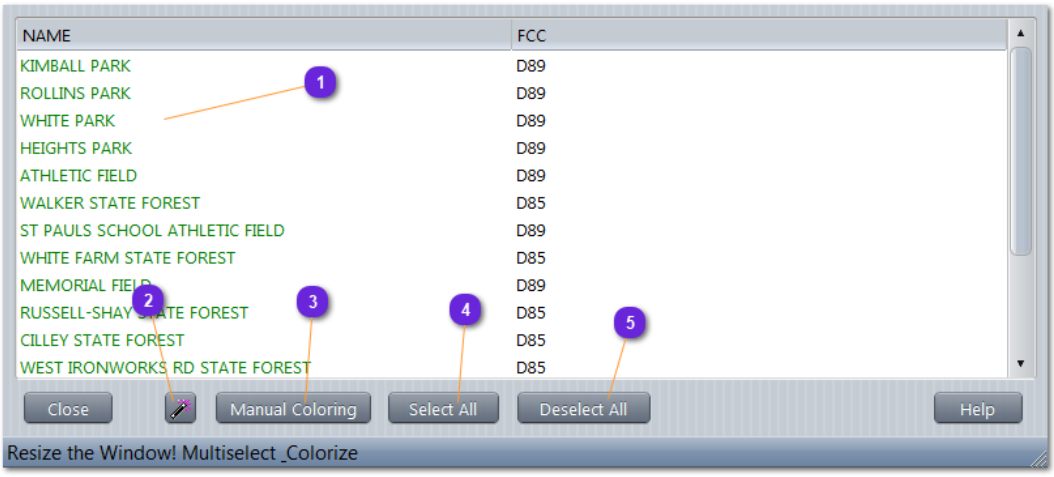

Use this button to **update** the **selected** layer with the changes applied in the settings block.
If Checked option is checked, **update** will apply for **all checked** layers of the list (1) and the selected one.

## 10 Save


Use this button to overwrite the Arc/Info file with the updated settings (otherwise, changes will be lost when database is closed)

# Editing

This window lists all the records belonging to the layer, organized by attributes.




## 1 Fields

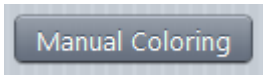
| NAME | FCC |
|---|---|
| KIMBALL PARK | D89 |
| ROLLINS PARK | D89 |
| WHITE PARK | D89 |

List of all attributes defined in the structure of the Arc/Info layer.
Each attribute is given column of the table.
Records of the layer are listed in the rows of the table.

## 2 Wizard Coloring

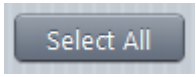
Use this button to automatically color all the records of the table.

## 3 Manual Coloring




Select one record in the table and chose the color to apply form the dialog box palette.

## 4 Select All

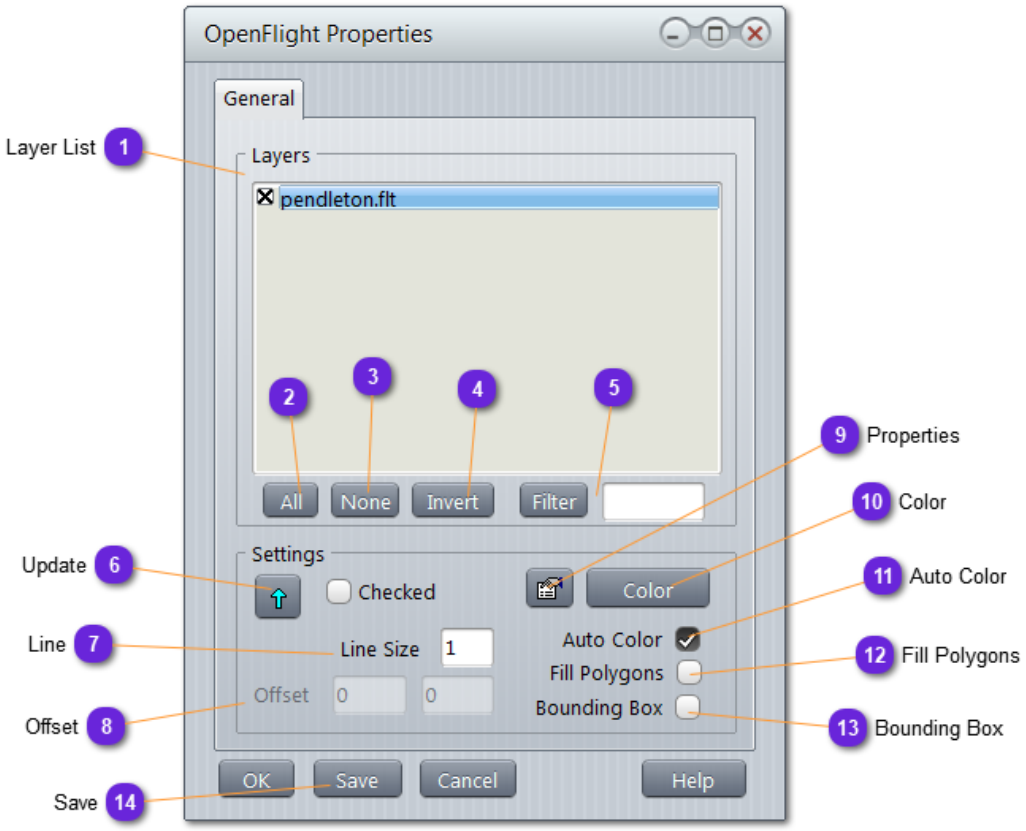



Select all records of the table (for manual coloring).

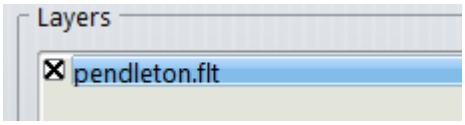
## 5 Deselect All


Deselect all records of the table.

# OpenFlight




## 1 Layer List

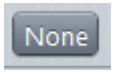



List of all the records belonging to the database file.
When a record is checked (x), it will be drawn on the map (visible). When unchecked, it will be hidden.

## 2 All

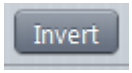

Check all records of the list.

## 3 None




Uncheck all records of the list.

## 4 Invert

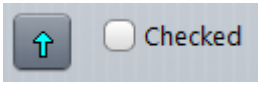



Check all unchecked records of the list and uncheck all checked ones.

## 5 Filter


Select only records whose name contains the motif written in the text field. Deselect all others.

## 6 Update




Use this button to **update** the **selected** record with the changed values in Settings.
If Checked option is checked, **update** will apply for **all checked** records of the list (1) and the selected one.

## 7 Line


Defines the thickness of the line used to display the polygons on the map.

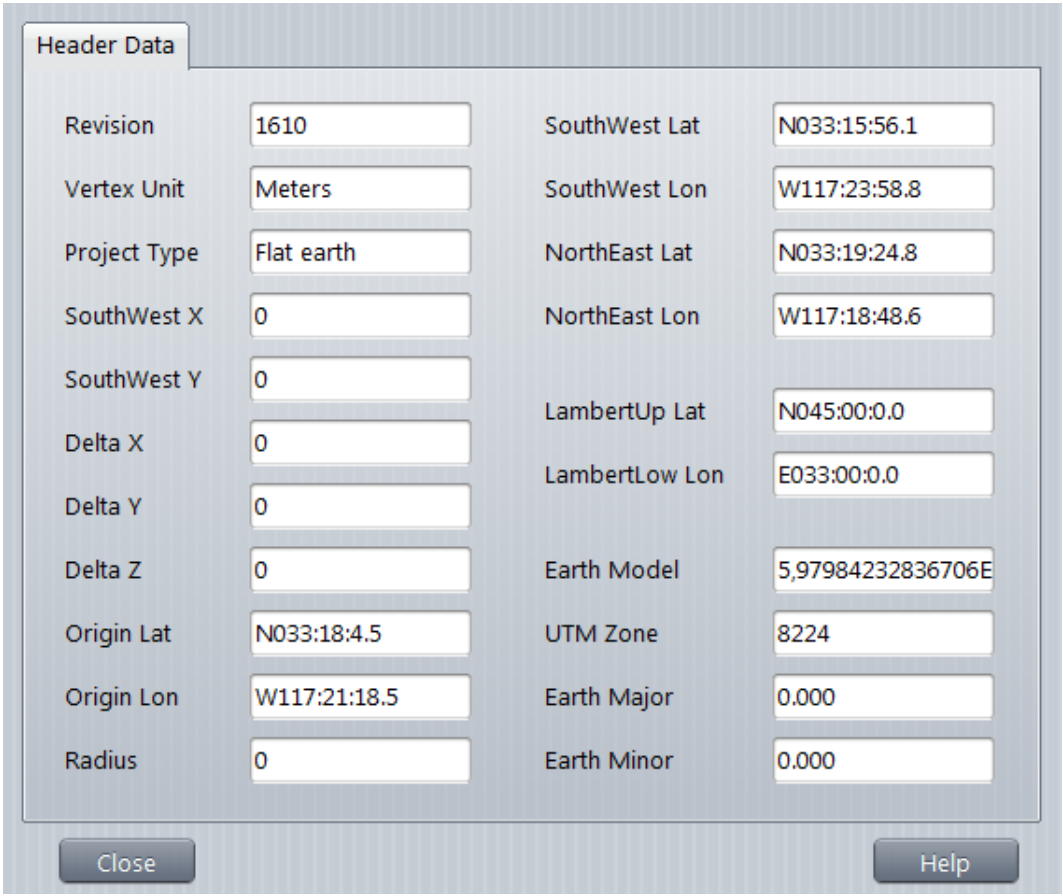
## 8 Offset


Moves the selected records by (x,y) meters.

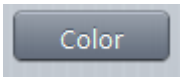
*The layer must have been marked as Selectable in the open-flight property window.*

## 9 Properties

Display the properties of the record, as extracted from the original database. See below:



## 10 Color

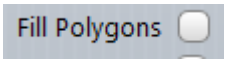



Set the color of all polygons and lines of the record.

## 11 Auto Color


When this is checked, the polygons edges are colored according to their altitude. The redder, the higher.

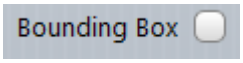
## 12 Fill Polygons




When checked, all polygons are drawn filled.


*This option is not very useful unless the color is set to very light gray and (11) is checked.*

## 13 Bounding Box



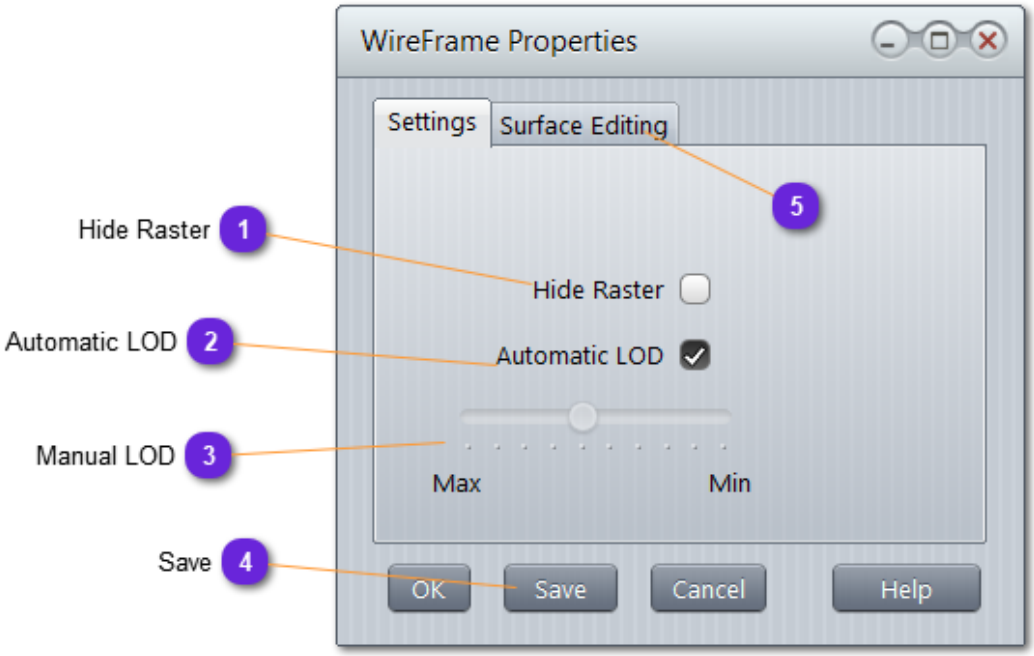

When checked, the layer is marked with a bounding box.


*This can be useful for records that are not displayed in detail but need to be located on the map.*

## 14 Save


Overwrite the database file with all the changes in the record settings.

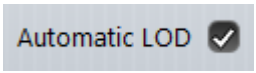
# WireFrame




## 1 Hide Raster


When checked, the raster skin will be hidden and only the skeleton mesh will be drawn.

## 2 Automatic LOD

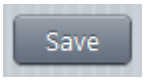



When checked, the accuracy of the mesh will change according to the zoom level.

## 3 Manual LOD

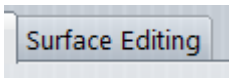

When enabled (Automatic LOD unchecked), use the slider to increase the Level of Detail to Maximum accuracy or Minimum accuracy.

## 4 Save



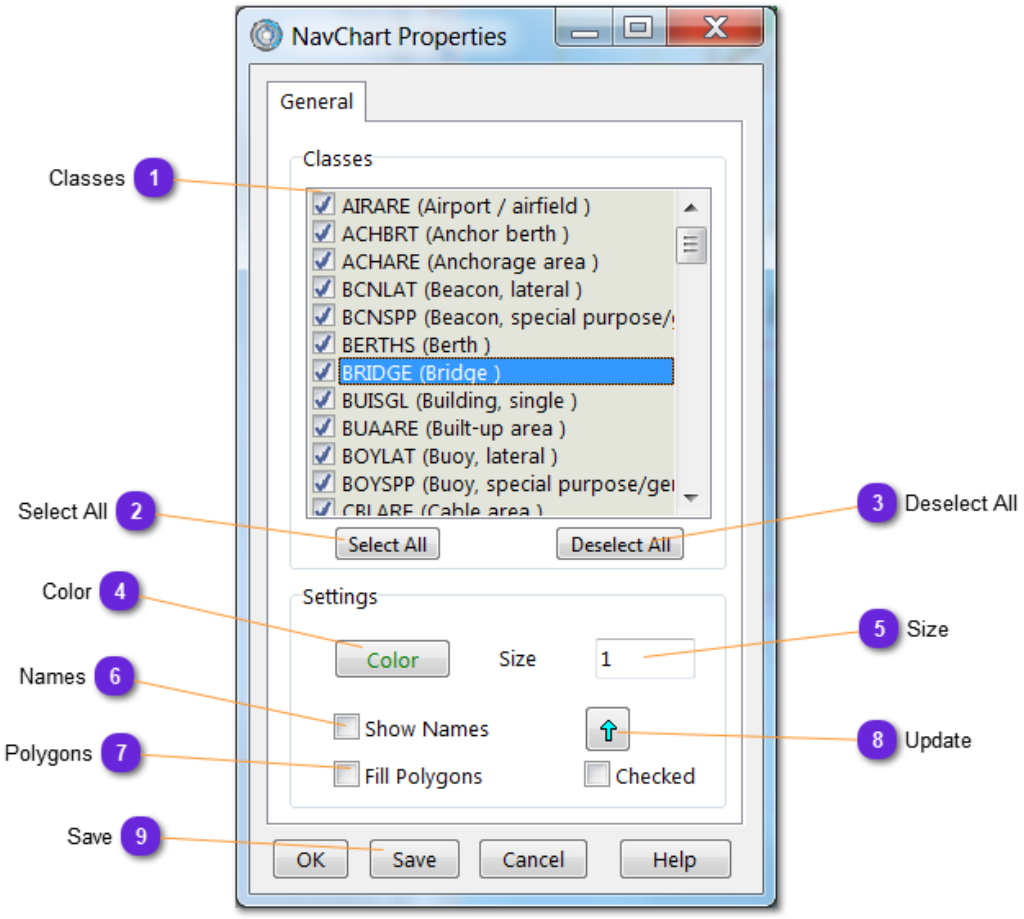

Overwrite the database file with all the changes in the record settings.

## 5 Surface Editing

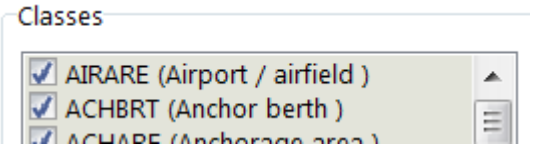



See here.

# NavChart

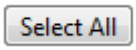



## 1 Classes



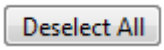

List of all object classes belonging to the database file.
When checked, the class will be hidden and not drawn on the map.

## 2 Select All




Use this button to select all the classes above. By doing that, all classes will all be visible and drawn on the map. Also, the settings will propagate to all (see **Update**)

## 3 Deselect All

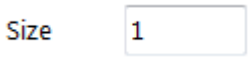



Use this button to deselect all the classes above. By doing that, all classes will all be removed from the drawing. Do that before selecting one by one the classes and see their appearance on the map.

## 4 Color

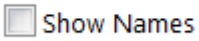

Select the color for drawing the objects of the selected class.

## 5 Size



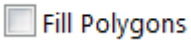

Size of the selected object class when drawn on the map (text, symbol or line thickness).

## 6 Names



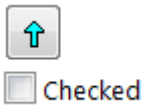

If check, the name of each object of the selected class will be drawn.

## 7 Polygons



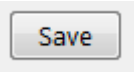

If check, all object polygons for the selected class will be drawn filled.

## 8 Update



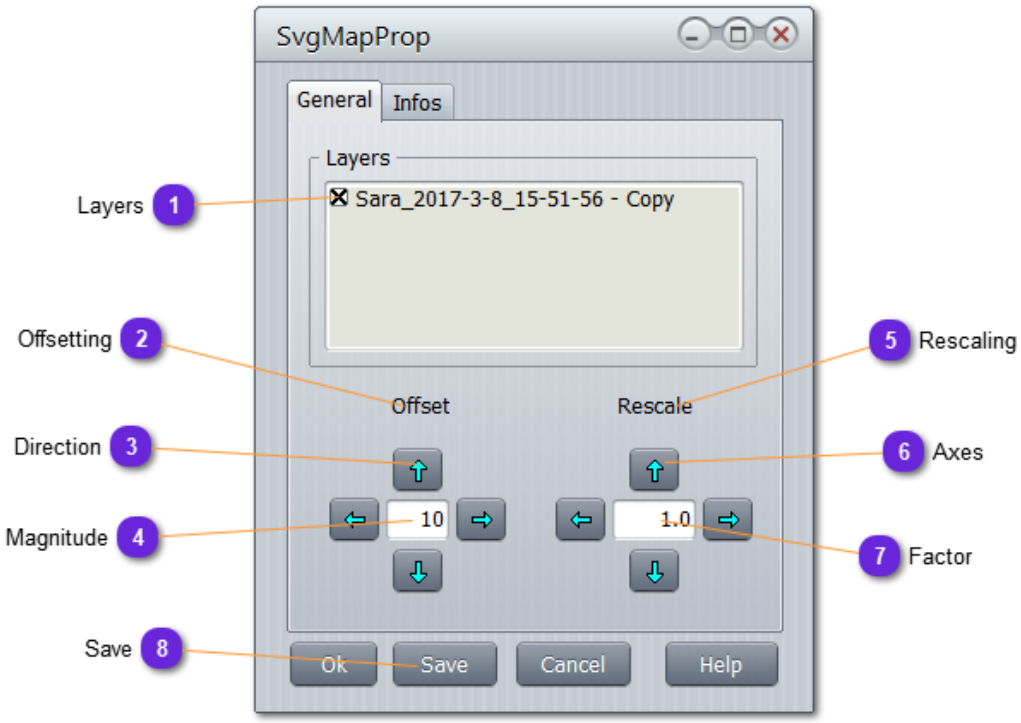

Us this button to **update** the **selected** class with all changes of the Settings block.
If Checked option is checked, **update** will apply for **all checked** classes of the list (1) in addition of the selected one.

## 9 Save

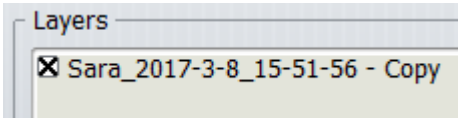



Overwrite the database file with all the changes in the class settings.

# SVG Map




## 1 Layers




List of all the layers belonging to the SVG database.
Only the ticked layers will be concerned with the offset and rescale functions.

## 2 Offsetting


Will shift the layer on the terrain by moving both lower-left and upper-right corners according to the offset value (4).

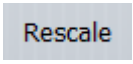
## 3 Direction

Use Up, Right, Down and Left buttons to offset the ticked layers.

## 4 Magnitude


Set here the value in meters for each offset step (each button click).

## 5 Rescaling

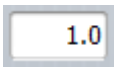



Will resize the ticked layers by moving the upper-right corner position according to the scale value (7).

## 6 Axes

Use the Up, Right, Bottom and Left buttons to rescale the ticked layers on the selected axis.

## 7 Factor

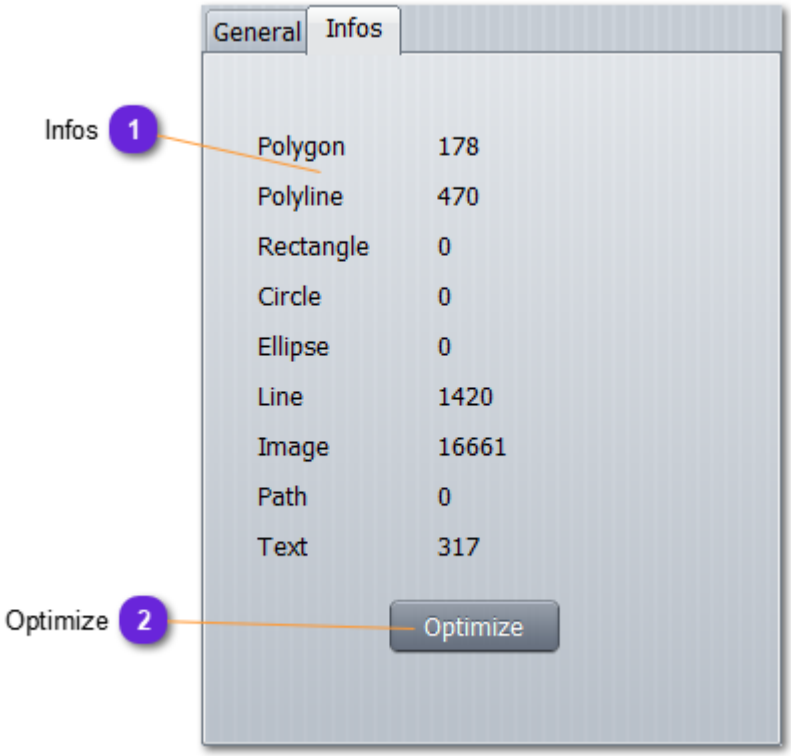



Set here the value of the rescaling applied on each button click. For ie, if the value is 2, pressing the Right button will double the actual width of all ticked layers.
Lower-left corner of each layer will remain unchanged.

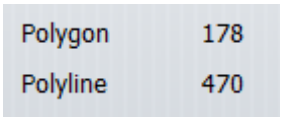
## 8 Save


Will overwrite the SVG database with the modified layers. Without saving, all the changes will be lost when the database is closed.

# Layer Info

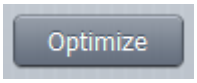



## 1 Infos




Count the number of each shape types for the selected layer (in General pane).

## 2 Optimize




Use this button to remove all invisible shapes that might exist in the selected layer.
Invisible shapes are too small rectangles, circles or ellipse, image without file attached, 0 width stroke for transparent polyline, polygon or path, etc.


*The optimized SVG Map will only be for the current session. The file must be saved on disk for the change to be permanent.*

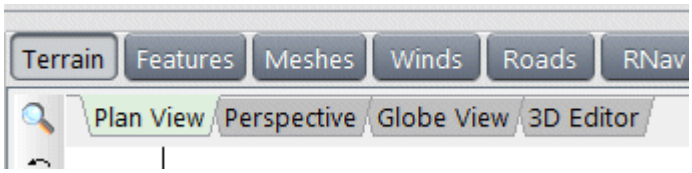
# Terrain View

Terrain layers (see here) are displayed on the Map panel, for each selected (activated) scenario.
Drawings are using OpenGL and plugins.

Mouse must be used to select any entity or plan point drawn on the map.
Double click on the map background will show the scenario property window.

Depress right mouse button (and hold) while moving the mouse to pan the terrain.
Use the mouse wheel to zoom in and out.
Depress left mouse button (and hold) while moving the mouse to select on the terrain.

Plan View is the default God eye view.




## • Terrain ToolBar

**Zoom** on the map (cursor changes). Click and select the area you want to zoom at.

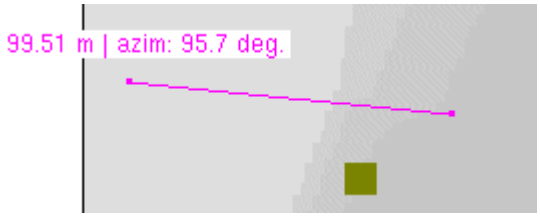
**Revert** zoom (goes back to optimal terrain view)

Allow **panning** of the terrain without using the right mouse button

Display/Hide the vertical view window

Activate/Deactivate the **measurement string** on the map.
Click on the tool, cursor changes to symbol +. Click on the map and (with mouse button down) move the mouse to define the measurement string.
Distance and azimuth are displayed on the map (see below).
When tool is activated (mouse symbol +), string extremities can be reselected and moved.
Extremities can also be dropped on entities and automatically hooked (this allow measurement between to entities, even at runtime).

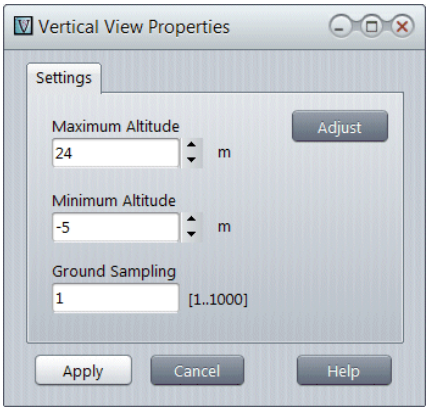
To remove the string, just click on the map and release.
Click again on the tool to deactivate it.



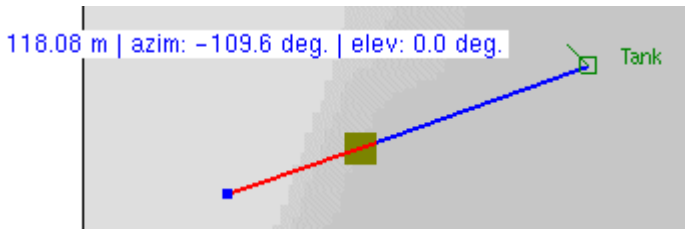

To activate this functionality on the OpenGL or GdiMap display, use **CTRL I** key or the following code:

```
ogl_draw->mouse.mmmode = M_String;    // activate
ogl_draw->mouse.mmmode = M_Void;  // deactivate (or ESC)
```

Activate/Deactivate the **line of sight** (LoS) string on the map.
Click on the tool, cursor changes to symbol +. Click on the map and (with mouse button down) move the mouse to define the LoS string.
The extremity of the string are dropped on the ground below it. The color change (blue/red) at the point where the terrain is blocking the view.
Extremities can also be dropped on entities and automatically hooked (this allow visibility between to entities, even at runtime).




Vertical view is also displayed with the associated setting window.
You can change the Ground Sampling parameter to increase/decrease the accuracy of the visibility check.

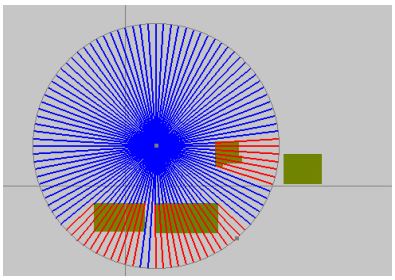



To remove the string, just click on the map and release.
Click again on the tool to deactivate it.

To activate this functionality on the OpenGL or GdiMap display, use **CTRL L** key or the following code:

```
ogl_draw->mouse.mmmode = M_LOS;    // activate
ogl_draw->mouse.mmmode = M_Void;  // deactivate (or ESC)
```

Activate/Deactivate the **area of sight** (AoS) fan on the map.



Click on the tool, cursor changes to symbol +. Click on the map to define the fan center and (with mouse button down) move the mouse to define the fan size. The center of the fan is dropped on the ground below it. For each sector, color change (blue/red) at the point where the terrain is blocking the view.
Fan center can also be dropped on entity and automatically hooked (this allow ground coverage by the entity, even at runtime).

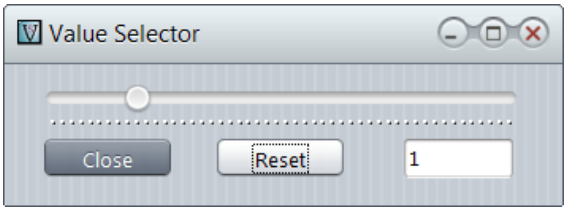
To remove the fan, just click on the map and release.
Click again on the tool to deactivate it.

Activate/Deactivate the **tracking** of the selected entity. The map will automatically centered on the map (useful at runtime).

To activate this functionality on the OpenGL or GdiMap display, use **CTRL O** key or the following code:

```
ogl_draw->mouse.mmmode = M_AOS;    // activate
ogl_draw->mouse.mmmode = M_Void;   // deactivate (or ESC)
```

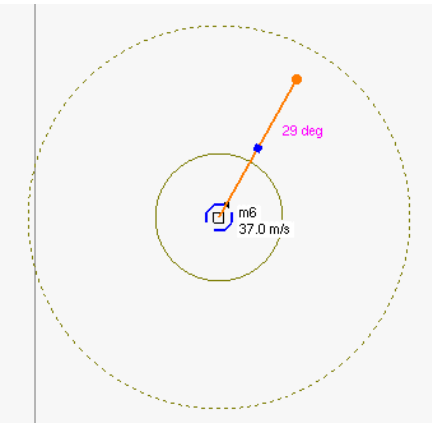
Use to change the **scale** of all entity symbols symbols




Use the slider to size up (right) or down (left) symbols on the map.
Reset will revert the factor to 1 (default).

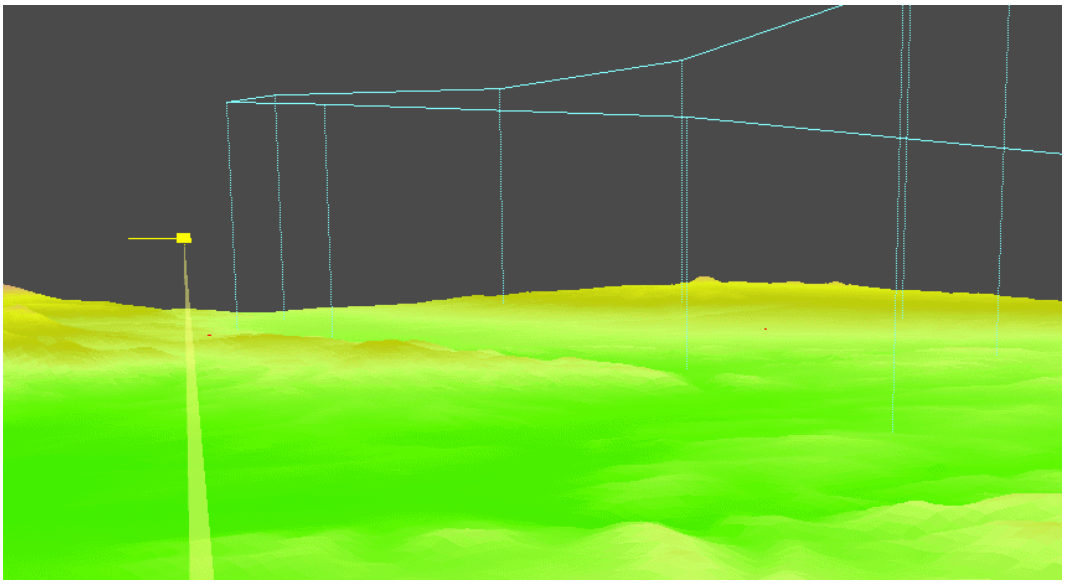
Same as entity popup menu option **basic setup**.
Displays the azimuth line (orange vector) and the speed selector (blue anchor).
Grab the orange anchor with the mouse and turn it to set the heading.
Grab the blue anchor with the mouse and slide it to change the speed.
Plain and dashed circles represent minimum and maximum speed for the entity.
Click anywhere to quit the mode.

## Terrain View



# Perspective

Use this mode to display the terrain map in 3D perspective view.
Entity selection and edition will not be possible in this mode, only viewing.



## • Mouse Control

**Left** mouse **button** down to **rotate** the **camera** around the center of the visible terrain

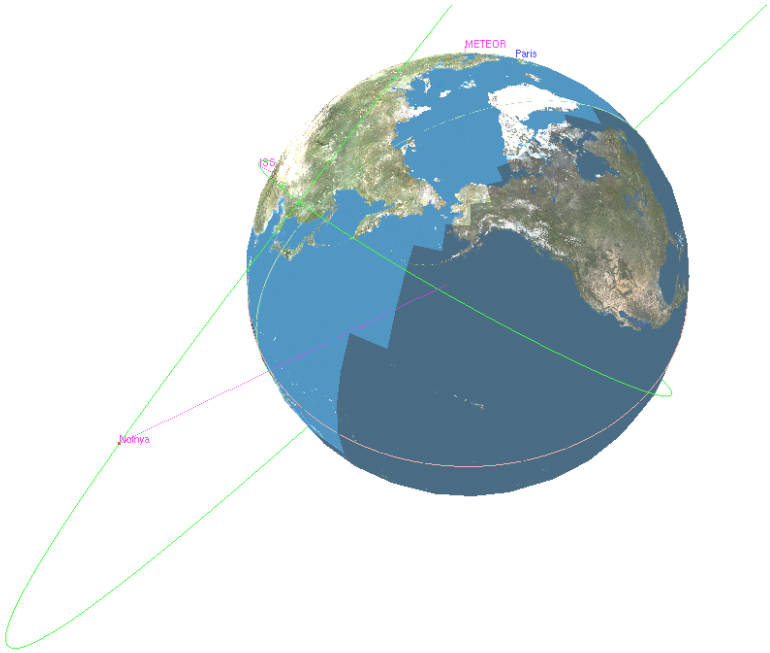
Mouse **wheel** to **zoom** in and out

**Right** mouse **button** down to **move** the **camera** on the horizontal plane.


*In perspective view, mouse selection is not available. Most of the 2D drawings are represented in 3D.*

# Globe

This mode is used for **satellite view** as it displays the **earth globe** with orbits.
vsTASKER needs to create the globe at first usage.
See the actual Earth Globe settings here.



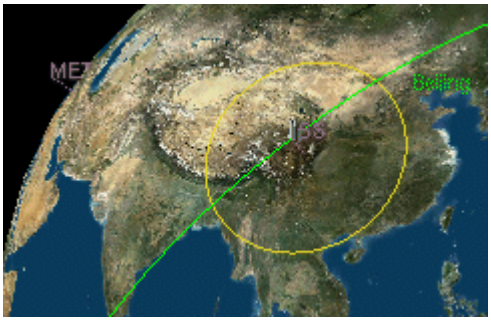

## • Camera control

Use the **Left** mouse **button** to **rotate** the **camera** in all directions, earth centered.

**Right** mouse **button** is not use for now (earth will automatically re-center after the offset)

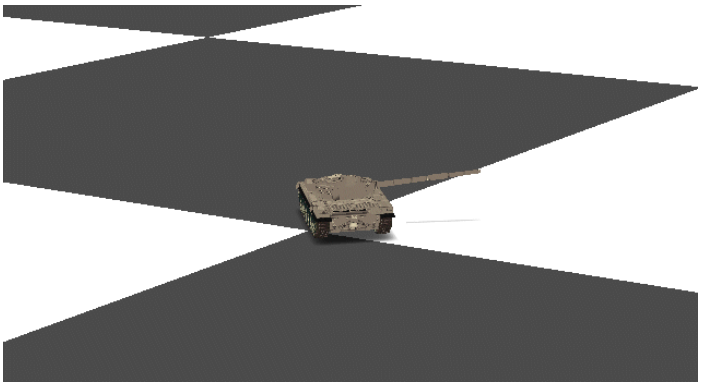
Use the mouse **wheel** to **zoom** in and out.

When a satellite is selected on the environment list, use the  button to **track** it.
**Right** mouse **button** will then **rotate** the **camera** in all directions, satellite centered.
Use the mouse **wheel** to **zoom** in and out the **satellite** 3D model.
If the satellite holds a sensor (like IsoElevation component or any other Space sensor whose drawing is handled by a **validation** function), it will be displayed on the earth in yellow (see below).



# Editor 3D

This mode is using an external plugin (either OpenSceneGraph or Delta3D).
By default, vsTASKER provides an OSG plugin (see Other here).
The DLL plugin must be loaded before activating this mode.



The plugin will try to represent some of the terrain layers (but not all). This editor is mainly used when an OSG database is used for the scenario. It is then mandatory to specify the name of the terrain file in the Terrain::Settings::3D Editor (see here).

Entity 3D model will have to be set in the Aspect 3D GUI panel of the property window (see here), otherwise, an orange cube will be used to represent the entity.

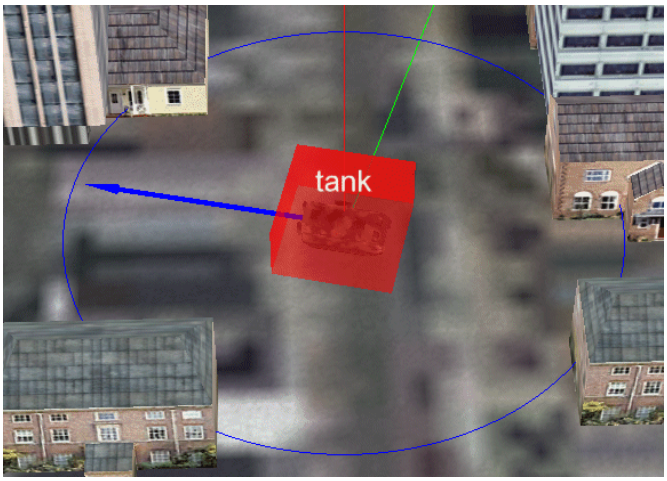
## • Camera control

**Left** mouse **button** down to **rotate** the **camera** around the center of the visible terrain
Mouse **wheel** to **zoom** in and out
**Right** mouse **button** down to **move** the **camera** on the horizontal plane.

## • Entity control

Select the entity from the environment or by clicking on the entity with keyboard **Alt** key depressed.
To **move the entity** freely on the map (if clamped, will be grounded automatically), use **Shift** key while depressing the **Left** mouse **button**.
To **rotate the entity**, use **Shift** key + **Left** mouse **button** on the blue arrow, then turn it around the circle.

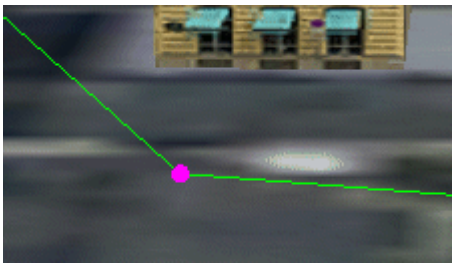
To **automatically zoom** on a selected entity (from the environment list), use the  button.




*The camera can be controlled without deselecting the entity. Popup menu can also be used.*

## • Plan edition

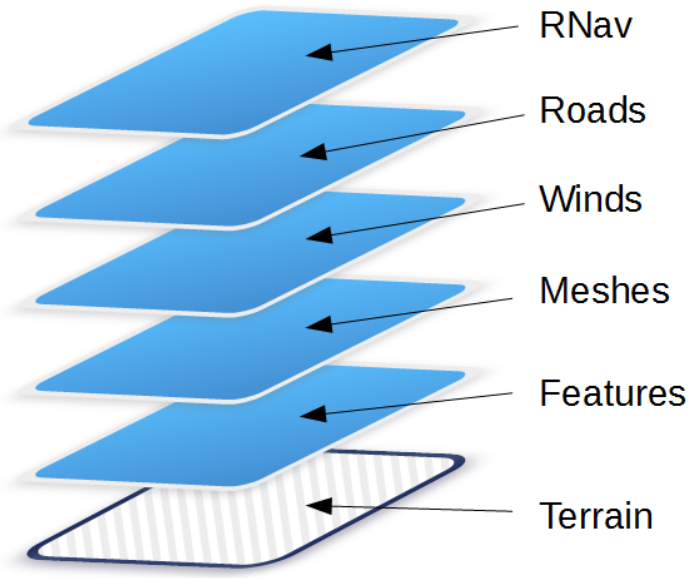
If the entity holds a plan, you can select each plan point using the **Alt** key + **Left** mouse **button**.
The plan point will turn magenta.
Use then the **Shift key** + **Left** mouse **button** to **relocate** it on the map.



*Popup menu can also be used.*

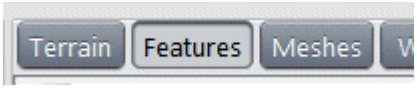
# Features

Features are specific capabilities added to the scenario. Each one have a dedicated layer, above the terrain map.
When a layer is focused, all other contents are not accessible, even if visible (reason why an entity cannot be selected if the layer Terrain is not focused).
All features have a internal data representation that can be used by any piece of code. They have also a visual representation that use the OpenGL layer for display.



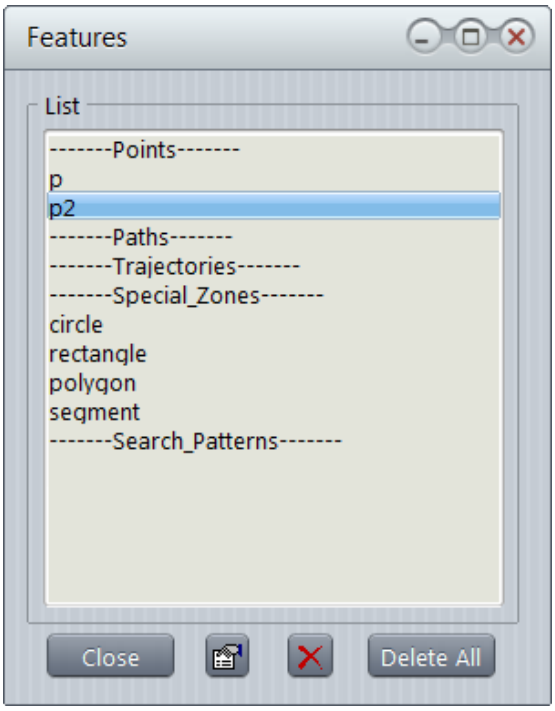

Feature can be created at design time and runtime, from the GUI or directly from the code.
Runtime features will be deleted at simulation stop.

To **access** this layer, depress the following Features button:




Once the layer is activated, only feature objects can be selected.

**Double click** the background to list all defined features.




## • Directory

Features are defined in /Feature directory.

## • Toolbar

Add Point

Add a Trajectory

Add a Path

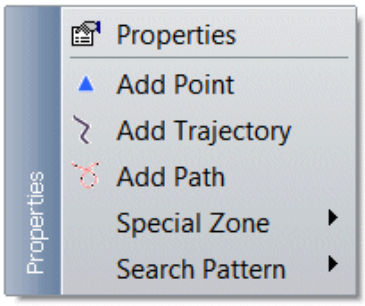
Add a Special Zone. Will open the factory window and the shape will have to be selected there.

Add a Search Pattern, default type is U-Search.

## • Popup Menu

### Features

**Right click** the background to add them:




**Properties**: see below

**Add Point**: add a new point

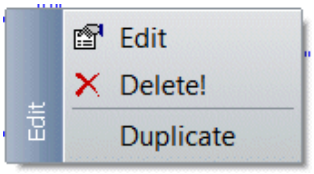
**Add Trajectory**: add a new trajectory

**Add Path**: add a new path

**Special Zone**: add a predefined special zone

**Search Pattern**: add a predefined search pattern

**Right click** any selected feature to display the popup menu:



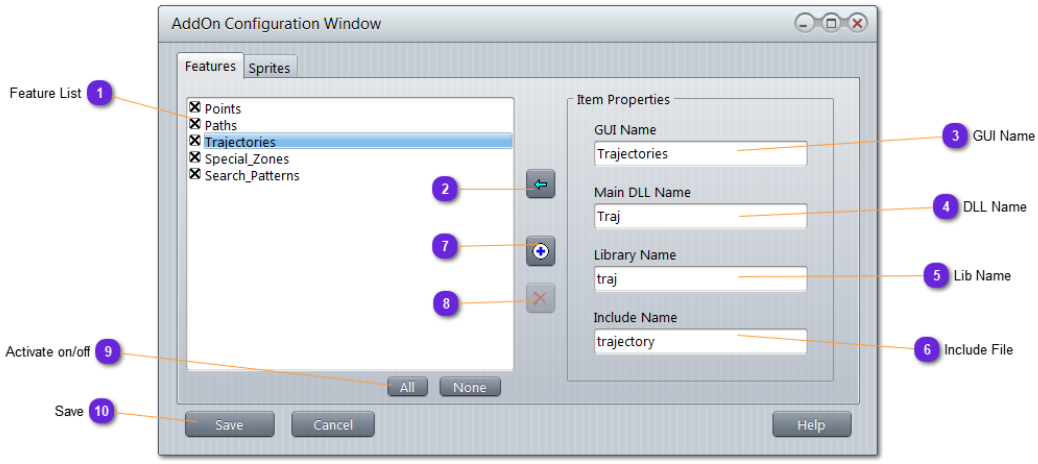

**Edit**: display the property window of the selected feature

**Delete**: remove the feature

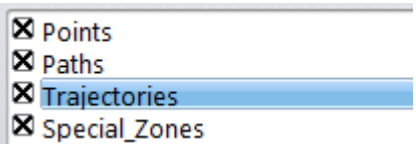
**Duplicate**: copy and paste (aside) the selected feature.

# Settings

Features are defined outside vsTASKER using specific DLL.
They are structured into /Feature directory. They can be included or excluded from the product environment.




## 1 Feature List




Uncheck a Feature to keep it in the file but forbid it to be loaded into vsTASKER.
To reduce the size of the vsTASKER memory footprint, uncheck the feature you will not use in your databases.

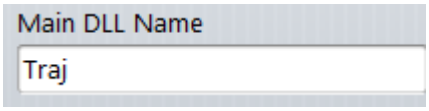
## 2 Update

Use this button to update the selected Feature (from the left list) with the data from the Item Properties block.

## 3 GUI Name

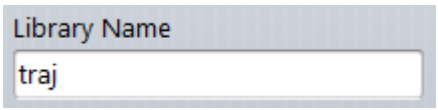

Specify here the **name** to be used in the menus, for the selected Feature entry.

## 4 DLL Name



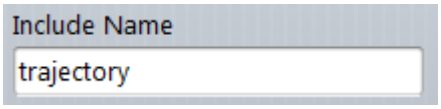

Name of the DLL in the /Features directory (needless to add the file extension).
The GUI will load the <name>M.dll interface that handle all windows while the <name>.dll (compiled with VisualStudio) will be handling the data objects. The <name>X.dll will be used for drawing the feature onto the OpenGL device.

## 5 Lib Name




Name of the library that will be produced into /lib/vcxx.
This lib will automatically be added into the *makefile* produced by vsTASKER to compile a simulation engine produced with a database using the corresponding Feature.

## 6 Include File




Name of the include file in /Feature/Include

## 7 Add

Add a new Feature (entry) into the list.

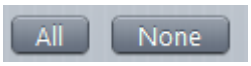
*This capability is only reserved for advanced users who are expanding the GUI using their own CodeGear C++ Builder IDE.*

## 8 Delete

Suppress the selected user added Feature.

*This capability is only reserved for advanced users who are expanding the GUI using their own CodeGear C++ Builder IDE.*

## 9 Activate on/off




Use these buttons to tick/activate **all** or **none** of the Feature Manager.
When a Manager is not ticked, it will not be available next time vsTASKER will be started. That means the associated DLL will not be loaded and the corresponding button in the toolbar will be disabled. When a database is loaded, if a Manager is disabled, all corresponding Features of the database will not be loaded. This can be useful when working with databases that do not have Features. Starting vsTASKER will be faster as Feature DLL will not be loaded. Memory footprint will also be reduced.
Do not forget to activate some or all Managers if you intend to use Features.

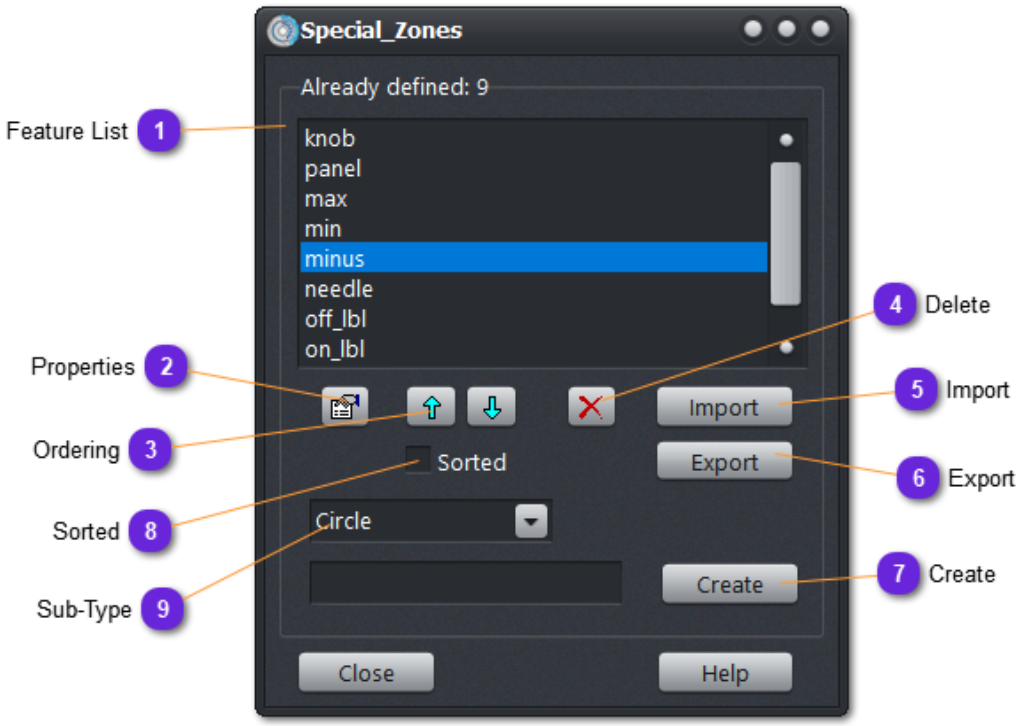
## 10 Save


Fix the configuration on a file so that next time vsTASKER is loaded, only the selected Feature DLLs will be loaded.

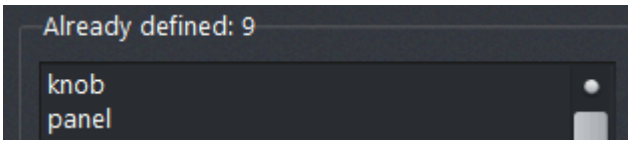
*This capability is only reserved for advanced users who are expanding the GUI using their own CodeGear C++ Builder IDE.*

# Factory

This window is used to create (or modify) any feature of a given type.




## 1 Feature List




List of all the defined features in the scenario.
The window title mention the type of feature processed by this window.

## 2 Properties

Display the **property** window of the selected feature (above).

## 3 Sorting

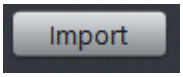
**Move** up or down the selected feature in the list.

*All features of the same type are drawn from top to bottom. If a feature is hidden behind another one, put it after the hiding one.*

## 4 Delete

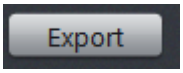
**Suppress** the selected features (one or many).

## 5 Import



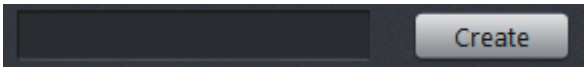

**Load** a previously exported feature from the /shared directory.

## 6 Export



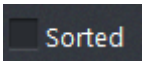

**Save** the selected feature into the /shared directory.

## 7 Create



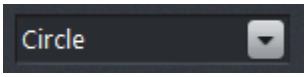

Type here the name of the feature you want to create and press the button. If the name is already taken, vsTASKER will automatically make it unique (by adding a number).

## 8 Sorted




When checked, the features are sorted alphabetically. They cannot be reordered manually. The status is recorded.

## 9 Sub-Type




For some Features, this combo-box allows specifying the sub-type to be created.
**Special Zones**:

- **Circle**
- **Rectangle**
- **Polygon**
- **Segment**

**Search-Patterns**:

- **Creeping U**
- **Creeping L**
- **Holding**
- **8 Shape**
- **Circular**
- **Spiral**
- **Sectors**

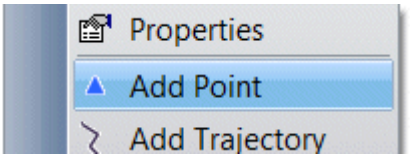
# Point Feature

A Point feature is a named mark and position on the ground. Some specific and predefined user data can be attached to the point.
The point can be used by component or logic either to direct an entity towards a specific location or compute the distance between
one entity to a point or between several points.

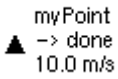
Points can also mark some sites (instead of using an entity) like an airport, a radar, a mine, a hole, etc.
It will be up to the components or logic to manipulate the point definitions.


*Built-in component which uses Point feature: MotionGoto.*

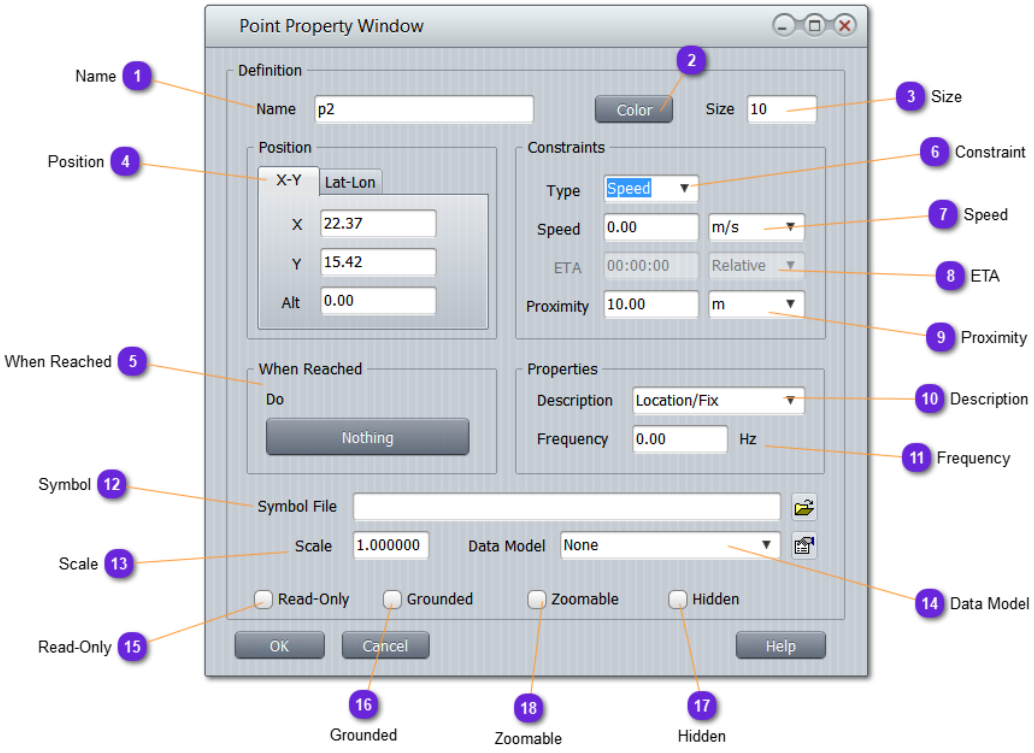
## • Popup Menu




*Example of a point (myPoint):*



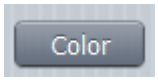
# Properties




## 1 Name


Unique **name** of the point.

## 2 Color




**Color** used to draw the default symbol on the map.

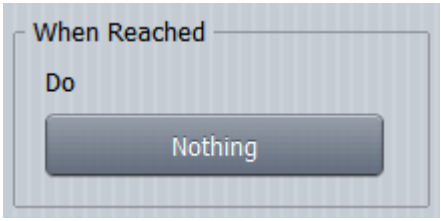
## 3 Size


Size of the default symbol (in pixels).

## 4 Position

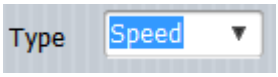

**Position** in coordinates XYZ or latitude, longitude, altitude of the point on the map.
Normally, these values are coming from the mouse positioning at creation time.
Can be updated here manually.

## 5 When Reached




This **event** setting is only informative and used by built-in component MotionGoto.
Use can set here any event and trigger it from any user code.
See event setting here.

## 6 Constraint




Specify here the type of **constraint** that must use the component MotionGoto for any entity having to direct towards the point:

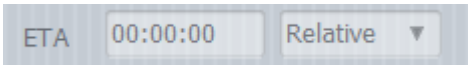
- None: nothing selected
- Speed: speed constraint. See (6)
- ETA: time constraint. See (7)

## 7 Speed


If speed constraint is selected, specify here (in the desired unit), the maximum speed the entity must reach to direct towards this point.

*MotionGoto uses this constraint but any user component or logic can do the same or adapt this speed.*

## 8 ETA



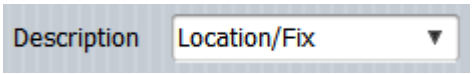

Estimated Time of Arrival setting. Use this value so that the entity will adapt its speed to reach the point at the desired time:

- Relative: duration to observe from the moment the instruction to head towards the point is received.
- Absolute: simulation time to observe.

## 9 Proximity

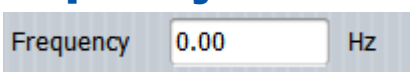

Minimal **distance** from the entity to the point used to consider the location reached or not. Used by MotionGoto component.

## 10 Description




Optional description that can also be used in Radio Navigation scenario.

## 11 Frequency

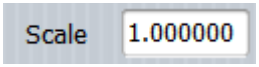



Optional frequency value that can be used in Radio Navigation scenario.

## 12 Symbol


If the default triangle symbol for point must be changed with another textured shape, use this field to specify the file.
Only TGA are supported with alpha channel set for transparencies.

## 13 Scale

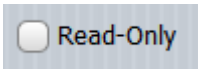



Scale of the textured shape (1 = default size)

## 14 Data Model


If a data model must be associated with the point (to get extra settings or give specific data to the user code), select it from the drop down list.
To set or change the parameters (interface) of the data model, use the button.
For example, a point might represent an airport. User can then create a data model named LocAirport with all kind of data some other components or logics might need to process. Using the data model attachment capability, user will be able to specialize any point of his scenario.

## 15 Read-Only

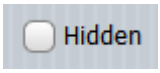



Check this option so that the user will **not** be able to **move** the point.

## 16 Grounded


Check this option so that the point altitude will always match the terrain level below it (**clamped**).

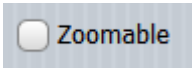
## 17 Hidden




Check this option to **hide** the point from the scenario map, when Feature layer not selected.


*Hidden features are always visible when the Feature layer is selected.*

## 18 Zoomable




Check this option for the point symbol (or textured shape) will automatically **grow** or **shrink** when zoom change.

# Trajectory Feature

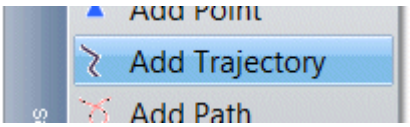
A Trajectory feature is a string of constrained waypoints intended to be followed by an entity.
Waypoints are linked with a straight line. A trajectory is not smoothed.
Like for other features, a trajectory will only be a support for a component or a logic.
A trajectory has extremity points and waypoints. Actions can be specified when an extremity point is reached.

A trajectory is not to be compared with an entity plan. A trajectory exists even without entity using it.
Also, several entities can use the same trajectory, at different time and speed.
A trajectory can also be followed with an offset, in one way or the other.

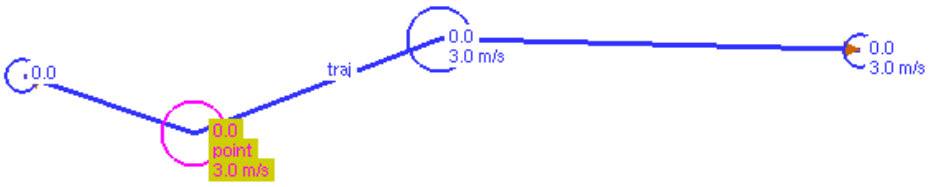
Trajectory can be modified at design and runtime.
Typical use of a trajectory is to provide direction and rough route following for entities, unit or some special zones (like clouds or bad weather).


*Built-in component which uses Trajectory feature: MotionFollow.*

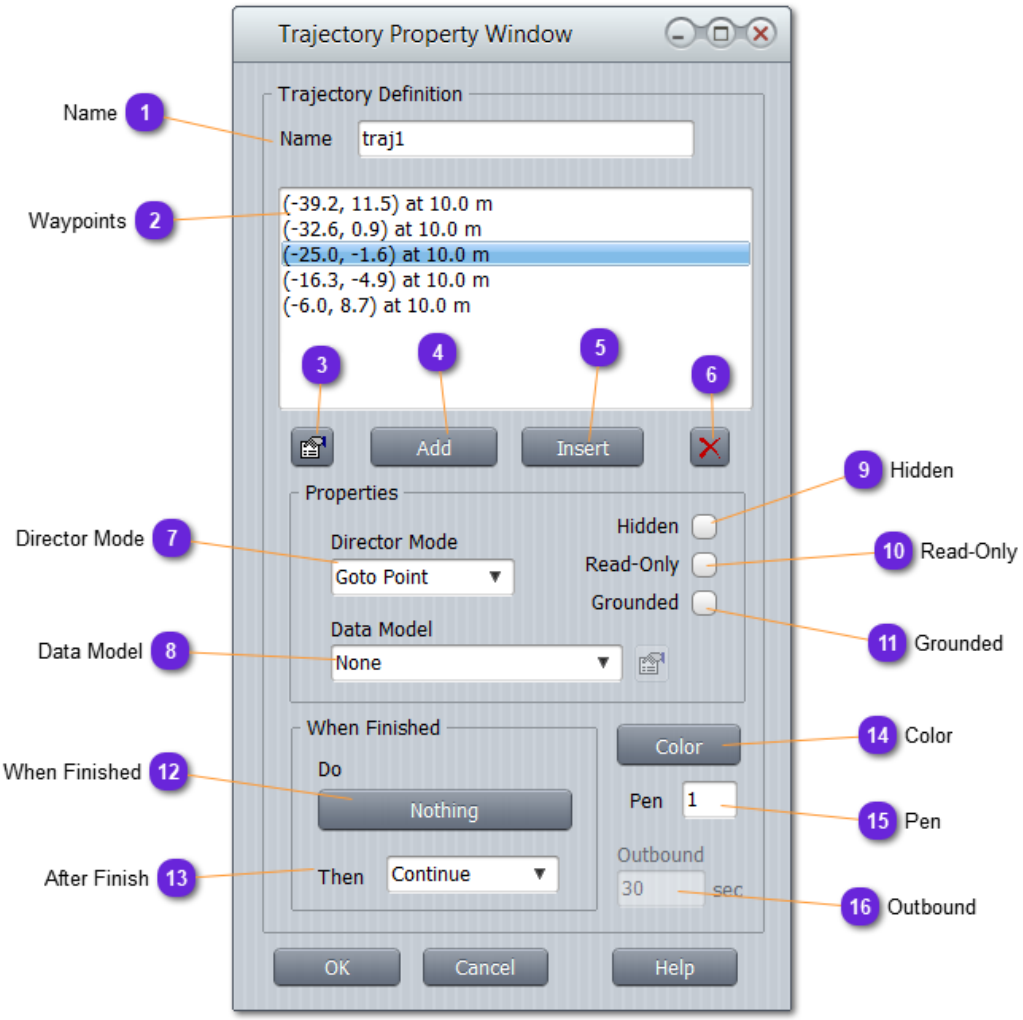
## • Popup Menu




*Example of a trajectory:*



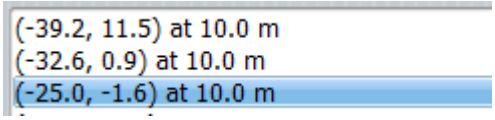
# Properties




## 1 Name


**Name** of the trajectory, unique in the scenario.

## 2 Waypoints

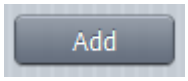



List of all the waypoints making the trajectory.
Are displayed in each line: **x,y** position and **proximity** value.

## 3 Properties

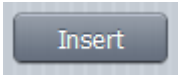
Call the property window of the selected waypoint.

## 4 Add




Add a new waypoints at the end of the trajectory. Mouse changes to **+**.
**Click** on the map to drop waypoints. **Right click** and Done to finish edition.

## 5 Insert

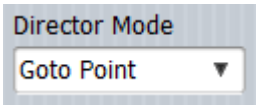



Add new waypoints after the selected waypoint in the list. Mouse changes to **+**.
**Click** on the map to drop waypoints. **Right click** and Done to finish edition.

## 6 Delete

**Delete** the selected waypoint in the list.

## 7 Director Mode




Informative information to the navigation modeler on how the trajectory is intended to be followed.

- Goto Point: Simple default mode. Entity will just head towards the next point regardless of its offset with the trajectory leg.
- Follow Leg: When heading to the next waypoint, entity will try to rejoin the leg.

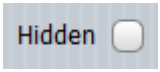
## 8 Data Model


If a data model must be associated with the trajectory (to get extra settings or give specific data to the user code), select it from the drop down list.

To set or change the parameters (interface) of the data model, use the button.

For example, a trajectory might represent an exit procedure. User can then create a data model named FireEngine with all kind of data some other components or logics might need to process. Using the data model attachment capability, user will be able to specialize any trajectory of his scenario.

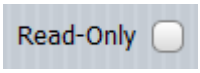
## 9 Hidden




Check this option to **hide** the trajectory from the scenario map, when Feature layer not selected.


*Hidden features are always visible when the Feature layer is selected.*

## 10 Read-Only

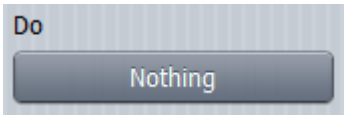



Check this option so that the user will **not** be able to **move** or **reshape** the trajectory.

## 11 Grounded

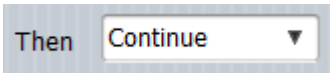

Check this option so that all waypoints altitude will always match the terrain level below them (**clamped**).

## 12 When Finished




When the last point of the trajectory is reached, if an **event** must be sent, use it here.
See event setting here.

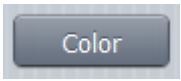
## 13 After Finish




When an extremity of the trajectory is reached, here is the advised **procedure**:

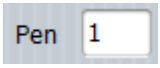
- Continue: leave the trajectory and keep the same speed and heading.
- Stop: Set speed to zero. Same heading.
- Loop: Go back to first waypoint
- Inverse: Do a U-turn maneuver to reenter the trajectory from the last point and follow it the other way, then back again when extremity is reached.

## 14 Color




**Color** used to draw the trajectory on the map.

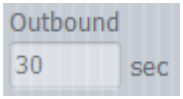
## 15 Pen




Specify here the **thickness** of the trajectory when drawn on the map.

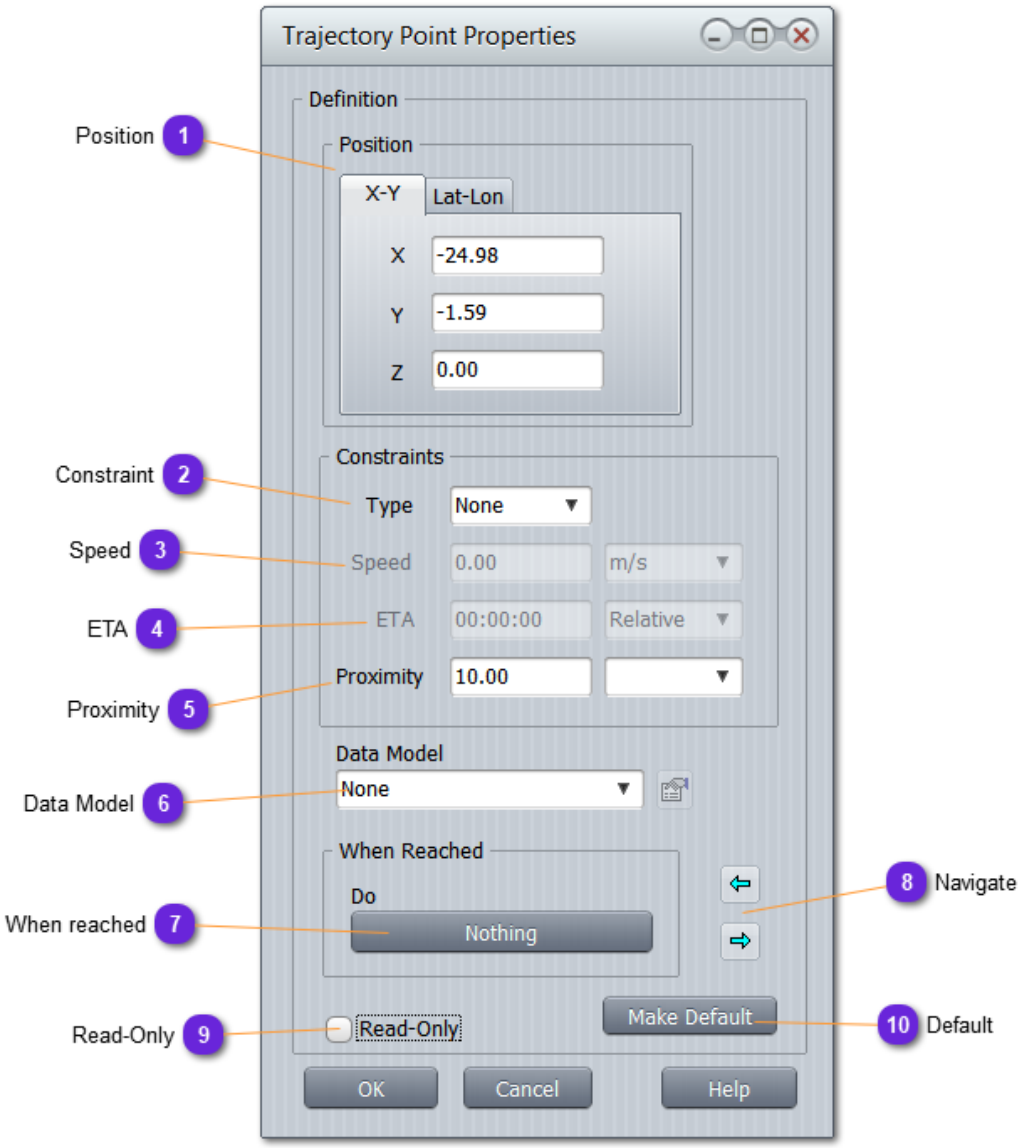

*Cannot be 0, use Hidden for that.*

## 16 Outbound

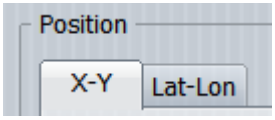



When end procedure is set to Inverse (13), specify here the **time** in seconds the entity will recede from the last point before engaging the U-turn maneuver.

# Waypoint

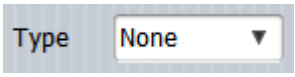



## 1 Position




**Position** in coordinates XYZ or latitude, longitude, altitude of the waypoint on the map.
Normally, these values are coming from the mouse positioning at creation time.
Can be updated here manually.

## 2 Constraint




Specify here the type of **constraint** that must use the component MotionFollow for any entity having to direct towards the waypoint.

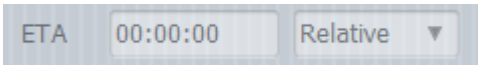
- None: nothing selected
- Speed: speed constraint. See (6)
- ETA: time constraint. See (7)

## 3 Speed


If speed constraint is selected, specify here (in the desired unit), the maximum speed the entity must reach to direct towards this waypoint.


*MotionFollow uses this constraint but any user component or logic can do the same or adapt this speed.*

## 4 ETA




Estimated Time of Arrival setting. Use this value so that the entity will adapt its speed to reach the waypoint at the desired time.

- Relative: duration to observe from the moment the instruction to head towards the point is received.
- Absolute: simulation time to observe.

## 5 Proximity


Minimal **distance** from the entity to the waypoint used to consider it reached or not. Used by MotionFollow component.

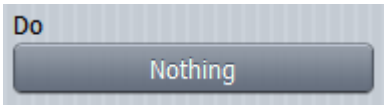
## 6 Data Model


If a data model must be associated with the waypoint (to get extra settings or give specific data to the user code), select it from the drop down list.

To set or change the parameters (interface) of the data model, use the button.

For example, a waypoint might represent a fix or a navaid. Using the data model attachment capability, user will be able to specialize any waypoint of his trajectory.

## 7 When reached

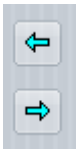



This **event** setting is only informative and used by built-in component MotionFollow.

Use can set here any event and trigger it from any user code.

See event setting here.

## 8 Navigate

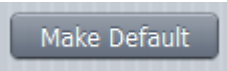


Use these button to navigate from one waypoint to the **next** (**right** arrow) or **previous** one (**left** arrow).

## 9 Read-Only


Check this option so that the user will **not** be able to **move** the waypoint.

## 10 Default




If clicked, **all parameters** of this waypoint (except its position x,y) will be **copied** to all other waypoints, including the altitude.

# Path Feature

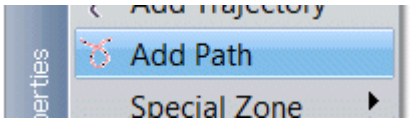
A Path feature is a string of constrained control points intended to be followed by an entity. Control points are linked with a smooth Bezier line.
Like for other features, a path will only be a support for a component or a logic.
A path has extremity points and points. Actions can be specified when an extremity point is reached.

A path is not to be compared with an entity plan. A path exists even without entity using it. Also, several entities can use the same path, at different time and speed.

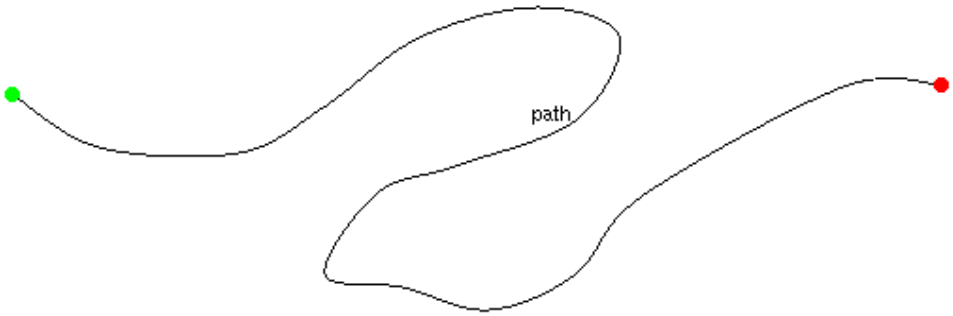
Path can be modified at design and runtime.
Typical use of a path is to provide a nice and smooth behavior for entities without dynamic component.


*Built-in component which uses Path feature: MotionSlide.*

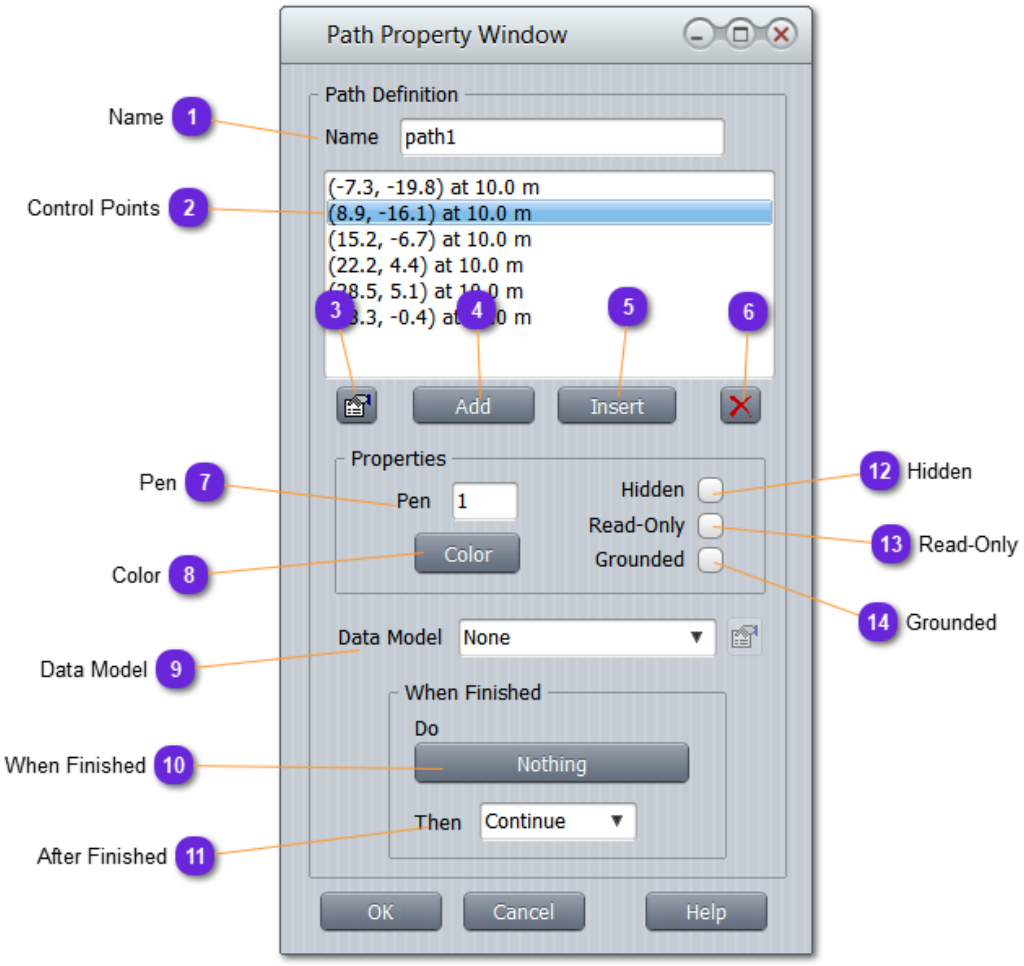
## • Popup Menu




*Example of a path:*



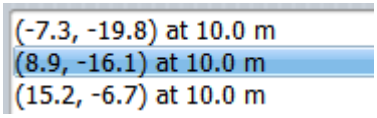
# Properties




## 1 Name


**Name** of the path, unique in the scenario

## 2 Control Points



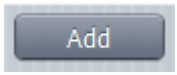

List of all the control points making the path.
Are displayed in each line: **x,y** position and **proximity** value.


*Control points define the shape of the Bezier spline.*

## 3 Properties

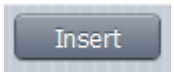
Call the property window of the selected control point.

## 4 Add




Add a new control point at the end of the path. Mouse changes to **+**.
**Click** on the map to drop points. **Right click** and Done to finish edition.

## 5 Insert

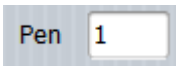



Add new control point after the selected point in the list. Mouse changes to **+**.
**Click** on the map to drop points. **Right click** and Done to finish edition.

## 6 Delete

**Delete** the selected control point in the list.

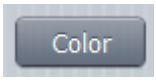
## 7 Pen




Specify here the **thickness** of the path when drawn on the map.


*Cannot be 0, use Hidden for that.*

## 8 Color



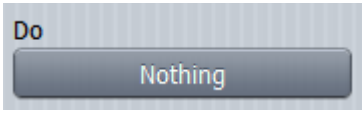

**Color** used to draw the path on the map.

## 9 Data Model


If a data model must be associated with the control point (to get extra settings or give specific data to the user code), select it from the drop down list.

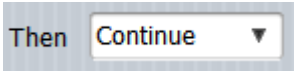
To set or change the parameters (interface) of the data model, use the button.
For example, a path might represent an exit procedure. User can then create a data model named FireEngine with all kind of data some other components or logics might need to process. Using the data model attachment capability, user will be able to specialize any path of his scenario.

## 10 When Finished




When the last point of the trajectory is reached, if an **event** must be sent, use it here.
See event setting here.

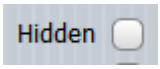
## 11 After Finished




When an extremity of the path is reached, here is the advised **procedure**:

- Continue: leave the path and keep the same speed and heading.
- Stop: Set speed to zero. Same heading.
- Loop: Go back to first control point
- Inverse: Do a U-turn maneuver to reenter the path from the last point and follow it the other way, then back again when extremity is reached.

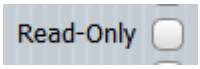
## 12 Hidden




Check this option to **hide** the path from the scenario map, when Feature layer not selected.


*Hidden features are always visible when the Feature layer is selected.*

## 13 Read-Only




Check this option so that the user will **not** be able to **move** or **reshape** the path.

## 14 Grounded


Check this option so that all waypoints altitude will always match the terrain level below them (**clamped**).

# Control Point

Path control point setting is similar to trajectory waypoint.
See properties here.

# Special Zones Feature

Special Zone feature (sp-zone) is a set of ground shapes or volumes, added on the terrain map, to represent areas of interest.
An sp-zone can represent anything and is intended to be used by components or logics but can also just be informative for the scenario designer.

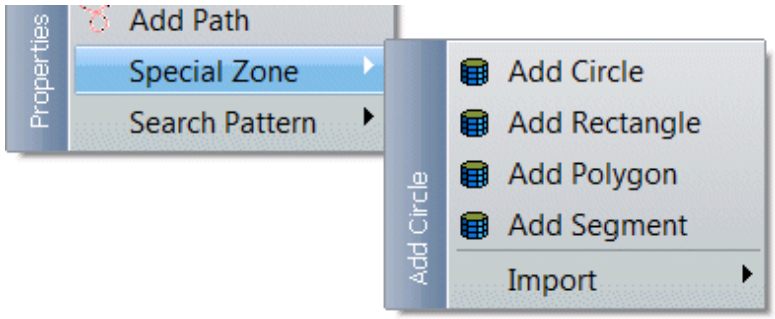
A special zone can describe only one area or a group of different areas. These areas can be represented (and drawn) by predefined types:

- Circle
- Sphere
- Cylinder
- Rectangle
- Box
- Segment
- Polygon
- Volume

Areas can be created at runtime, from the design editor or from the user code. They can be resized, moved and deleted.
A typical usage of a sp-zone is for mine field, dangerous areas, weather areas, etc.


*Built-in component which uses Special Zone feature: SpecialZone.*

## • Popup Menu




**Add Circle**: see here

**Add Rectangle**: see here

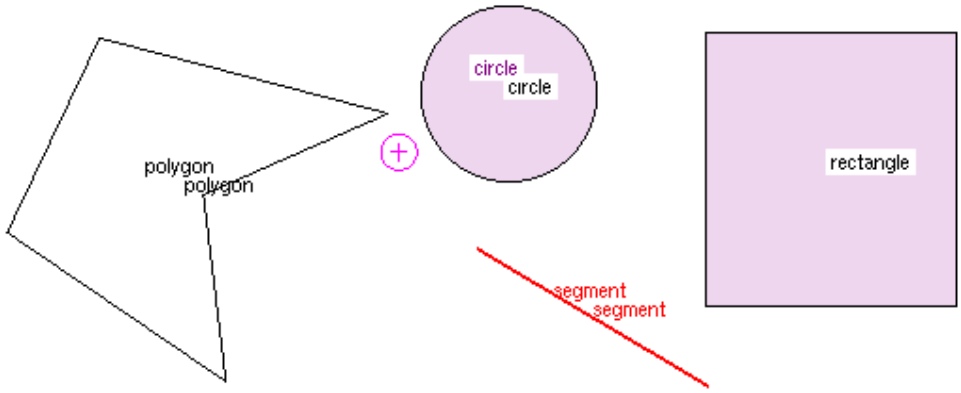
**Add Polygon**: see here

**Add Segment**: see here

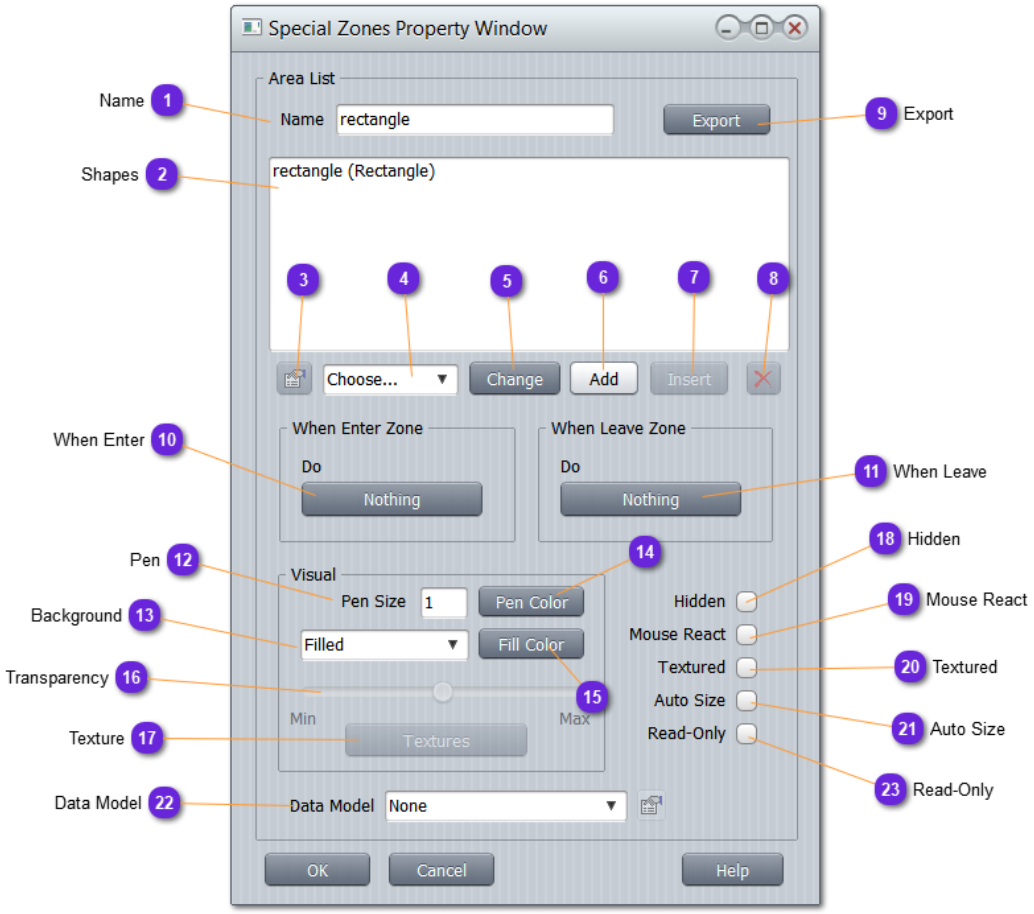
**Import**: list all special zones (of any type) exported into /data/shared/features

*Example of special zones:*

## Special Zones



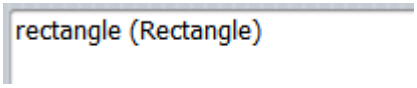
# Properties




## 1 Name


Unique **name** of the special zone.

## 2 Shapes



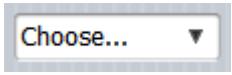

List of all the **areas** (shapes) defining the special zone.
For example, a sand region can be made of several shapes distributed on the terrain. Gathering several shapes into a unique zone will faster the processing of such zone as only one component will be needed. It also can make sense to combine different types of areas into the same zone, if the purpose is to avoid them.

## 3 Properties

Open the property window of the selected area (see Circle, Rectangle, Polygon or Segment).

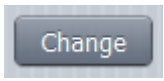

*Areas are processed and drawn from top to bottom.*

## 4 Shape Type



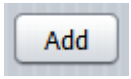

Select the **shape** type from the list (to be used with Change, Add and Insert)

## 5 Change



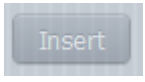

**Change** the shape of the selected area with the drop down shape selector (4)

## 6 Add




**Add** a new area of the selected shape (4) at the **end** of the list.

## 7 Insert

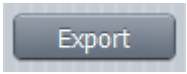



**Insert** a new area of the selected shape (4) **after** the **selected** area in the list.

## 8 Delete

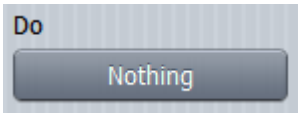
**Remove** the **selected** area from the zone list.

## 9 Export



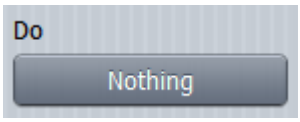

Special zones can be **exported** to /data/shared for later reuse (through import function)

## 10 When Enter




Set here the event to be triggered when an entity enters any areas of the zone.
This behavior must be handled by the SpecialZone component.

## 11 When Leave




Set here the event to be triggered when an entity leaves any areas of the zone.
This behavior must be handled by the SpecialZone component.

## 12 Pen


Specify here the **thickness** of the area shape outline when drawn on the map.

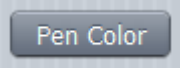

*Use 0 to remove the outline frame.*

## 13 Background


Specify is the area must be filled or just outlined.

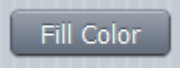
- Empty: no filling
- Filled: opaque filling
- Transparent: transparent filling

## 14 Pen Color




**Color** used to draw the outline of the areas.

## 15 Fill Color

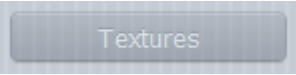



**Color** used to fill the area shape.

## 16 Transparency


When the filling is set to Transparent, use this scroll bar to set the transparency level from Min (full opacity) to Max (invisible)

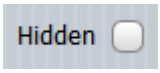
## 17 Texture




Select the file that will be used to texture the area shapes. Textured check box (20) must be checked.


*Supported types: TGA, JPEG, PNG.*

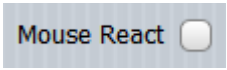
## 18 Hidden




Check this option to **hide** the zone (all areas) from the scenario map, when Feature layer not selected.


*Hidden features are always visible when the Feature layer is selected.*

## 19 Mouse React




The special zone will trigger the event when the mouse enters and leaves areas.

## 20 Textured


When textured, the file (17) will be used and the filling setting will be skipped.

## 21 Auto Size

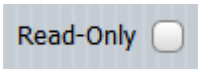

When checked, the size of each defined areas will be resized according to the width and height of the loaded texture file.

## 22 Data Model


If a data model must be associated with the special zone (to get extra settings or give specific data to the user code), select it from the drop down list.

To set or change the parameters (interface) of the data model, use the button.
For example, a special zone might represent a mine field. User can then create a data model named MineField with all kind of data some other components or logics might need to process. Using the data model attachment capability, user will be able to specialize any zone of his scenario.

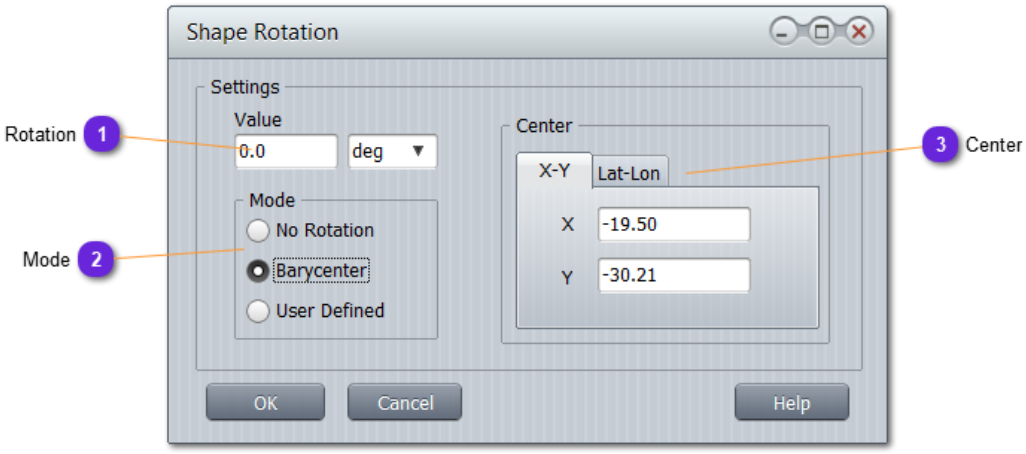
## 23 Read-Only



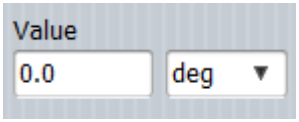

Check this option so that the user will **not** be able to **move** or **reshape** the special zone.

# Rotation

Specify how a shape must be rotated.

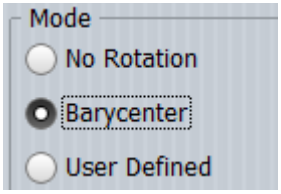



## 1 Rotation




Set here the rotation **value** you want to apply on the shape.

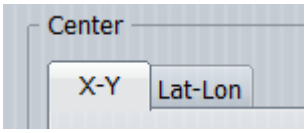
## 2 Mode




Select which rotation **mode** to apply:

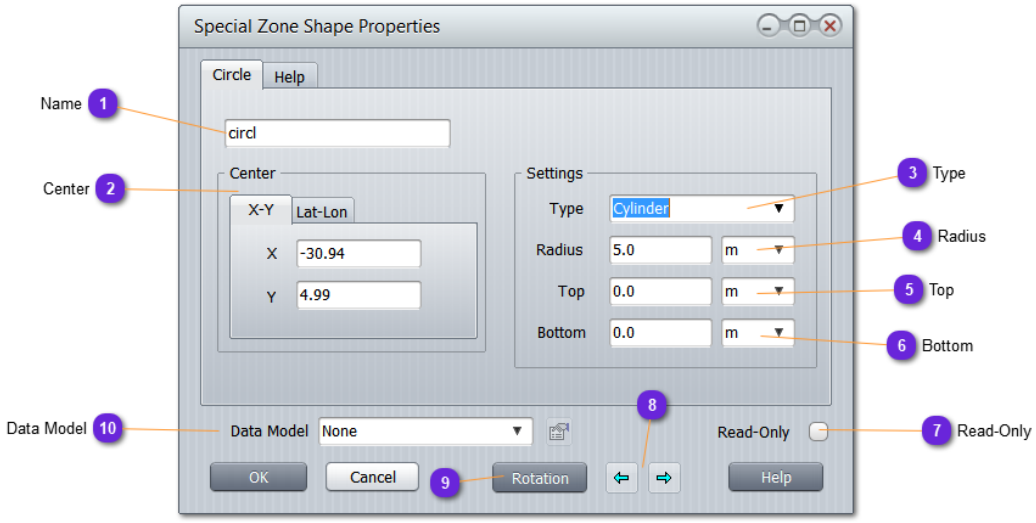
- No Rotation: nothing will be done (rotation value will be kept unchanged but ignored)
- Barycenter: the barycenter of the area will be used as the rotation point.
- User Defined: use the Center block to define it.

## 3 Center




Rotation center coordinates.
Editable only when User Defined mode is selected.

# Circle

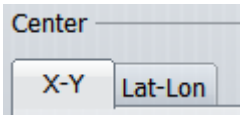



## 1 Name


Name of the area in the zone.

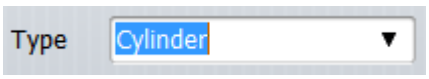

*The area name is useful when having to delete or access from user code some specific areas.*

## 2 Center




Coordinates of the center of the shape.
User modifiable.

## 3 Type




Select here the **type** of volumetric shape:

- Ground: circle clamped on the ground.
- Cylinder: vertical cylinder with a top and bottom, to make it floating.
- Sphere: 3D sphere.

## 4 Radius


Set here the **radius** of the circle/cyclinder/sphere, in the selected unit.

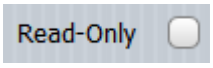
## 5 Top


If type is Cylinder, set the **top** altitude of the shape.

## 6 Bottom


If type is Cylinder, set the **bottom** (base) altitude of the shape.

## 7 Read-Only

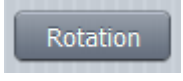



When checked, the area shape cannot be modified.

## 8 Navigate

Use these buttons to **navigate** from one area to the next one (if many), back and forward.

## 9 Rotation




Activated only when the area shape can be rotated.
See here.

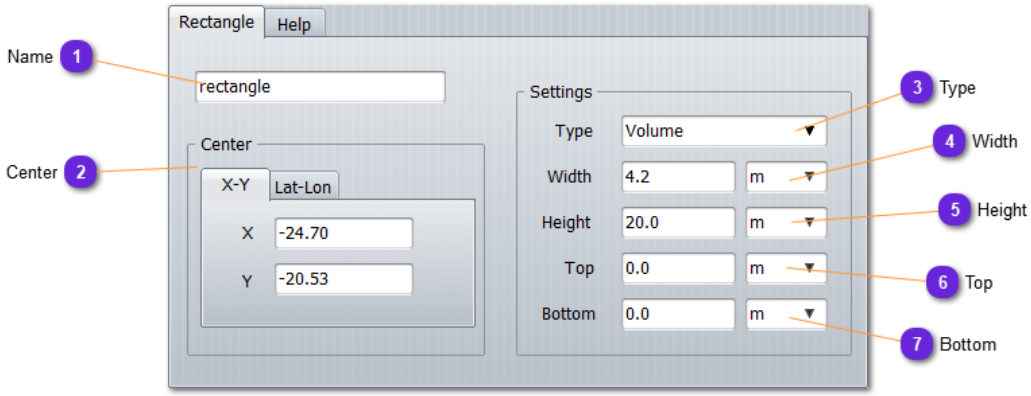
## 10 Data Model


If a data model must be associated with the area (to get extra settings or give specific data to the user code), select it from the drop down list.

To set or change the parameters (interface) of the data model, use the button.

For example, an area might represent a sand pit. User can then create a data model named SandPit with all kind of data some other components or logics might need to process. Using the data model attachment capability, user will be able to specialize any area of his scenario.

# Rectangle

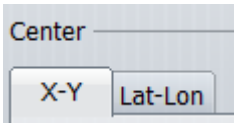



## 1 Name


Name of the area in the zone.

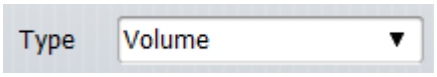

*The area name is useful when having to delete or access from user code some specific areas.*

## 2 Center




Coordinates of the center of the shape.
User modifiable.

## 3 Type




Select here the **type** of volumetric shape:

- Ground: rectangle clamped on the ground.
- Volume: 3D box with a top and bottom, to make it floating.

## 4 Width


Horizontal **width** of the rectangle, in the selected unit.

## 5 Height


Vertical **height** of the rectangle, in the selected unit.

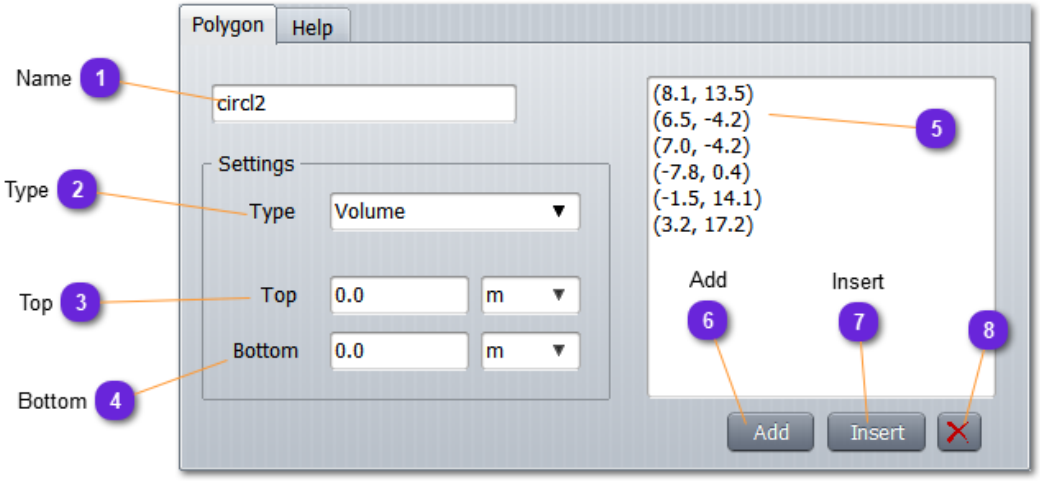
## 6 Top


Set here the **top** altitude of the box.

## 7 Bottom


Set the **bottom** altitude of the box.

# Polygon

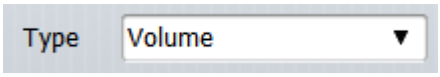



## 1 Name


Name of the area in the zone.


*The area name is useful when having to delete or access from user code some specific areas.*

## 2 Type




Select here the **type** of volumetric shape:

- Ground: polygon clamped on the ground.
- Volume: 3D volume with a top and bottom, to make it floating.

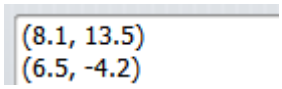
## 3 Top


Set here the **top** altitude of the volume.

## 4 Bottom

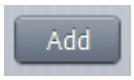

Set the **bottom** altitude of the volume.

## 5 Point



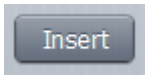

List of all the **points** of the polygon.

## 6 Add




**Add** a new point at the end of the list.

## 7 Insert

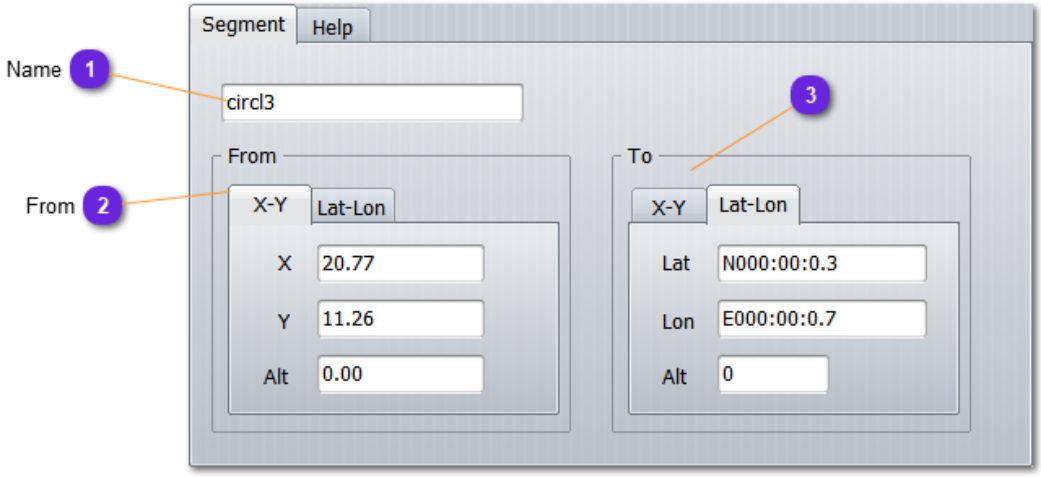



**Insert** a new point after the selected one in the list.

## 8 Delete

**Suppress** the selected point.

# Segment

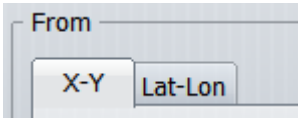



## 1 Name


Name of the area in the zone.

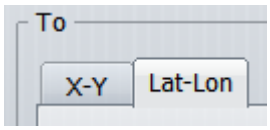

*The area name is useful when having to delete or access from user code some specific areas.*

## 2 From




Position of the **from** point of the segment.

## 3 To



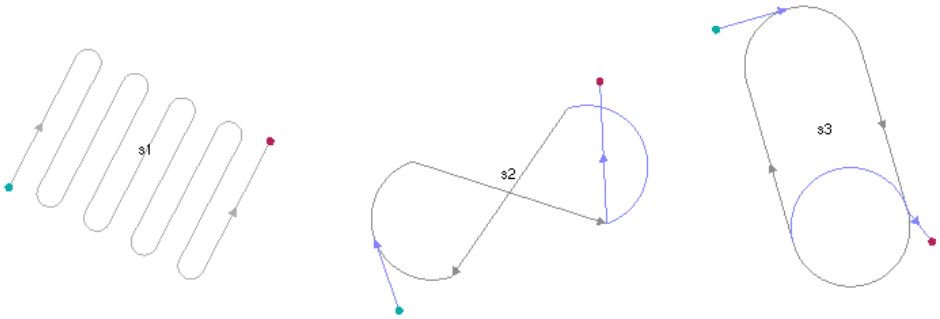

Position of the **to** point of the segment.

# Search Patterns Feature

Search patterns are predefined **rectangular areas** including a path with a predefined shape. Several different shapes are offered.
They are intended to be followed by all kind of entities (submarine, surface, ground or air vehicles).

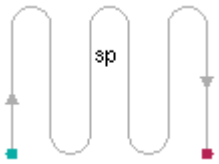
A search pattern is typically used as an holding pattern, by an aircraft, before landing or as a search procedure for a patroller, a UAV or a ground robot.
The path inside a search pattern can be **one** or **both ways**.
It has two **extremity** points: an entry (in green) and an exit (in red). These points cannot be modified.


*Built-in component which uses Search Pattern feature: SearchPattern.*



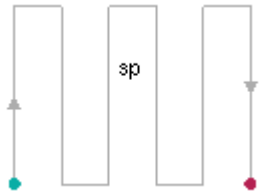

## • Shape Types

### Creeping U

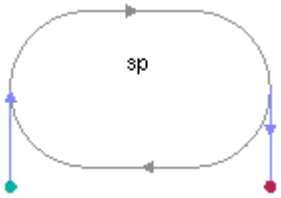



### Creeping L

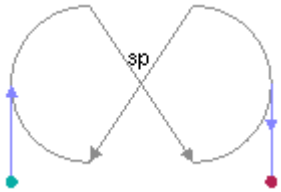
### Search Patterns




### Holding




### 8 Shape




### Circular

*Not available*

### Spiral

*Not available*

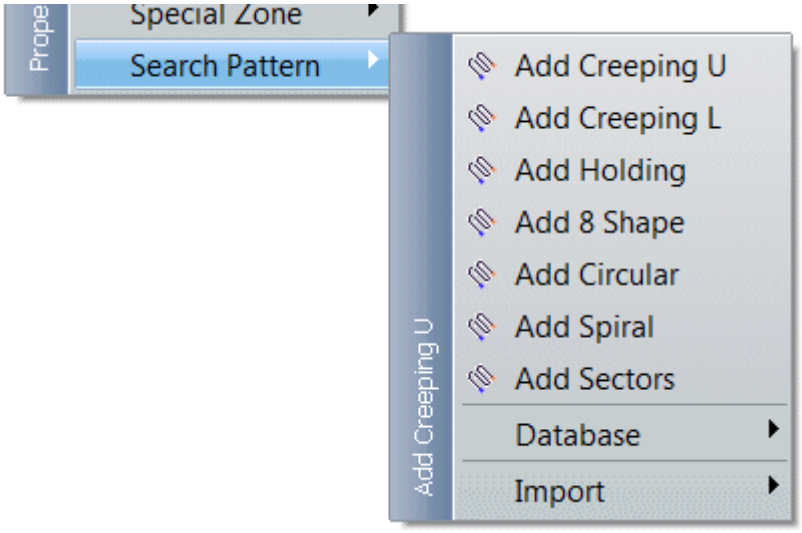
### Sectors

*Not available*

### User Defined

With such type, only the rectangular area is considered. There is no predefined path shape.
User will need to provide its own component to control the entity into the search area.
Typically, this is used for random trajectory manager to avoid predictive planning, for evasive drone for example.

## • Popup Menu

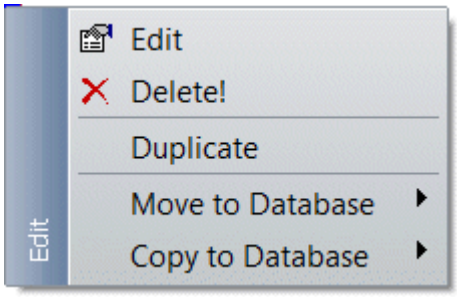



**Add Creeping U:**
**Creeping L:**
**Holding:**
**8 Shape:**
**Circular:**
**Spiral:**
**Sectors**: create a search pattern from the predefined type.

**Database**: list all search patterns stored in the Database (see here).

**Import**: list all patterns (of any type) exported into /data/shared/features.

When a search pattern is **selected**:




**Edit**: popup the property window

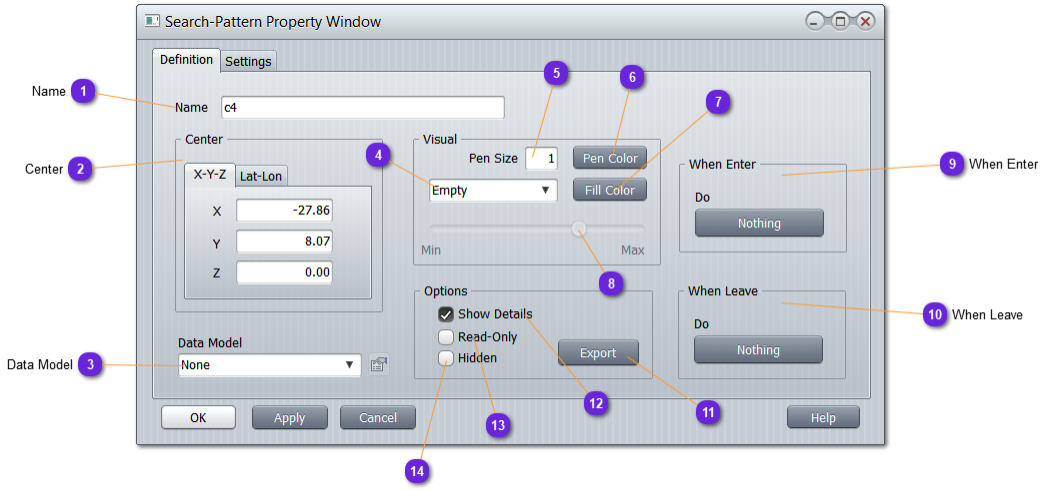
**Delete**: suppress the selected search pattern.

**Duplicate**: copy/paste the selected pattern.

**Move to Database**: copy the selected pattern to the database environment and remove it from the current scenario.

**Copy to Database**: copy the selected pattern to the database environment.

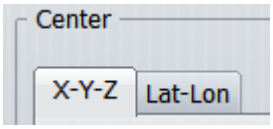
# Properties




## 1 Name


Name of the search pattern.

## 2 Center




**Position** of the search pattern center.
Can be modified manually.

## 3 Data Model


If a data model must be associated with the search pattern (to get extra settings or give specific data to the user code), select it from the drop down list.

To set or change the parameters (interface) of the data model, use the button.

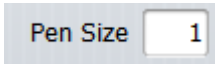
For example, a pattern might represent an holding procedure. User can then create a data model named HoldingProc with all kind of data some other components or logics might need to process. Using the data model attachment capability, user will be able to specialize any search pattern of his scenario.

## 4 Background


Specify the way the rectangular pattern **area** will be drawn

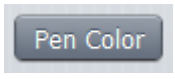
- Empty: no background color
- Opaque: fill the rectangular area with selected color.
- Transparent: fill the rectangular area with transparent selected color.

## 5 Pen Size



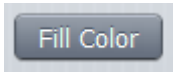

Specify the **width** of the **pen** used to draw the path inside the search area.

## 6 Pen Color




Select the **color** of the **pen** used to draw the path inside the search area.

## 7 Fill Color

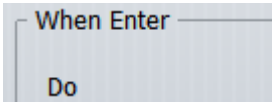



Select here the **background color** to use for filling the pattern area.

## 8 Transparency


When background type is Transparent (4), use the slider to define the opacity level of the Fill color (7).
Min (glass), Max (opaque)

## 9 When Enter

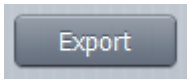



Set here the event to be triggered when an entity goes by the entry point of the search pattern.
This behavior is internally handled by the SearchPattern component.

## 10 When Leave

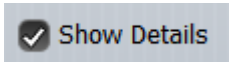

Set here the event to be triggered when an entity goes by the exit point of the search pattern.
This behavior is internally handled by the SearchPattern component.

## 11 Export




Use this function to **save** the search pattern into a file in /data/shared/features directory.
Exported search pattern can then be reused into another databases.
To share search patterns between scenarios (of the same database) or store them for runtime, use the Pattern environment save/move function of the popup menu.

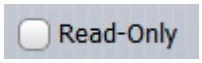
## 12 Show Details




When checked, the path (including entry/exit points) will be drawn inside the area.
If unchecked, only the area rectangle will be drawn.

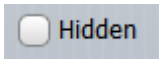
*This only apply when Features layer is **not** selected.*

## 13 Read-Only




When checked, the pattern will not be modifiable or displaced on the map.

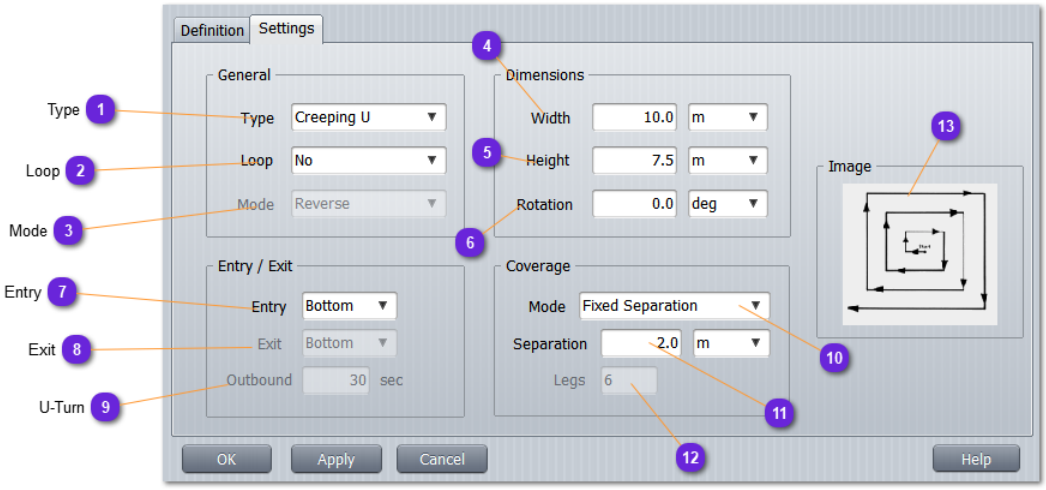
## 14 Hidden



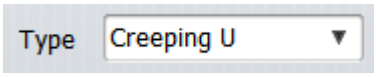

When checked, the rectangular area (including the path) will not be drawn on the map. Pattern will be invisible.

*This only apply when Features layer is **not** selected.*

# Settings




## 1 Type



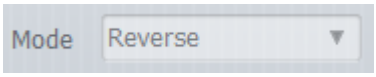

Chose in the list which type of path to use for the pattern. See here.

- Creeping U
- Creeping L
- Holding
- 8 Shape
- Circular
- Spiral
- Sectors
- User Defined

## 2 Loop


Set the one time only of do it again mode.
If No is selected, pattern will be left when exit point is reached.
If Yes is selected, pattern will be followed again according to Mode (3) when extremity point is reached.

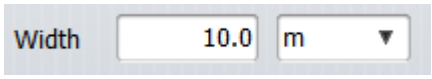
## 3 Mode




Use to define the loop mode (2) when activated.

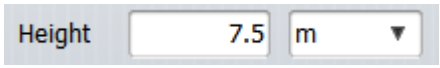
- Reverse: component manager will initiate a special maneuver to recede from the extremity point, then **U-turn** before entering the pattern from the last extremity point.
- Continue: works only with some **circular** kind of shapes. Exit point will be headed only when the component manager will receive the order to leave.
- Restart: component will guide the entity towards the **entry point** again using a predefined procedure which removes the aircraft from the reentry point until it is able to properly return without over shooting too much during the U-turn.

## 4 Width



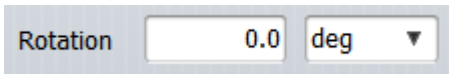

**Width** of the rectangular area, in the selected unit. User modifiable.

## 5 Height



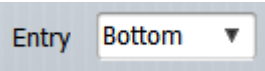

**Height** of the rectangular area, in the selected unit. User modifiable.

## 6 Rotation




**Rotation** angle of the rectangular area, in the selected unit. 0 for horizontal, 90 for vertical (in degrees). User modifiable.

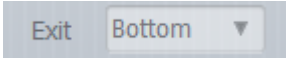
## 7 Entry




Position of the **entry point** regarding the rectangular area.

- Top: upper part of the area.
- Bottom: lower part of the area.

## 8 Exit



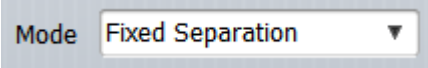

Position of the **exit point** regarding the rectangular area.

- Top: upper part of the area.
- Bottom: lower part of the area.

## 9 U-Turn


In case of Reverse mode (3), specify the **time** in seconds the entity will **recede** before engaging the U-turn maneuver.

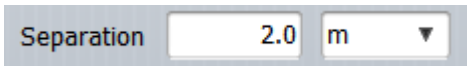
## 10 Creeping




Works only for type Creeping U and Creeping L
Specify how the legs must be computed to fill the rectangular area

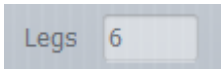
- Fixed Separation: legs will be spaced with the same exact value specified in Separation (11). It might not fill the area for big values of the ratio **width / separation**.
- Fixed Number of Legs: legs will be spaced with a computed value **width / legs**.

## 11 Separation




Set here the intended **separation** value between legs of the creeping shape.

## 12 Legs




Set here the intended **number** of legs to be used for the creeping shape.

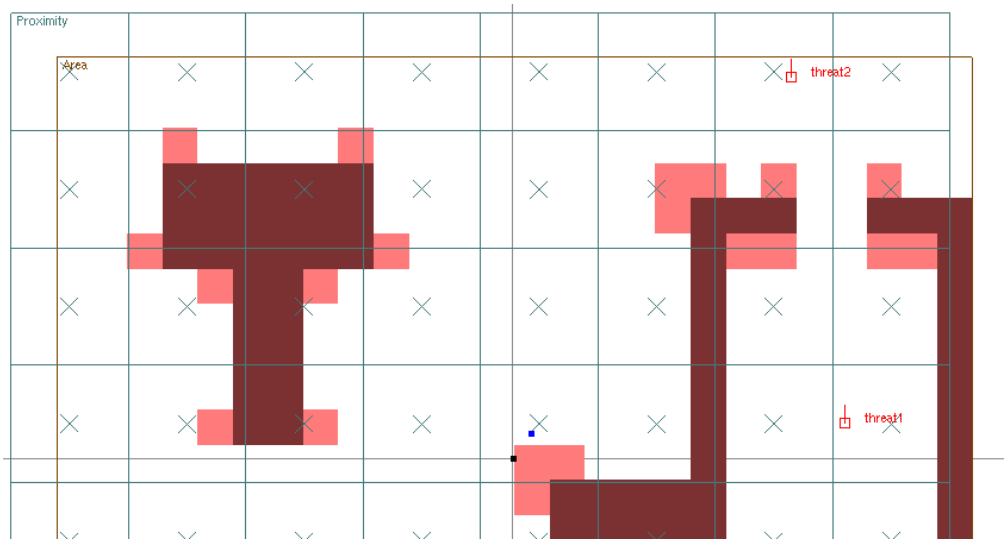
## 13 Preview


*Not available*

# Meshes

Meshes are rectangular grid that are dropped on the terrain in order to manually cover some areas with specific elements.
Several meshes of different types, size and density can be used on the same terrain.
They can overlap although the component working with them can conflict if two Meshes of the same type are overlapping.

A Mesh can be seen as a matrix of nodes.



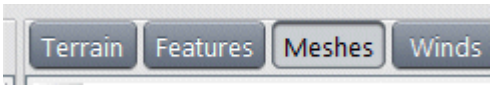

## • Mesh Types

There is two kind of meshes

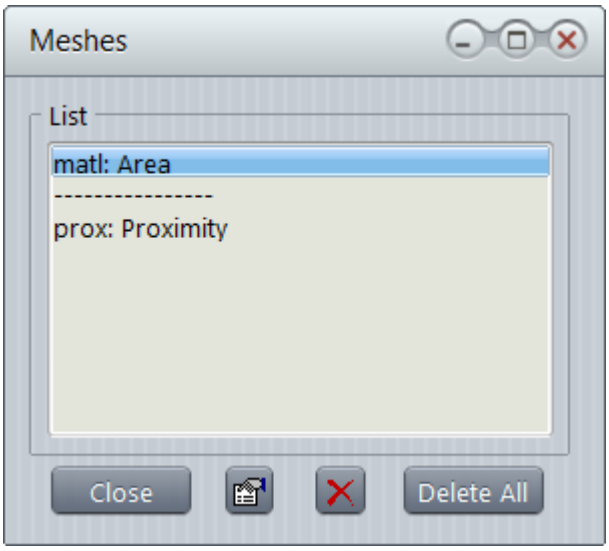
**Material**: contains nodes that holds special tags to describe the mesh and drive the path following algorithm.

**Proximity**: contains nodes that holds list of entities belonging to the node area. This mesh is for runtime only.

To **access** this layer, depress the following Meshes button:




**Double click** the background to list all defined meshes.




## • Mesh Nodes

Each node is linked with the surrounding ones using pointers, to speed up the traversing of the mesh. A node holds some user values.
Three predefined nodes are provided:

**TagNode**: used by PathFinding component. Can hold any of the following tags:

**ProxNode**: proximity node, used for entity collision and to speed up the detection algorithm with thousands of entities. Used by Proximate component.

**UserNode**: can hold any data user want to drop into the node. Specific component must be written to handle such mesh.

## • Toolbar

When a mesh is selected, the distance string is used to initiate the path finding algorithm between the two extremities. A default cost function is taken, penalizing the diagonal moves.

Creates a new mesh. Cursor changes to . Set the **top-left** corner of the mesh, depress mouse button, hold it and move the mouse to the **bottom-right** corner of the mesh.
Release the mouse button and set the property window.

When a **mesh** is **selected**, use the following tool to **paint** the mesh with **tags**:

### Meshes

 Clears the content of the node. *Material mesh only.*

 Set the node tag to a blocking wall (cannot pass). *Material mesh only.*

 Set the node tag to walkway. A walkway is always preferred over a cleared node. *Material mesh only.*

 Set the node tag to no walk tag. The path following algorithm will give a high cost to this node. Not blocking if no other way. *Material mesh only.*

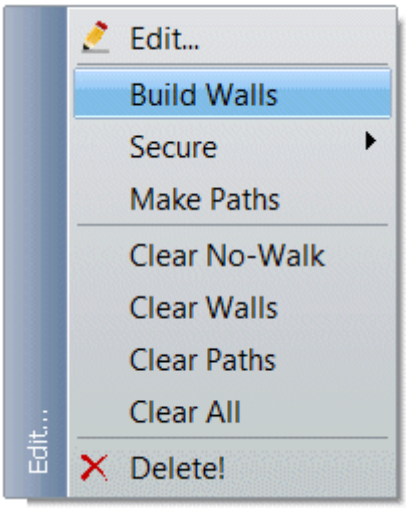
## • Popup Menu

**Right click** the background to add a mesh:


**Mesh**: add a new mesh layer

**Import**: import from /data/shared directory a previously exported mesh.

**Right click** any selected mesh to display the popup menu:




**Edit**: call the property window.

**Build Walls**: automatically scan the terrain below the mesh and add walls wherever the slope or height delta is above the threshold Auto Wall defined in the mesh settings.

**Secure**: **Corners**: add a no walk tag at wall corners only; **Walls**: add no walk tags around all walls (see Wall Coat in mesh settings).

**Make Paths**: try to build walkways between walls to avoid path finding to raze walls.

**Clear No-Walk**: remove all no walk tags from the mesh.

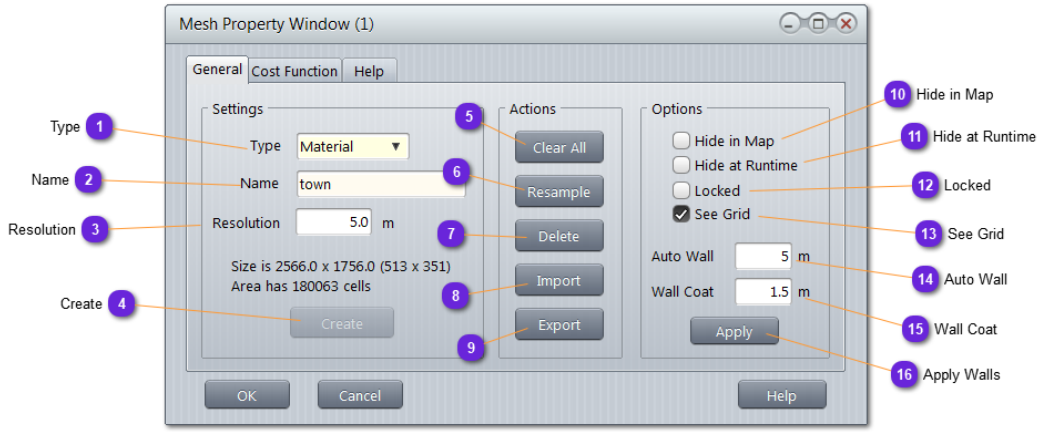
**Clear Walls**: remove all wall tags from the mesh.

**Clear Paths**: remove all walkway tags from the mesh.

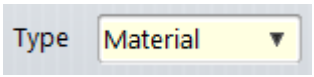
**Clear All**: remove all tags from the mesh.

**Delete**: remove the mesh from the scenario.

# Properties



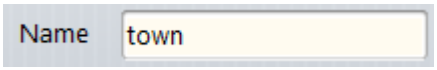

## 1 Type




Type of mesh:

- Material: use to store tag for each node
- Proximity: use for entity grouping to accelerate collision and close detections.

## 2 Name

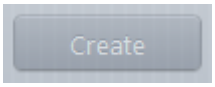



Unique **name** of the mesh.

## 3 Resolution

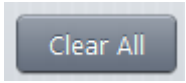

Width or radius of each node.
For **material** mesh, the smaller the value, the bigger the mesh and the accuracy of the sampling.
For **proximity** mesh, it is better to have a reasonable size for a node as too many will reduce drastically the efficiency of the Proximate component.

## 4 Create



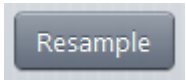

If the mesh has not been created yet, use this button to create it. Will be grayed out at edition.

## 5 Clear All



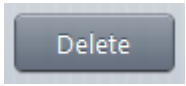

Remove all tags or user data from all nodes of the mesh.

## 6 Resample



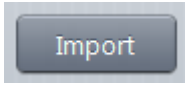

If the resolution is changed, the mesh needs to be recreated. Clear All command is called before recreating a new mesh with the new resolution.

## 7 Delete



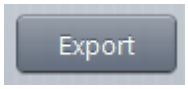

Delete all nodes of the mesh, physically, from the memory. You can force the deletion before creating again with the same or a different resolution.

## 8 Import



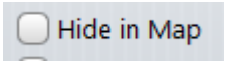

Will delete the current mesh and replace it with the selected one from the /data/shared directory.

## 9 Export




Export the current mesh to /data/shared for later import.

## 10 Hide in Map



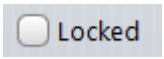

If checked, the mesh and content (for material) will not be displayed on the terrain, even if selected.


*Meshes are always displayed when Meshes layer is selected.*

## 11 Hide at Runtime

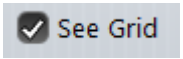

If checked, the mesh and content will not be displayed on the simulation engine if OpenGL viewer (or equivalent) is selected.
Use this option when Hide in Map (10) is not checked. The mesh will be visible on the GUI map (for info) but not on the runtime window (for performances or other reasons).

## 12 Locked




If locked, the mesh cannot be displaced with the mouse.

## 13 See Grid

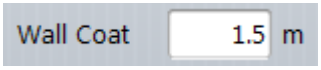



When checked, the center of each node is mark with an orange cross. It materialize the density of the node.

## 14 Auto Wall

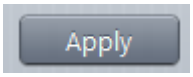

To be used with popup menu **Build Walls** command.
Specify the minimum height step (up or down, in meters) which represent an insurmountable obstacle for the path finding algorithm.

## 15 Wall Coat




To be used with popup menu **Secure Walls** command.
Specify the width (in meters) of the No Walk tag area around walls.

## 16 Apply Walls



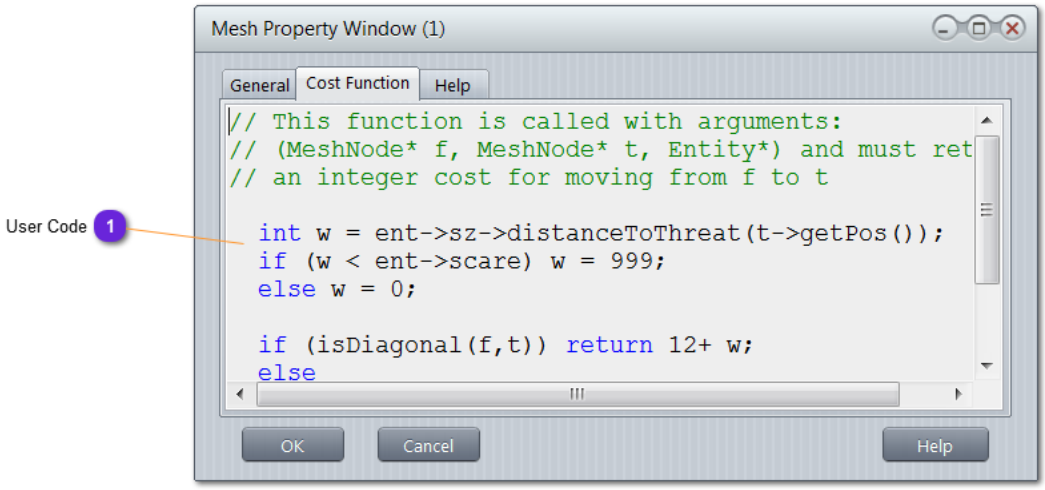

Clear all tags of the Mesh (same as Clear All (5) or popup menu **Clear All**), call the wall detection procedure using Auto Wall value (same as popup menu **Build Walls**),
secure the walls using the Wall Coat value (same as popup menu **Secure Walls**).

# Cost Function

Path finding algorithm (PathFinding component) use a cost function.
The user can change the default cost function as needed. Parameters are specified in the comments.

The value returned by the function is unit less. What only matters is the value itself compared to others. Returning a low cost will increase chance for the path-finding algorithm to keep the node while returning a high cost will probably end up with a rejection of the cell.

In the cost function, not only can you query the cell tag, but also the cells around it or even the entity location or attributes (getting too far from a destination, etc.)
The cost function is the most important part of the algorithm and the more sensitive.




## 1 User Code

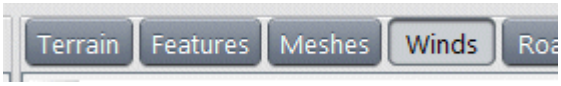
```
int w = ent->sz->distanceToThreat(t->getPos());
if (w < ent->scare) w = 999;
```

Enter here the cost function that will return the arbitrary value of the node, according to any kind of conditions.
What is important is not the value itself but its ranking regarding the other node values for the same mesh and for one path search (per entity).
The function must return the cost of moving from the node `f` to the node `t` for the entity `ent`.
The **lower** the value, the **more chance** to be **selected** by the path finding algorithm.
The **higher** the value, the more chance to be **rejected**.

# Winds

Winds are used to model environmental conditions into the scenario.
High Fidelity dynamics components shall take into account wind information for entity drifting and ground speed impact.

Winds can only be defined (and modified) under the Winds panel:




## • Wind Types

Two kind of representations are defined to model winds for a scenario:

- **Area** which is a 3D box of wind cells.
- **Tube** which is a vertical pile of wind cells.

A wind cell is a space location that holds the following data:

**Location**: X,Y,Z position of the cell in the scenario (space)
**Force**: strength of the wind in this cell, in Knots.
**Azimuth**: direction of the wind force in Degrees.
**Temperature**: temperature in Celsius inside this cell.
**Pressure**: pressure in Pascal inside this cell.

## • Toolbar

Wind areas and tubes can then be added into the scenario using the toolbar icons:

 Add a 2/3 wind area on the map.

 Add a wind column at the given mouse position.

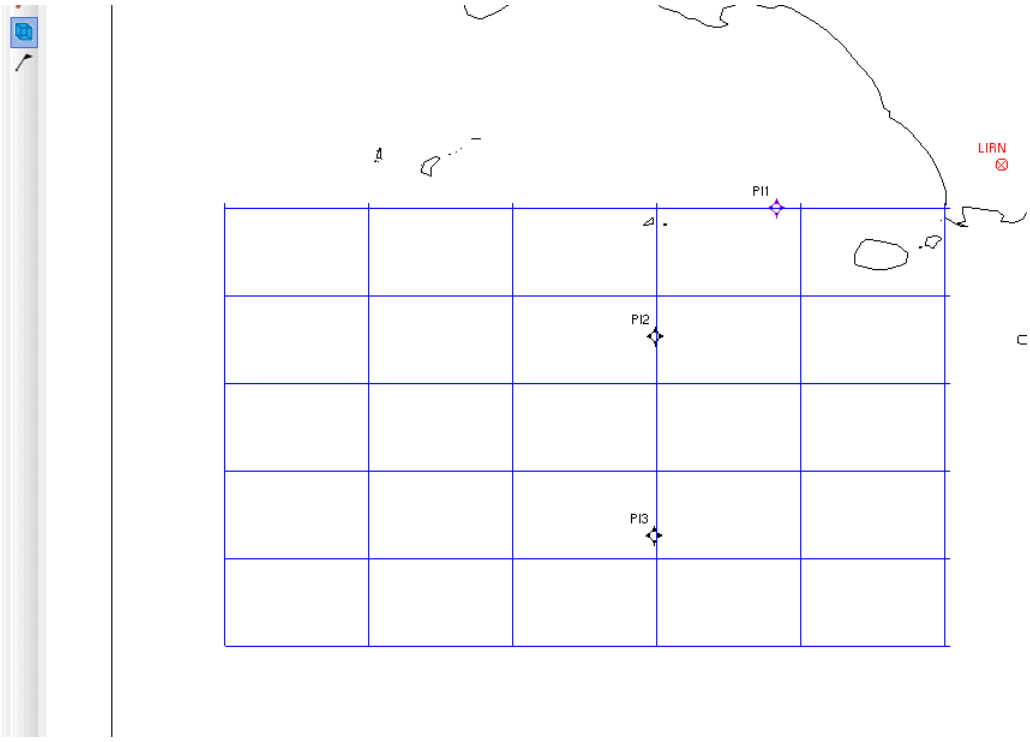
## • Popup Menu

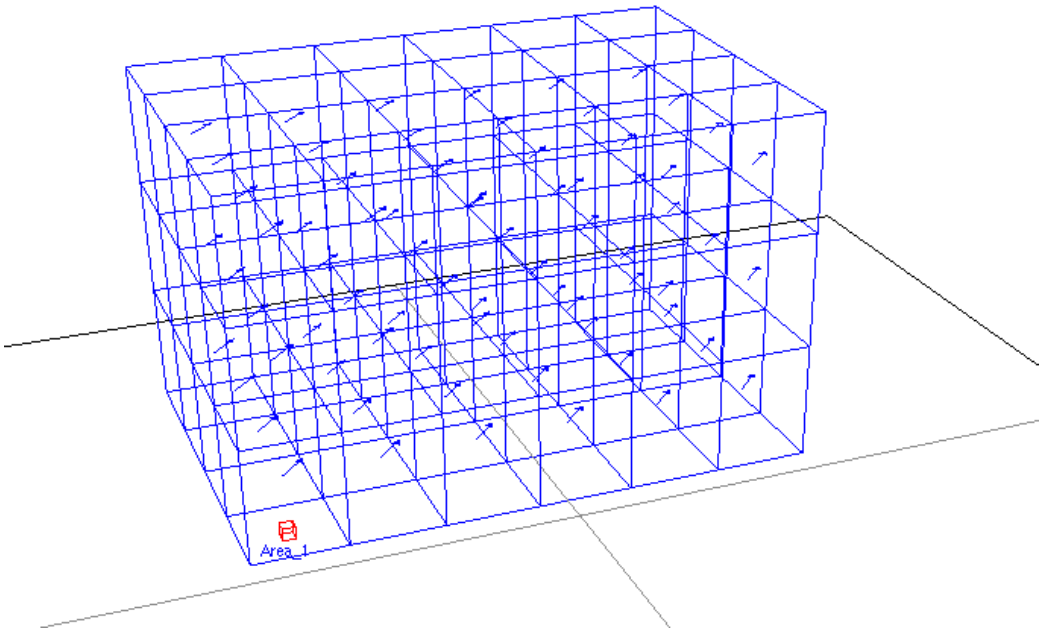
### Winds


**WindArea**: Add a 2/3 wind cube volume on the map

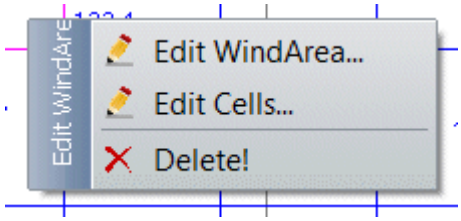
**WindTube**: Add a wind column at the given mouse position

To add a **wind area**, select the icon  (or use the contextual popup menu), mouse changes to  . Draw the area over the scenario using the mouse from the bottom-left corner (holding the left mouse button) to the top-right one.
Then set the properties.






When a wind **area** is **selected**, use the current menu:

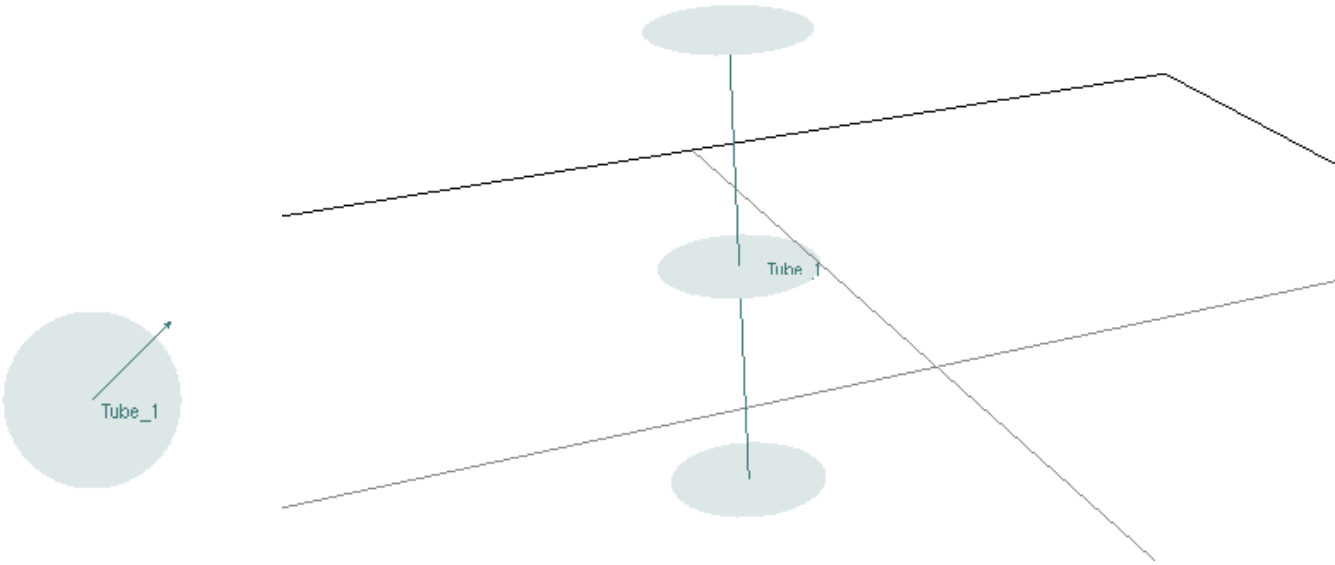



**Edit**: open the area property window.

**Edit Cell**: open the cells property window.

**Delete**: suppress the area from the scenario

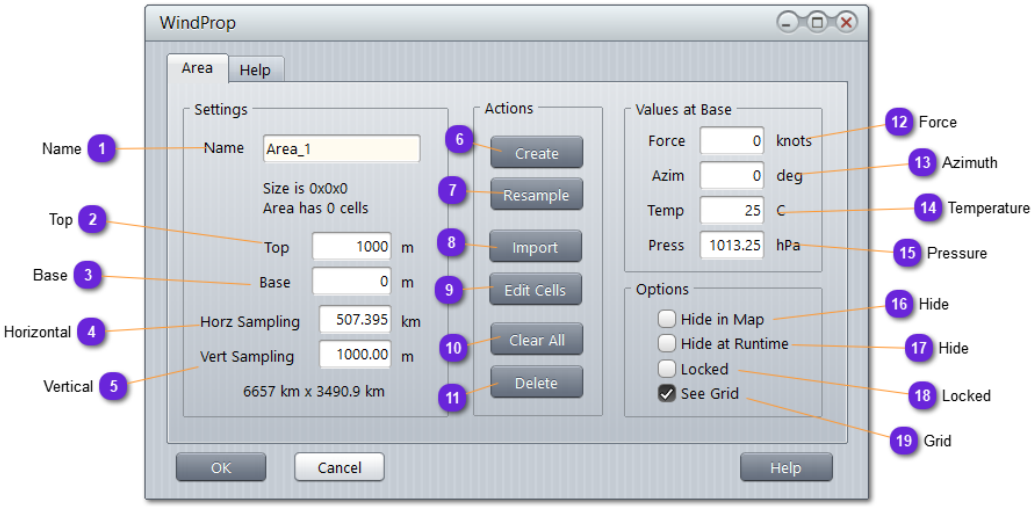
To add a **wind tube**, select the icon  (or use the contextual popup menu) then click on the correct location to position it.
Then set the properties.



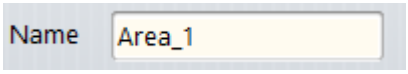

## • How to use

If winds must effect airborne entities position, each of them must have attached a WindEffect component.
Refer to the Developer Guide to learn how to setup this component.

# Wind Area




## 1 Name




**Name** of the area. Also used to be retrieved from the wind manager.

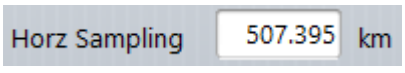
## 2 Top


**Ceiling** altitude of the area bloc, in meters.

## 3 Base

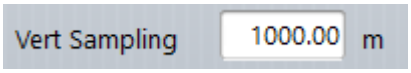

**Bottom** altitude of the area bloc, in meters.

## 4 Horizontal



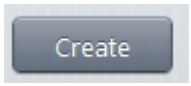

Set here the **width** the wind cell.
The number of rows and columns of cells are automatically computed according to dimensions of the bloc and this information is displayed below the name.

## 5 Vertical



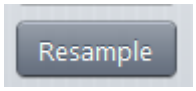

Set here the **height** of a desired wind cell.
The number of storey is automatically computed according to height of the bloc.

## 6 Create




Once the parameters are defined in the Settings box, use this button to physically **create** the wind bloc in memory.

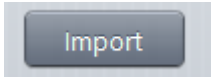
## 7 Resample




**Free** the wind bloc from all wind cells and recreate them according to data in Settings box and sampling values.
Cells will also get the Force, Azim, Temp and Press values.


*Temperature and Pressure will be modified according to altitude.*

## 8 Import

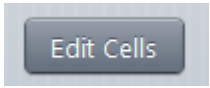



**Create** automatically a wind area from a formatted weather **grid ASCII file**:

```
Lat [deg]      Lon [deg]      Alt [m]      Wind N [m/s]      Wind
E [m/s]      Temp [K]      Pressure [Pa]
```

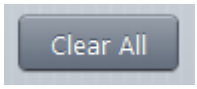
Each entry in the file, all separated with space or tab, represent a wind cell.

## 9 Edit Cells



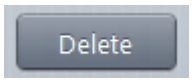

Call the Cells Editor for the selected column of the area (shown in magenta on the Wind Area grid)
If no column is selected, the button is not effective.

## 10 Clear All




**Clean** all data from all wind cells.

## 11 Delete




Delete all wind cells, free the memory and remove the area from the scenario.

## 12 Force


Wind **force** value at **base**. The higher the value, the longer the arrow.

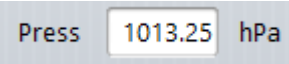
## 13 Azimuth


Wind **azimuth** value in degrees at **base** to give to all the cells of the area. The wind force **arrow** shows the azimuth, up for north, right for east, and down for south.

## 14 Temperature

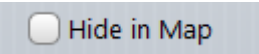

Mean **temperature** at **base** of the area.
Temperature will be automatically adjusted according to altitude gain (6 degrees loss per 1000 m)

## 15 Pressure




Mean **pressure** at **base** of the area.
Pressure will be automatically adjusted according to altitude according to the following formula: `p(z) = 1013.25 * (1-(0.0065*z/288.15)) ^ 5.255`

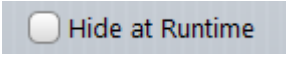
## 16 Hide




If checked, the wind area will not be displayed on the terrain, even if selected.

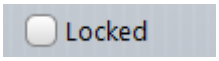
*Wind areas are always displayed when Winds layer is selected.*

## 17 Hide




If checked, the mesh and content will not be displayed on the simulation engine if OpenGL viewer (or equivalent) is selected.
Use this option when Hide in Map (10) is not checked. The mesh will be visible on the GUI map (for info) but not on the runtime window (for performances or other reasons).

## 18 Locked

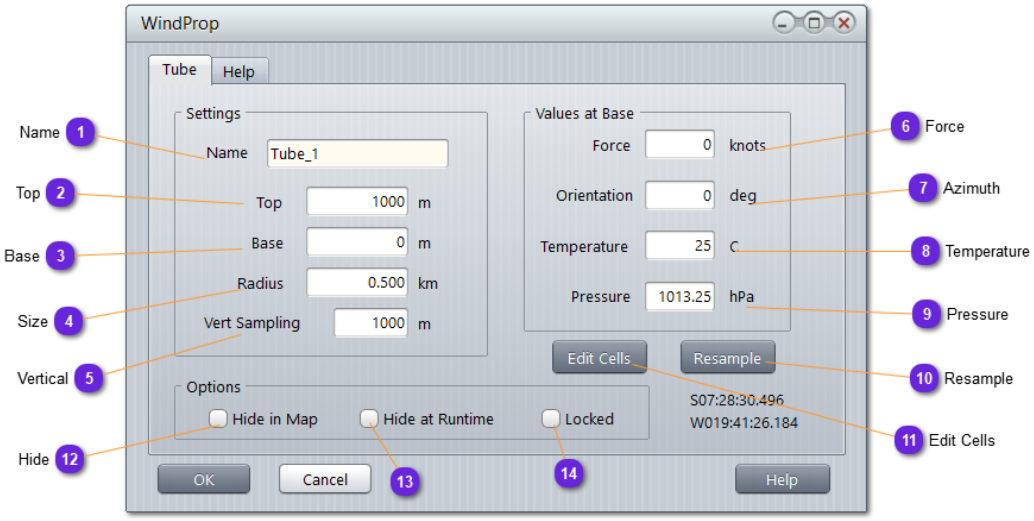



If locked, the area cannot be displaced with the mouse

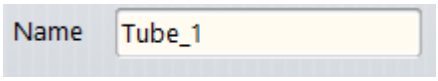
## 19 Grid


When checked, each node is drawn with a square (plan view) or a cube (perspective view).

# Wind Tube




## 1 Name




**Name** of the wind tube

## 2 Top


**Top altitude** of the tube, in meters.

## 3 Base


**Bottom altitude** of the tube, in meters.

## 4 Size


**Radius** of the tube, in kilometers.

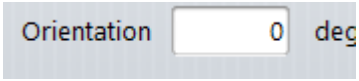
## 5 Vertical


Set here the **height** of a desired wind cell.
The number of slices is automatically computed according to height of the type.

## 6 Force

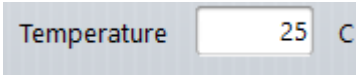

Wind **force** value at **base**. The higher the value, the longer the arrow.

## 7 Azimuth



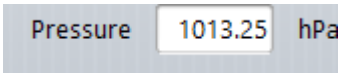

Wind **azimuth** value in degrees at **base**
The wind force **arrow** shows the azimuth, up for north, right for east, and down for south.

## 8 Temperature



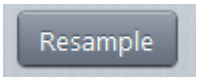

Mean **temperature** at **base** of the tube.
Temperature will be automatically adjusted according to altitude gain (6 degrees loss per 1000 m)

## 9 Pressure




Mean **pressure** at **base** of the area.
Pressure will be automatically adjusted according to altitude according to the following formula: `p(z) = 1013.25 * (1-(0.0065*z/288.15)) ^ 5.255`

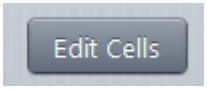
## 10 Resample




**Free** the wind tube from all cells and recreate them according to data in Settings box and sampling values.
Cells will also get the Force, Azim, Temp and Press values.

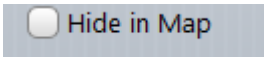

*Temperature and Pressure will be modified according to altitude.*

## 11 Edit Cells




Call the Cells Editor for the selected wind tube.

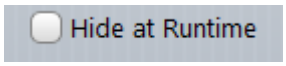
## 12 Hide




If checked, the wind tube will not be displayed on the terrain, even if selected.

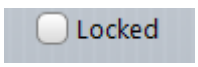

*Wind tubes are always displayed when Winds layer is selected.*

## 13 Hide at Runtime




If checked, the tube and content will not be displayed on the simulation engine if OpenGL viewer (or equivalent) is selected.
Use this option when Hide in Map (10) is not checked. The tube will be visible on the GUI map (for info) but not on the runtime window (for performances or other reasons).

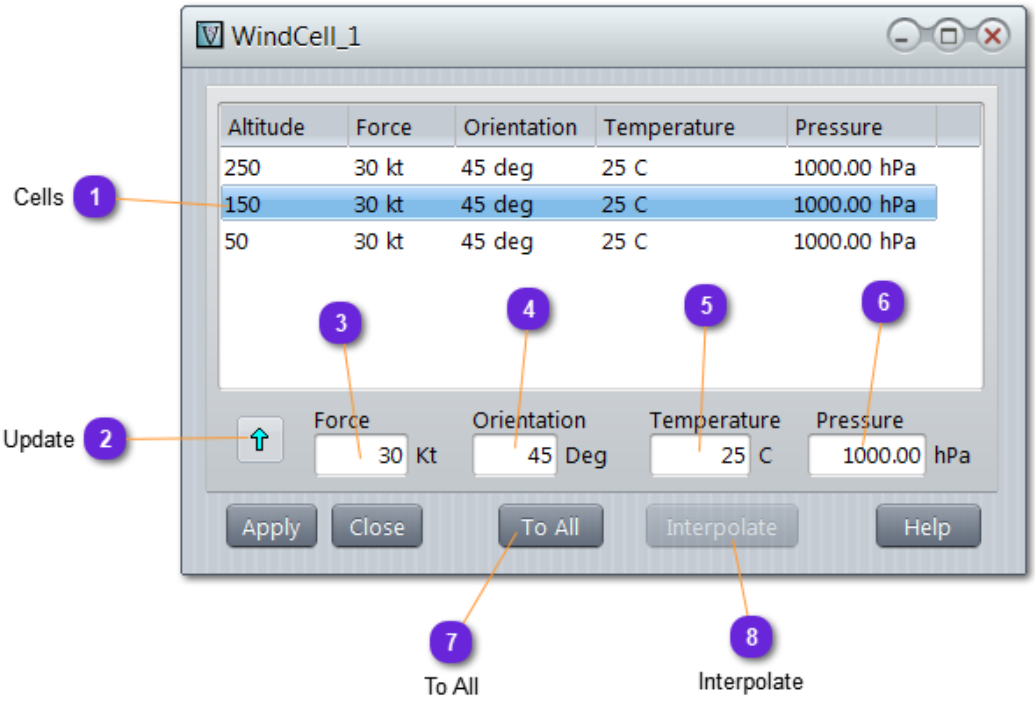
## 14 Locked



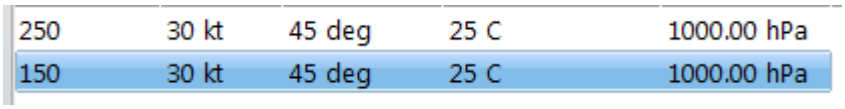

If locked, the tube cannot be displaced with the mouse

# Wind Cells

This window list off wind cells from the same column between base and top of a wind area or tube.




## 1 Cells

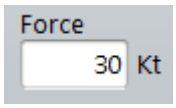



List all the cells on the same (selected) column of the area, from the base (bottom of the list) up to the ceiling (top of the list)

## 2 Update

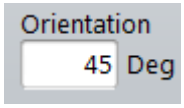
**Overwrite** the selected cell with the **values** of the text fields (Force, Azim, Temp, Press)

## 3 Force



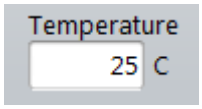

Wind **force** value for the selected cell.

## 4 Azimuth



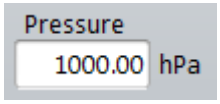

Wind **azimuth** value for the selected cell.

## 5 Temperature



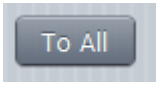

**Temperature** for the selected cell.

## 6 Pressure



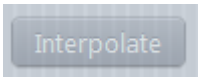

**Pressure** for the selected cell.

## 7 To All




**Overwrite all** wind cells with the text field values.

## 8 Interpolate



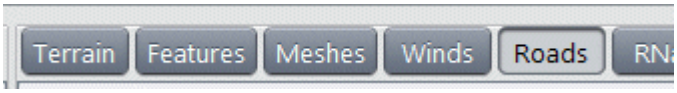

**Interpolate** all values between the selected cells.
Select either all of some cells in the list and click on the button.

# Roads

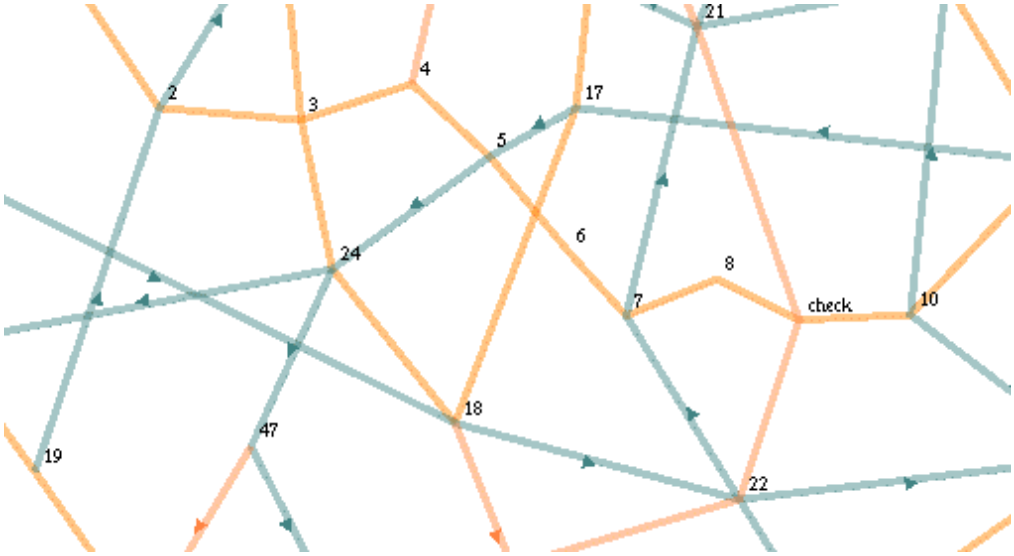
Road network is a functionality to simulate traffic or connections between nodes, along ways, under constraints.
Although roads can be used for something else than vehicle traffic, it is mostly intended for that purpose.
For air traffic, see RNav.
Maritime routes is not covered yet.

A **road** is made of strips and points.
A point can either be a road point joining two strips, or a junction point when another road is crossing.

Roads can only be defined (and modified) under the Roads panel:



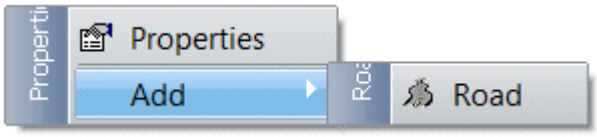

**Components** used with road network:
RoadFinding: use the **Dijkstra** algorithm to find the shortest itinerary in a graph
RoadFollow: make an entity follow an itinerary found by RoadFinding component.




## • Toolbar

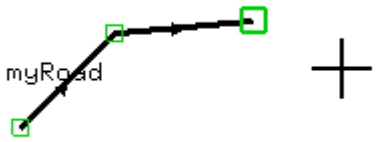
 Add a **new** road on the scenario.

# • Popup Menus

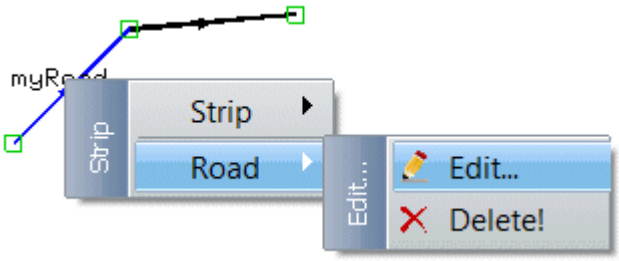



**Add Road**: create a road on the scenario.

When **adding** a **new** road, enter the intended name of the road.
Cursor will change to +
Click on the map to add all road points, from the first to the last one.
When finish, use the mouse right click and select Done.



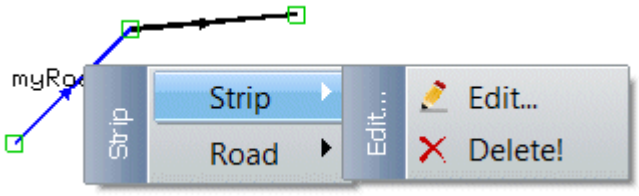

To **edit** a **road**, just double click on it or select it and use the Road menu:




**Road Edit**: popup the road property window.

**Road Delete**: remove/suppress the whole road.

To **edit** a **strip** (lane between to points), use the strip menu:

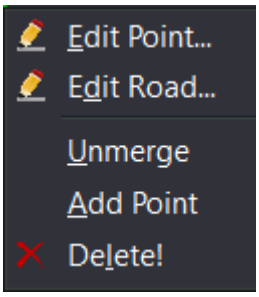



**Strip Edit**: popup the strip property window.

**Strip Delete**: remove/suppress the strip. Road will be disconnected.

To **edit** a **point** (strip extremity), use the point menu:

### Roads




**Edit Point**: popup the point property window.

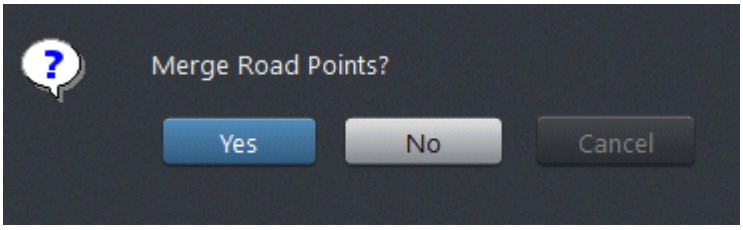
**Edit Road**: popup the road property window.

**Unmerge**: when a point is an Intersection, will create two (or more) separate points. Roads no longer intersect.

**Add Point**: cursor will change to +. Clicking on the map will add a new point at the mouse location. A strip will be created to join the selected point with the new one.

**Delete**: remove the point and the connecting strips.

To **add** a **junction** or to **create** an **intersection** between roads, select a road point with the mouse and **drag it over another point** of any road.
The cursor will stick to the anchoring point.
Release mouse button. A message will ask permission to merge the two points.

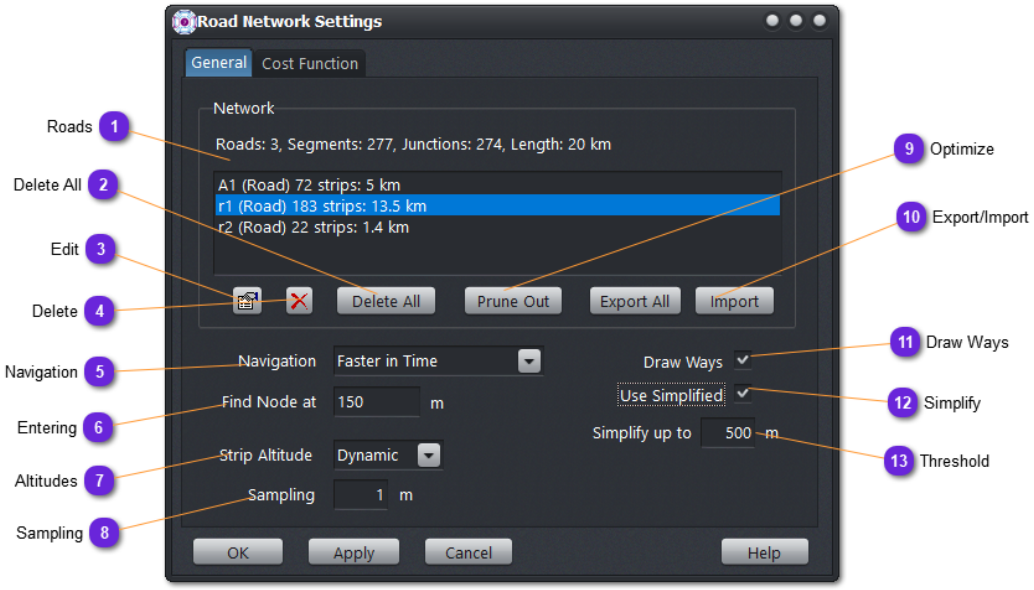



Choose **Yes**.
A junction point (intersection) is then created and drawn with a brown color:

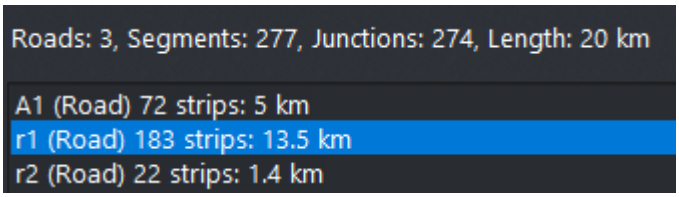
To unmerge an intersection point, select it, right click and select the **Unmerge** option.


*Road strips cannot make angle less than 45 degrees. Position will be refused. Try to put more* `RoadPoints` *to round the road and* ***avoid sharp angles****.*

# Road Network Properties

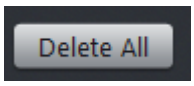



## 1 Roads




List of all the **roads** defined in the network.

## 2 Delete All




Delete/**remove all** roads of the network (and the network itself).

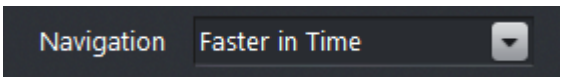
## 3 Edit

Display the property window of the selected road (of the list).

## 4 Delete

Delete/**remove** the selected road (of the list).

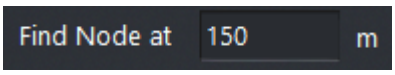
## 5 Navigation




Select the navigation **mode** that must follow the RoadFinding component:

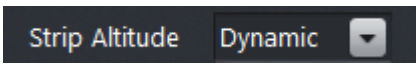
- Faster in Time: select the roads that minimize the time to destination.
- Shorter in Distance: select the roads that minimize the distance to destination.
- Faster with Traffic: select the roads that minimize the time to destination, including the current traffic.

## 6 Entering



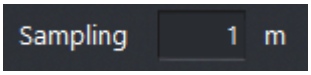

Used by the RoadFinding component.
When a destination point is set, try to **enter** the **network** using the closest point in the specified radius.

## 7 Altitudes



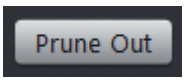

By default, this option is set to None.
When set to Dynamic, at runtime, an altitude vector will be created for each strip; terrain will be queried for each position in the vector. In the simulation code, the strip method `getAltAt(WCoord)` will return the value stored in the vector for the given position. This is useful when altitude queries are CPU consuming and when altitudes do not change.
When set to Cache, altitudes will be computed for all strips of all ground base roads and stored in a file named {runtime_name}.alts stored on the same directory of the runtime database (.rt) and runtime engine (.exe). If cache file exists, will be used (faster load). To force the regeneration of the cache, just delete the file.

## 8 Sampling




When **Strip Altitude** option is set to Dynamic, the value stands for the distance between two altitude queries along a strip. The higher the value (in meters), the less accuracy; the lower the value, the better the accuracy, but initialization time might be longer and memory consumption higher, in case of many strips or big network area.

## 9 Optimize

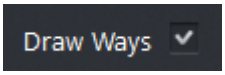



Use this button to remove from the database all roads falling entirely outside the actual terrain boundaries.
When the network is big, it can be useful to concentrate on one specific area to reduce the computation load when searching for a path.
This cannot be undo.

## 10 Export/Import

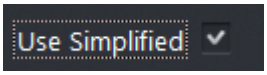

You can export the actual route network (all routes) to a text file, for sharing with other database or backup.
The import button allow to add an exported network to the actual one. If roads with same names are found, they will be updated.

## 11 Draw Ways



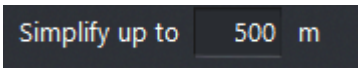

When checks, the lane strips **directions** (one or both ways) are drawn on the map using **arrows**.

## 12 Simplify



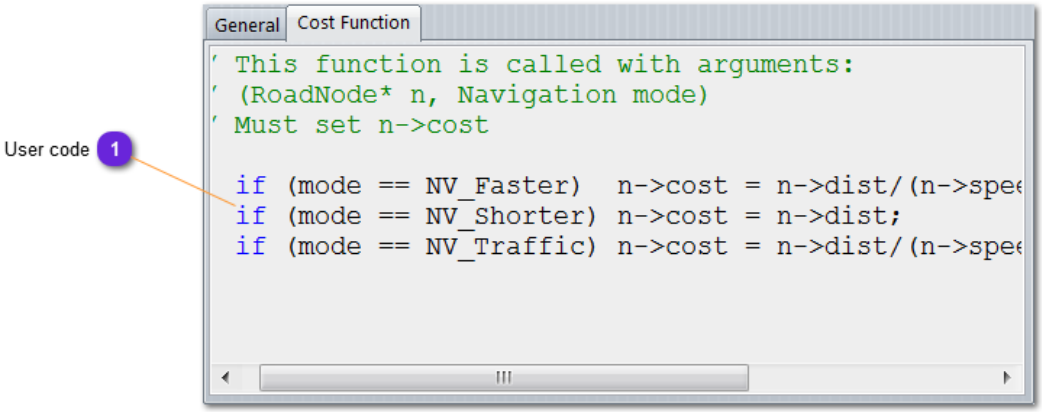

When checked, too close points are **not drawn**. This mode improves the drawing speed but reduce the smoothness of some road curves.
If unchecked, all road points are drawn, even if they are numerous, mostly on curves.

## 13 Threshold




For Show Simplified mode (9), set the **minimal** distance (in meters) between two road points for both to be displayed.

# Cost Function



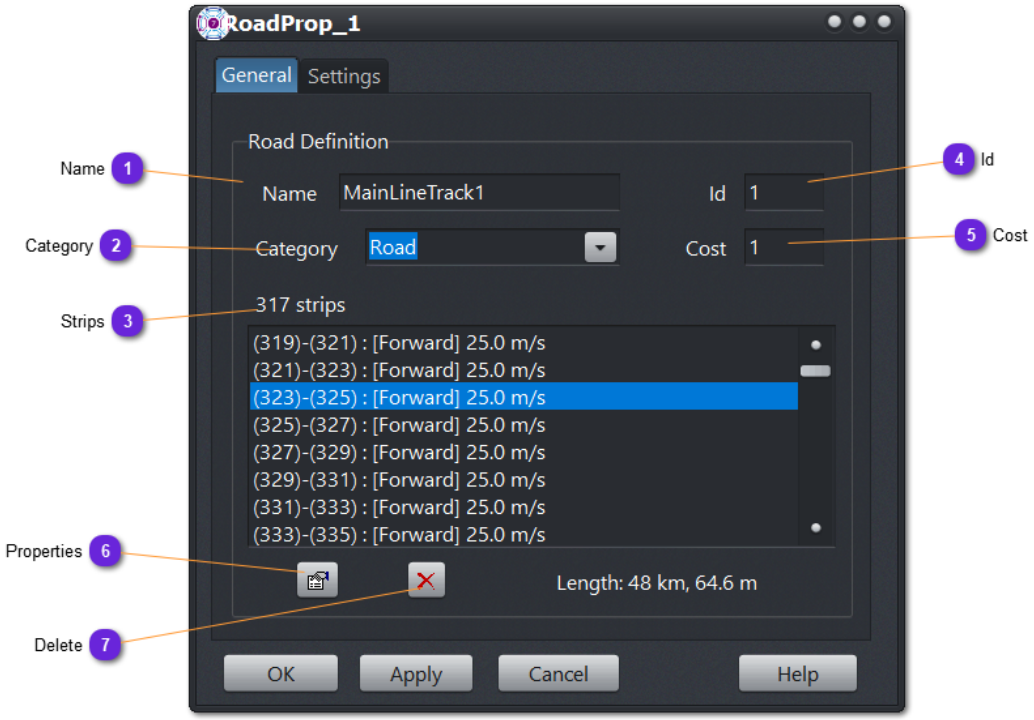

## 1 User code

```
if (mode == NV_Shorter) n->cost = n->dist;
```

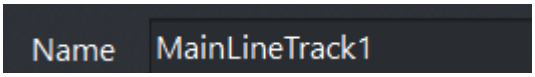
This function is used to return the cost for selecting a specific strip lane.
The `RoadData` structure given to the function can be seen at /include/engine/vt_roads.h

This function shall return any arbitrary value, according to any kind of conditions extracted from the strip of any other data structure.
What is important is not the value itself but its ranking regarding the other node values for one road search (per entity).
The function must return the cost for using the lane `n` to be used by the Dijkstra algorithm.
The **lower** the value, the **more chance** to be **selected** by the road finding algorithm. The **higher** the value, the more chance to be **rejected**.

# Road properties

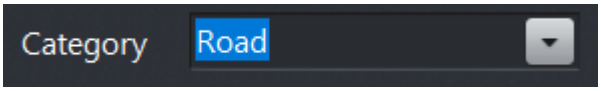



## 1 Name




**Name** of the road.

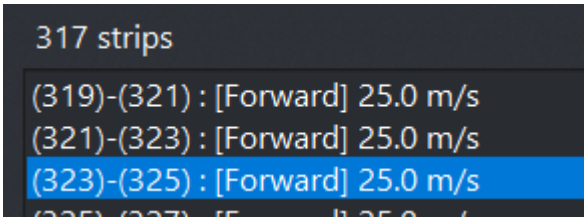
## 2 Category




Specify the type of the strip:

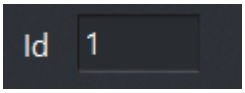
- Street
- Road
- Highway
- Ferry
- Route
- Taxiway
- Rail

## 3 Strips



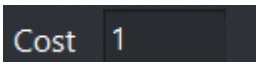

List of all **strips** of the road.

## 4 Id



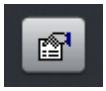

Unique **id** of the road.

## 5 Cost




Arbitrary road cost for automatic routing computation. In determining a route, the higher the cost of a section, the less likely it is to be selected.

## 6 Strip

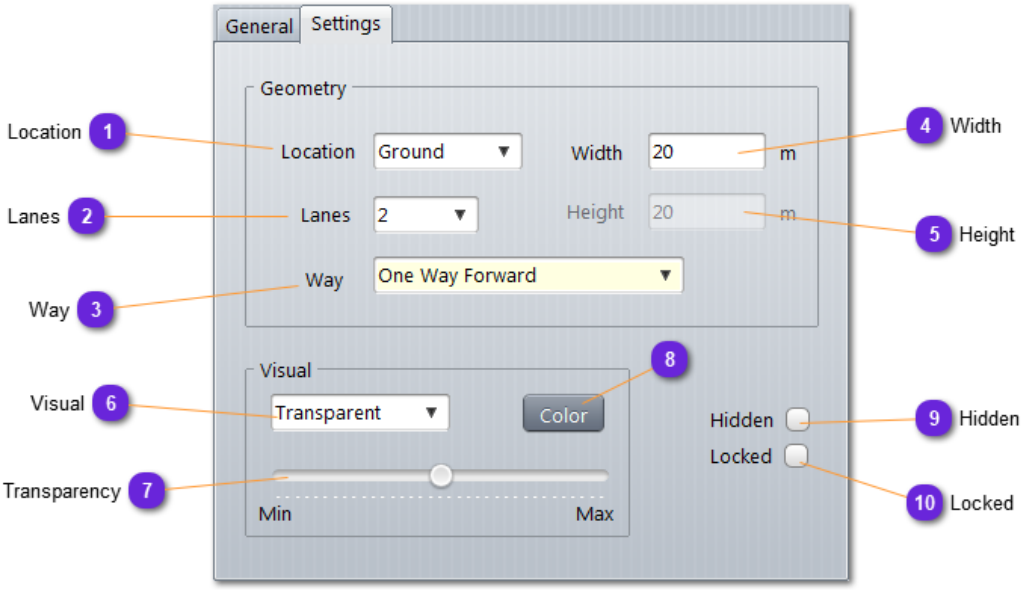


Call the **strip** property window for the selected one in the list.

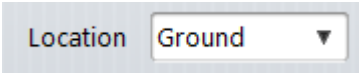
## 7 Delete

Remove/**suppress** the selected strip of the road.

# Road Settings



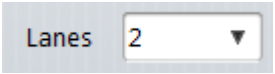

## 1 Location




Specify if the road is on the ground or in the air.

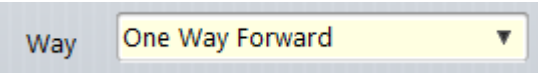
- **Ground**: if selected, the terrain altitude will be used for all points of the strip road.
- **Air**: user will need to specify altitude for each strip points.

## 2 Lanes




Number of **lanes** for the road, from 1 to 5.

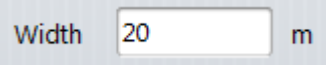
## 3 Way




Specify the orientation of the traffic on the road:

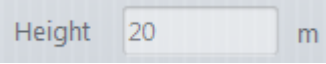
- One way Forward: all lanes are used forward from first point to last.
- One way Backward: all lanes are used backward from last to first.
- Two ways: half of the lanes are used forward and the other half used backward.
- No way: cannot pass whatever the direction.

## 4 Width



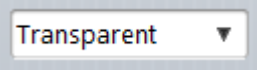

**Width** of the road (including all lanes).

## 5 Height




For air road only, **altitude** of the road.

## 6 Visual



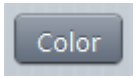

Define the **drawing** mode of the road (strips):

- Outline: borders of the strips only
- Opaque: strips are filled with the selected color (8)
- Transparent: strips are filled with a transparent (9) selected color (8)

## 7 Transparency

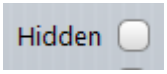
Set the **transparency** value of the color used to fill the road strips.
Min for **opaque** and Max for **glass** (invisible)

## 8 Color




**Color** used to draw the road strips.

## 9 Hidden

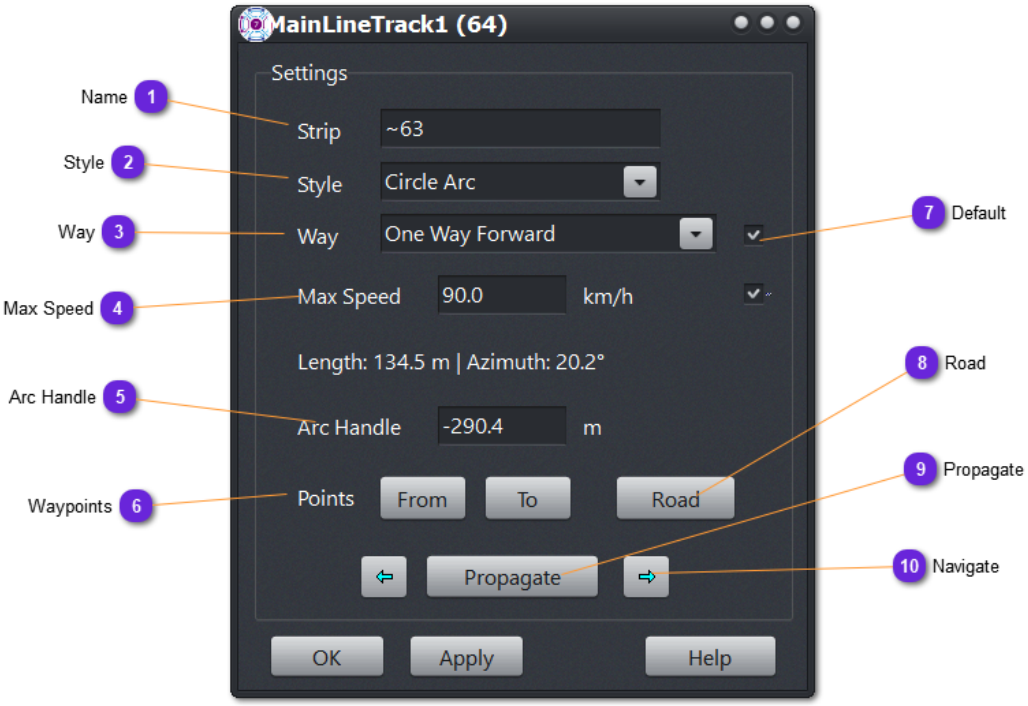



When checked, the road is **not drawn** on the map.

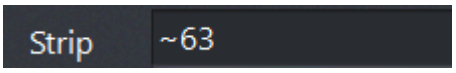
## 10 Locked


If checked, the road **cannot** be **modified**. All points are locked including altitude.

# Strip Properties

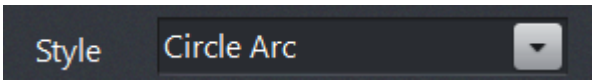



## 1 Name




Optional **name** of the strip. Automatically drawn.

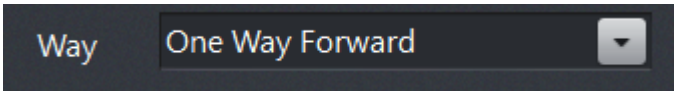
## 2 Style




Specify the type of segment between the From and To points:

- Straight Line
- Circle Arc

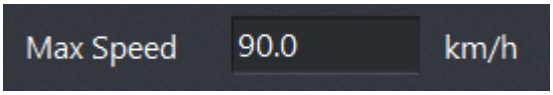
## 3 Way




Specify the orientation of the traffic on the strip:

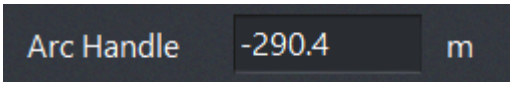
- One way Forward: all lanes are used forward from first point to last.
- One way Backward: all lanes are used backward from last to first.
- Two ways: half of the lanes are used forward and the other half used backward.
- No way: cannot pass whatever the direction.

## 4 Max Speed



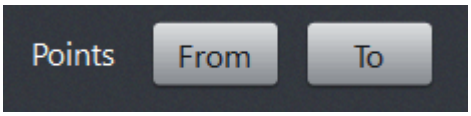

Set the maximum speed allowed on the strip.

## 5 Arc Handle




When the style is set to Circle Arc, this value is used to move away the handle defining the curvature of the arc between the two points. The higher the value, the flatter the arc.
Use negative value to change the arc way.
Press **Apply** button to see the result on the map.

## 6 Waypoints

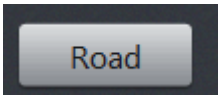



Display the property window of each extremity point of the strip.

## 7 Default

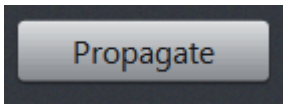
Check this value as **Default** for the Propagate button (9).

## 8 Road




Open the Road property window the Strip belongs to.

## 9 Propagate

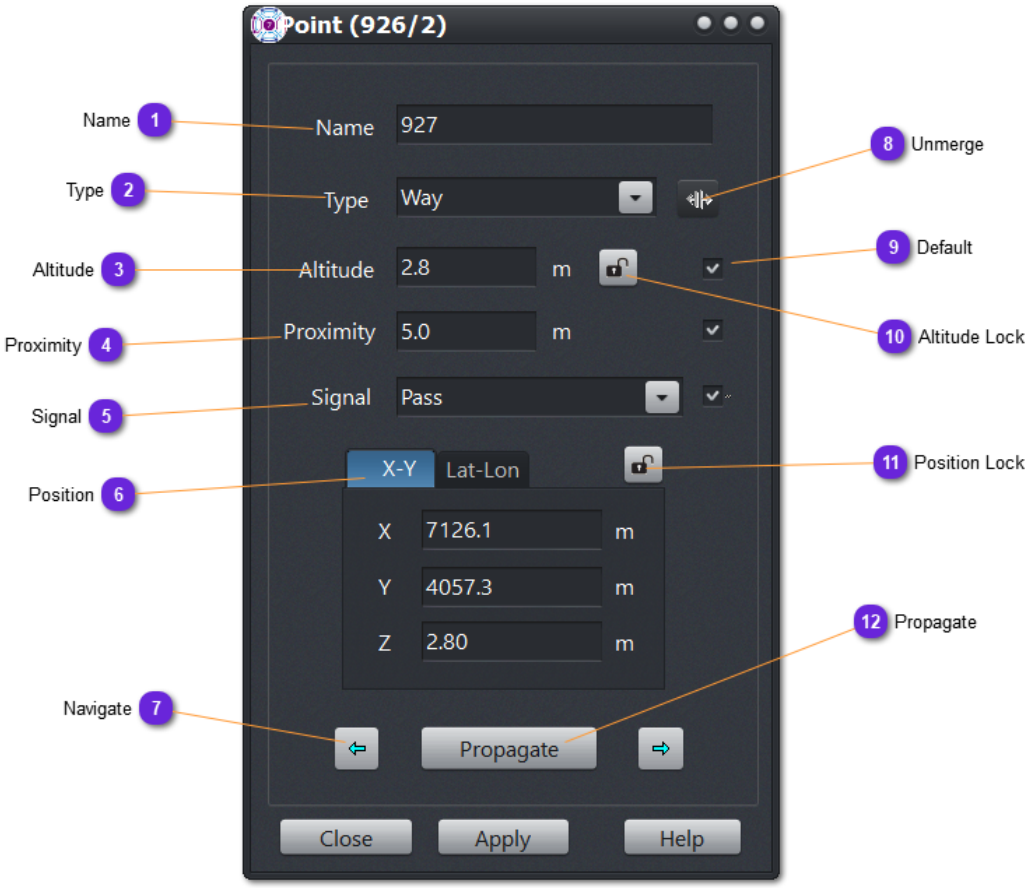



Will **propagate** to all strips of the road the values checked as **Default** (7)

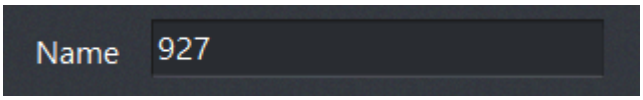
## 10 Navigate

Use these buttons (right and left) to **navigate** the road from one strip to the next one successively, **forward** and **backward**.

# Point Properties

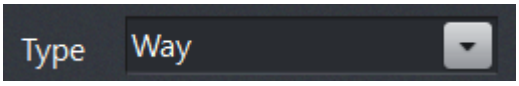



## 1 Name




**Name** of the point.

## 2 Type

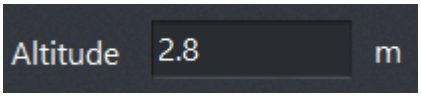



Display the **type** of the point:

- Intersect: intersection point of more than 3 strips.
- Way: connecting point of two strips or extremity point of a road.

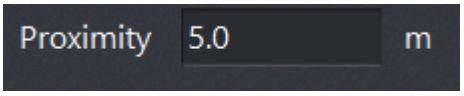

*This field is read only.*

## 3 Altitude



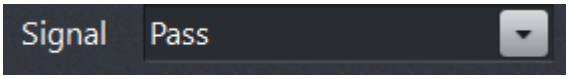

**Altitude** of the point.
If the road is grounded, will be altitude of the terrain below the point.

## 4 Proximity




Used by the RoadFollow component.
A point is considered passed or reached at the following distance.

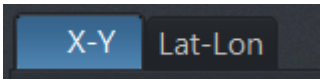
## 5 Signal




Used by the RoadFollow component:

- Type of the signal or behavior for a point/junction.
- Pass: default value for all point that are not intersection.
- Stop: entity will stop and wait for other strips to be cleared before continuing the itinerary.
- Yield: entity will yield the way on the other strips before continuing.
- Crossing: right priority will apply.
- Traffic-Light: *not implemented yet.*

## 6 Position




**Coordinates** of the point in XY or Lat-Lon.

## 7 Navigate

Use these buttons (right and left) to **navigate** the road from one point to the next one successively, **forward** and **backward**.

## 8 Unmerge

If the point is used in a junction, pressing this button will isolate it and create intermediate points for all shared strips

## 9 Default

Check this to make the data **default** one for Propagate function.

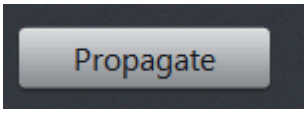
## 10 Altitude Lock

Lock/Unlock the altitude. Useful for ground roads to prevent automatic change of the value. Once a road (or individual point) is locked, the altitude of all its points will remain unchanged.

## 11 Position Lock

Lock/Unlock the position. Useful to prevent accidental move of a road point. Once a road (or individual point) is locked, the position of all its points won't be modifiable.

## 12 Propagate




Will **propagate** to all points of the road the values checked as default (9)

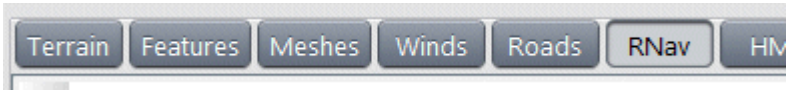
# RNav

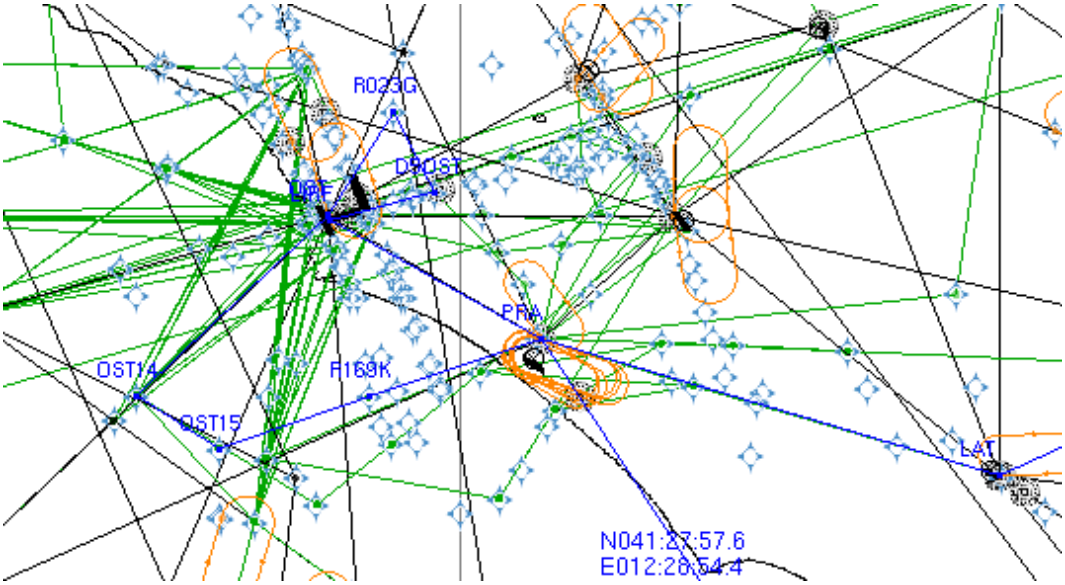
RNav (Area Navigation) is a database layer that provides airborne navigation records (items) for scenarios to do ATC simulation using ARINC 424 standardized data.
vsTASKER also provides capabilities to update the database and create own records, equipments with user defined data.

When loaded, RNav database can be displayed on the map.
See RNav in Navigation environment to see how to use Flight-Plans for making entities navigating in the network.

**Component** using RNav: FlightPlan

RNav can only be defined (and modified) under the RNav panel:




## • Toolbar

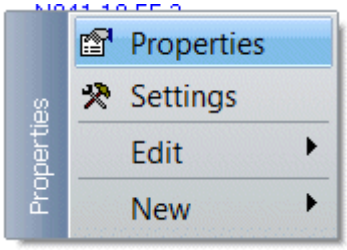
**A** Add an **airport** (record) on the database

**W** Add a **waypoint** (record) on the database

**N** Add a **navaid** (record) on the database

**H** Add an **holding** (record) point on the database

## • Popup Menu

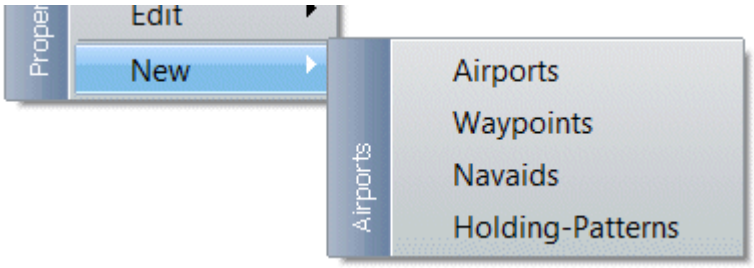



**Properties**: open the RNav database property window.

**Settings**: open the settings window.

**Edit**: select which RNav record type to list (can be changed from the drop down menu there)

All new records added to the database will be stored aside from the ARINC RNav database loaded by vsTASKER.
They will belong to the current scenario database until merged (see Save option here)

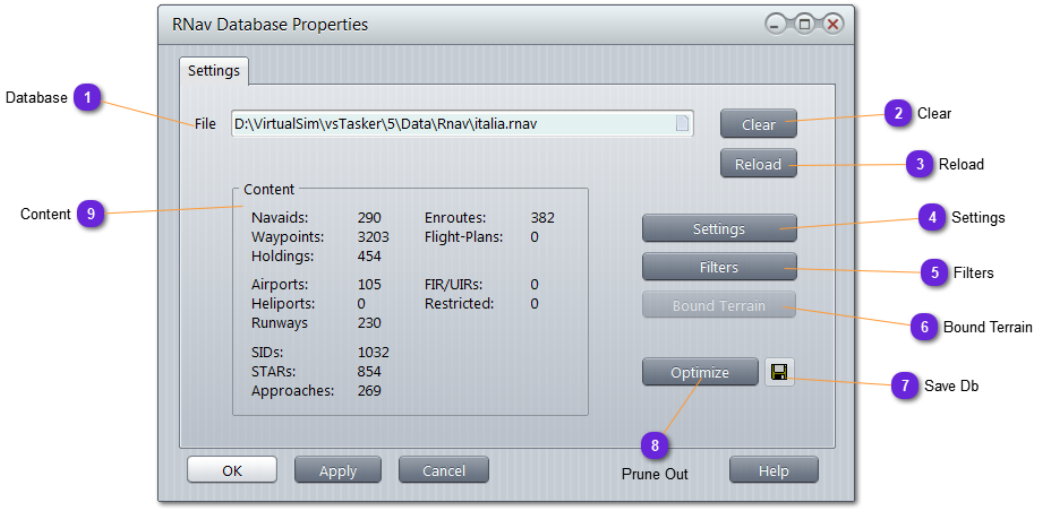



**Airports**:
**Waypoints**:
**Navaids**:
**Holding-Patterns**: Toolbar shortcut (see above)

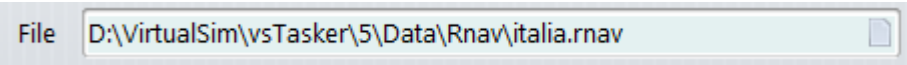

**Edit**: call the record property window for the RNav element selected.

**Delete**: Remove from the database the selected element.

# RNav Database

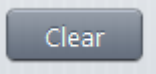



## 1 Database



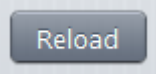

Proprietary database **file** produced by the TerrainBuilder tool to be used for the RNav navigation.
The name can be changed by the user to either duplicate this database or to save the optimized database.

## 2 Clear



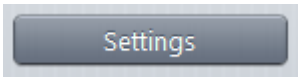

**Free** the memory from the RNav database.
All records are cleared, including user added data.

## 3 Reload



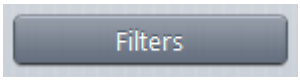

Clear and **load again** the database file (1).
This way clears all the user added records.

## 4 Settings



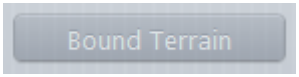

Call the Settings property window.

## 5 Filters




Call the Filters property window.

## 6 Bound Terrain



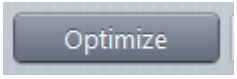

*Not available yet.*

## 7 Save Db

Use this button to overwrite the actual ARINC RNav database with the content of the current RNav content. If no action has been conducted on the RNav content, the database will remain unchanged. If some RNav items (waypoints, navaids, etc.) have been modified or removed, then the RNav database will be modified accordingly.
This should be done after optimization (prune out).
If some RNav items have been manually added, question about merging them with the database will be asked. If answer is yes, then the added items will no more be tagged "user" and will belong to the RNav database.


*The save process will use the file specified in File text field above. Change it in order to create another database and not overwrite the current one.*

## 8 Prune Out

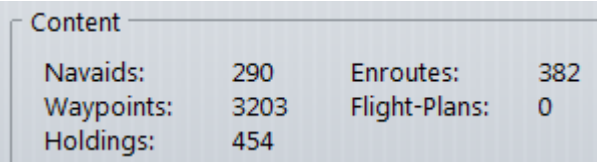



This function will remove from the memory all RNav records outside the terrains boundaries.

Using the button will rebuild the database file of (1) once the optimization is done.

Optimizing should be used to drastically reduce the amount of records loaded in memory when a scenario is concentrating only on a small area of the ARINC database. For example, using the europe.rnav database with a scenario doing ATC around Paris requires to much of memory. It is a good idea to optimize the database, rename the database as paris.rnav and save the optimized database under this new file.

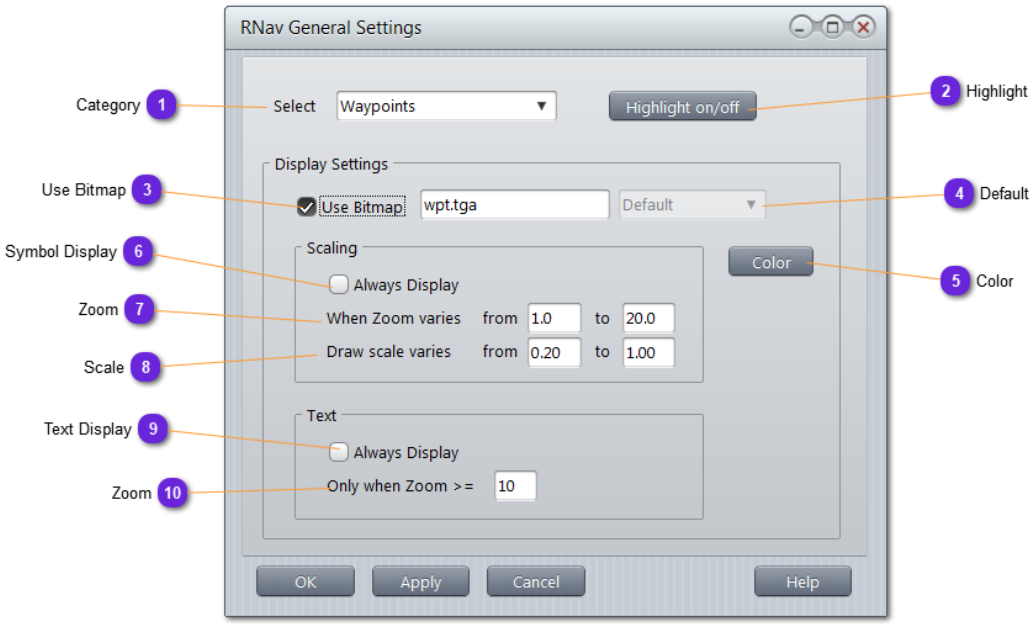
## 9 Content



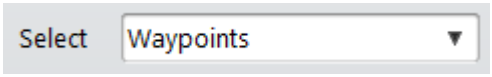

Detail of the loaded RNav database.

# RNav Settings

This window is used to control the drawing of all RNav items (records) on the terrain map.
They are organized by categories. It is not possible to set any particular record, only a category.

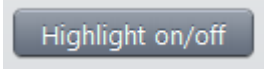



## 1 Category



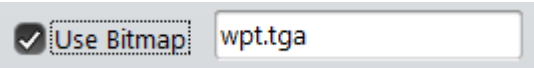

Select which **category** of RNav item to set.

## 2 Highlight



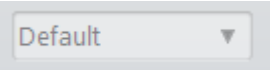

Depress to **force display** on the map all records of the selected category, regardless of the (6) and (9) settings. Depress again to revert to normal drawing.

## 3 Use Bitmap




If checked, the bitmap mentioned in the text field will be used instead of the predefined drawing (4).

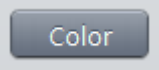
## 4 Default




If Use Bitmap (3) is unchecked, use this drop list to select the predefined symbol to use for the selected category:

- Default: vsTASKER will automatically select the standard drawing or raster for the category
- Dot:
- Circle:
- Reticule:
- Square:
- Triangle Up:
- Triangle Down:
- Diamond:

## 5 Color

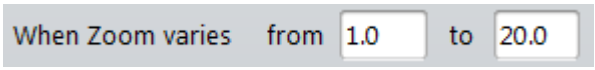



Open the color selection box to select the color to use for text and symbol. No effect on bitmaps.

## 6 Symbol Display

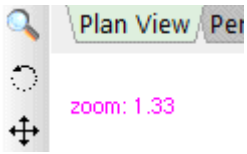

When checked, category will be displayed whatever the zoom level.
If unchecked, display of the bitmaps/symbols depends on the zoom level (automatic).

## 7 Zoom

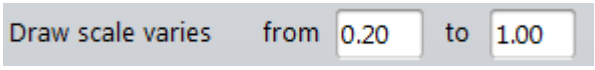



If automatic display (6), the category will be displayed if terrain zoom is between the from and to boundaries.

*According to the scenario size, it might be useful to adjust the boundaries according to the real zoom and the density of the category.*



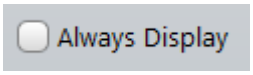

## 8 Scale




Set here the **scale** of the drawing (bitmap or symbol) regarding the zoom level.
Value in text fields are between [0..1] and shall be compared with the zoom level of line above (7).
In the sample above, for a zoom value up to 1, scale will be 0.2 and for zoom above 20, scale will be 1.

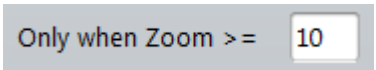
*Linear interpolation is applied between bounds.*

## 9 Text Display




When checked, item label will be displayed whatever the zoom level.
If unchecked, display of the label depends on the zoom level (automatic).

## 10 Zoom




If automatic display is set (9), when zoom is above the text field value, the item label is displayed. Hidden when the zoom level is below.

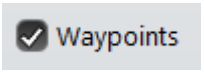
# RNAV Filters

Use this window to filter in and out various category of records to use and display in the database.
Although all records from all supported categories are loaded into the memory, only the selected one will be used.


## 1 Navaids

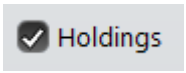

Select **navaid** records in the database.

## 2 Waypoints




Select **waypoint** records in the database.

## 3 Holdings

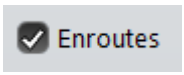



Select **holding** records in the database.

## 4 Markers

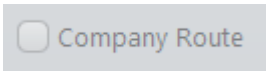

*Not available yet*

## 5 Enroutes



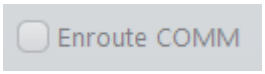

Select **enroute** records in the database.

## 6 Company Route



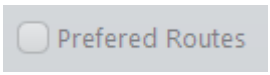

*Not available yet*

## 7 Enroute COMM



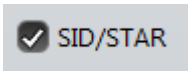

*Not available yet*

## 8 Prefered Routes



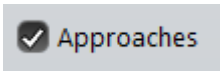

*Not available yet*

## 9 SID/STAR



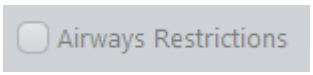

Select **SID** & **STAR** records in the database.

## 10 Approaches



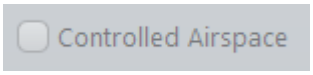

Select **approach** records in the database.

## 11 Airways Restrictions



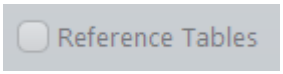

*Not available yet*

## 12 Controlled Airspace



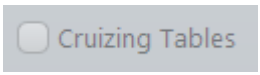

*Not available yet*

## 13 Reference Tables



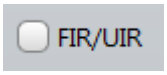

*Not available yet*

## 14 Cruizing Tables




*Not available yet*

## 15 FIR/UIR

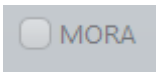



Select **FIR** & **UIR** records in the database.

## 16 Restrictive Airspace

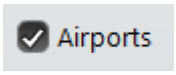

Select **restrictive airspace** records in the database.

## 17 MORA




*Not available yet*

## 18 Airports

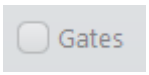



Select **airport** records in the database.

## 19 Runways

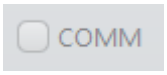

Select **runway** records in the database.

## 20 Gates



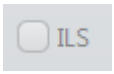

*Not available yet*

## 21 COMM



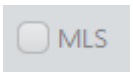

*Not available yet*

## 22 ILS



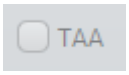

*Not available yet*

## 23 MLS



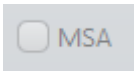

*Not available yet*

## 24 TAA



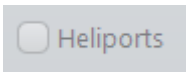

*Not available yet*

## 25 MSA




*Not available yet*

## 26 Heliports

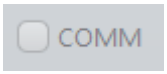



*Not available yet*

## 27 Terminal Waypoint

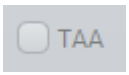

*Not available yet*

## 28 COMM



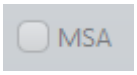

*Not available yet*

## 29 TAA



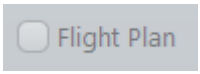

*Not available yet*

## 30 MSA



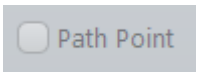

*Not available yet*

## 31 Flight Plan



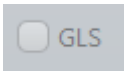

*Not available yet*

## 32 Path Point



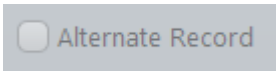

*Not available yet*

## 33 GLS




*Not available yet*

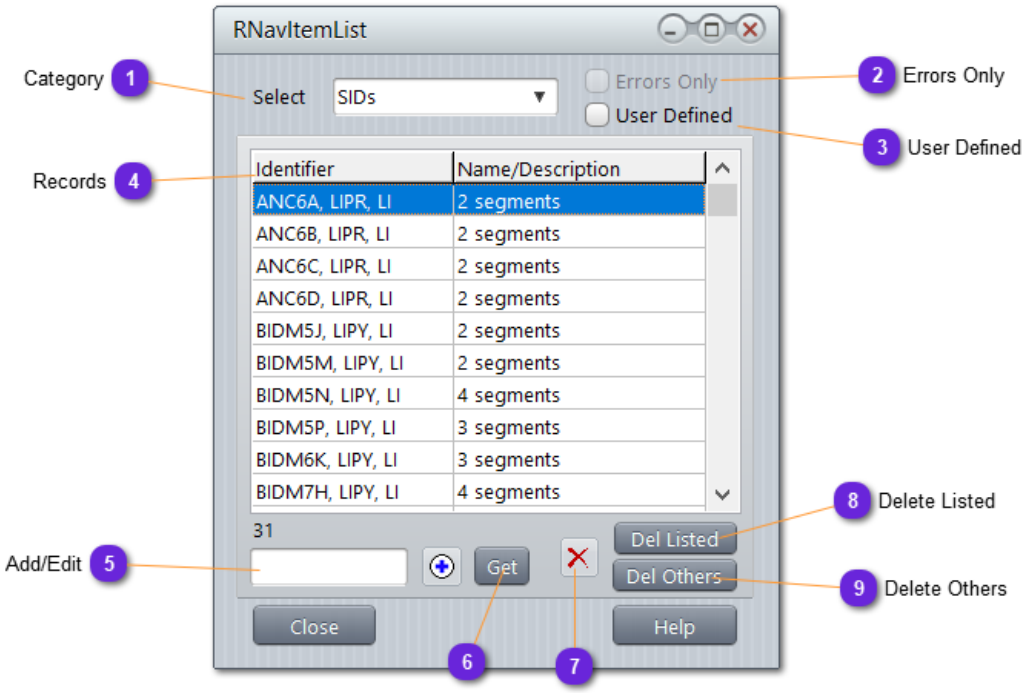
## 34 Alternate Record



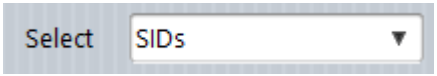

*Not available yet*

# RNav Record List

Allow exploring all RNav records per category.

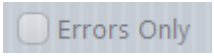



## 1 Category




Select the category to list.

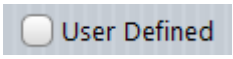
## 2 Errors Only




**Filter** only the records with errors.

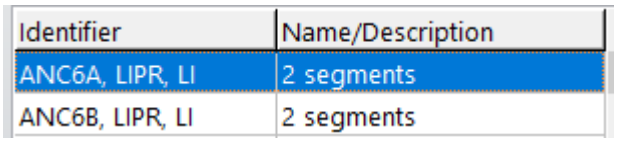

*Not available yet.*

## 3 User Defined




**Filter** only the user added records (not belonging to the database but added on the scenario as RNav items) using toolbar or Add popup menu.

## 4 Records

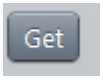



**List** all records of the selected category.
Select one record with the mouse either to delete it (if user added) or to view/ edit details.

## 5 Add/Edit

Use this field to change or add a new record in this category.
Use the button + to add it.

## 6 Get

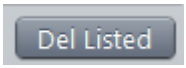



**Open** the property window of the selected record or retrieve the record mentioned in the (5) text field.

## 7 Delete

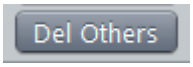
Remove/**suppress** the selected record. Not available for RNav original database records.

## 8 Delete Listed




Remove/suppress all listed items of this window. This is useful combined with filtering (5) or using Errors Only or User Defined flags.

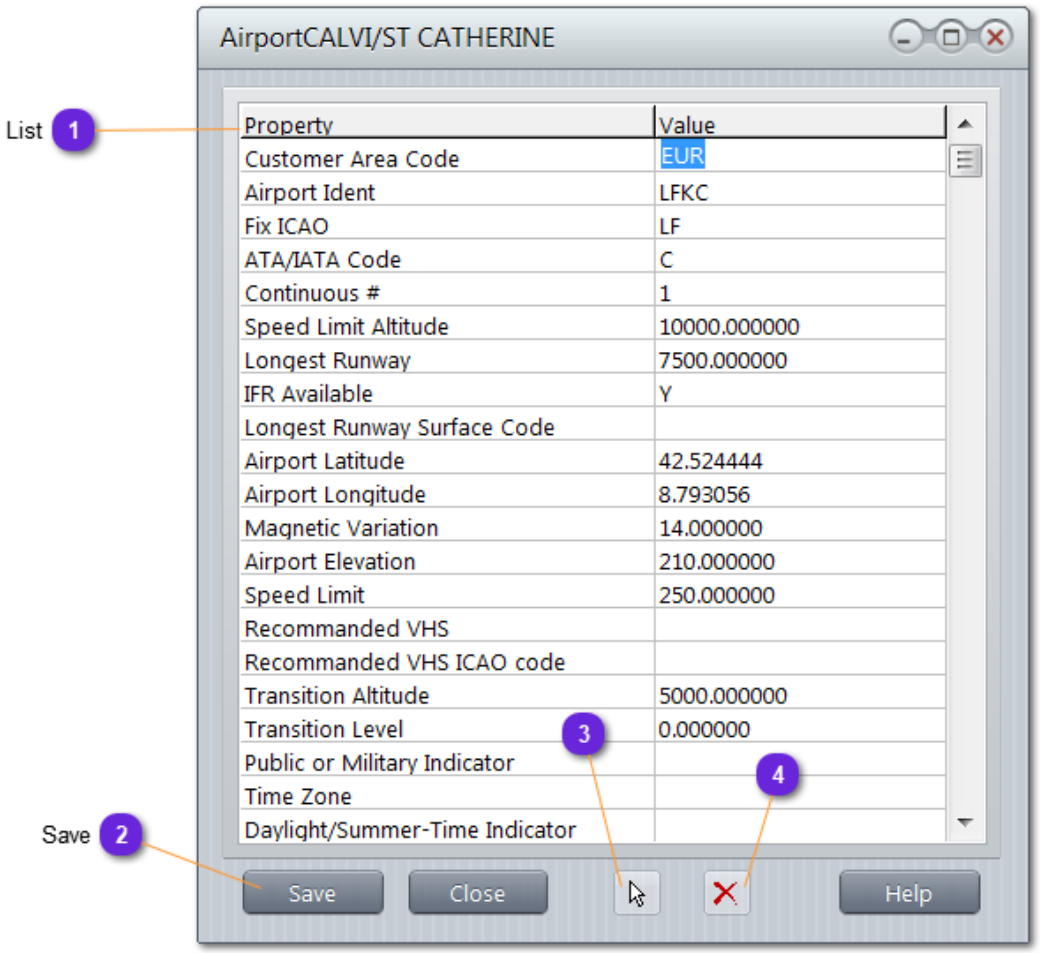
## 9 Delete Others



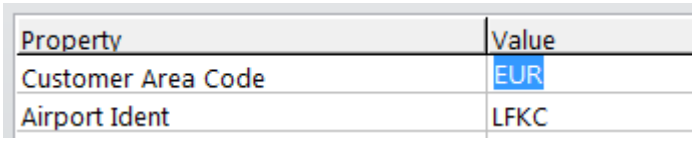

Remove/suppress all items of this category (1) which are **not** listed in the windows. This is useful combined with filtering (5) when it is important to get rid of everything but what has been filtered in.

# RNav Record

Allow reviewing and editing (if allowed) all value entries of a particular record.

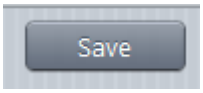



## 1 List




List all the **entries** of the selected RNav record (item). They are following the ARINC standard per category.
Name of the record is included in the window title.
Select any of the Value field to update the content.

## 2 Save




If some values have been changed or updated, use this button to update the record in the database.


*Available only for used added records.*

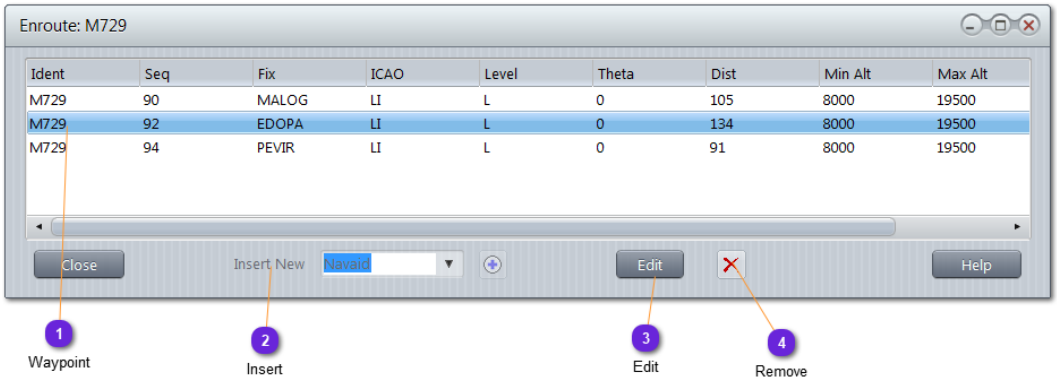
## 3 Position

Use this button to get the position selector. Mouse changes to +.
Click on the map at the desired position. Lat/Lon values are automatically updated with the mouse position.

## 4 Delete

Delete the actual selected record only if user added item.
Window will automatically close.

# Enroute

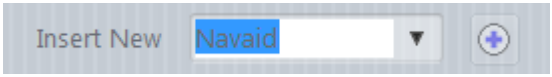



## 1 Waypoint

| Ident | Seq | Fix | ICAO | Level | Theta | Dist | Min Alt | Max Alt |
|---|---|---|---|---|---|---|---|---|
| M729 | 90 | MALOG | LI | L | 0 | 105 | 8000 | 19500 |
| M729 | 92 | EDOPA | LI | L | 0 | 134 | 8000 | 19500 |
| M729 | 94 | PEVIR | LI | L | 0 | 91 | 8000 | 19500 |

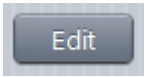
List all the waypoints/navaids defining the Enroute.

## 2 Insert




*Not available yet.*

## 3 Edit




Select one Enroute waypoint and use this button to review or edit it (available for edition only if a user element and not a database one).

## 4 Remove

When used, remove from the Enroute definition the selected point (but does not remove the point from the database).

# ADS-B

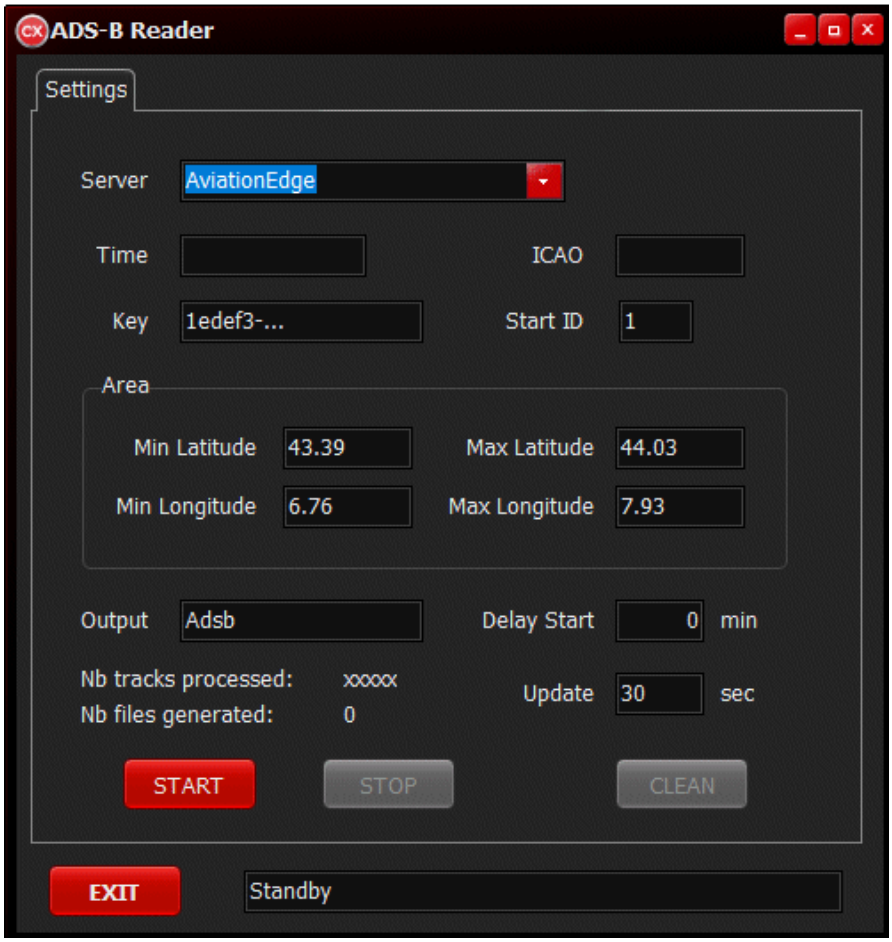
vsTASKER is offering an ADS-B support by importing and managing data from several ADS-B sources.
For now, the supported server formats are from Open-Networks and Aviation-Edge, although we may support more feeders in future versions.

ADS-B import is based on data extracted from Internet and stored into a directory. vsTASKER is using a proprietary feeder called **NetReaderApp** which is automatically started and configured by the **ADSbReader** component (see *Development Guide*).

Once the NetReaderApp is running, it stores all current flights from a given region into /Data/ADS-B/<output> folder. Files will be named output_index.log, index starting from 1 and incremented each time. All these files will make the ADS-B database for online/real-time or replay scenarios.


*Whatever the server selected, the output format will be the same and will match the data expecting at most by **ADSbData** Data-Model.*

### ADS-B




vsTASKER can run in sync with NetReaderApp, loading files as soon as ready and processing them when necessary. In this mode, vsTASKER can only run real-time.

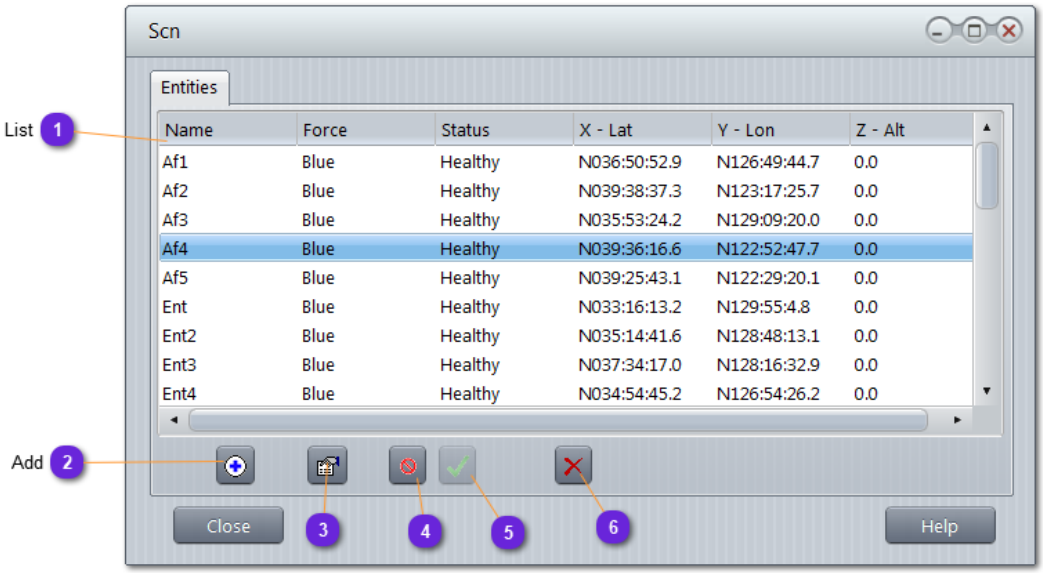
If the data (files) are already available (from a previous real-time run), it is possible to run vsTASKER in offline mode, based on the stored data, at a higher speed.

In db/RNAV, some samples are provided to help you setting up your first ADS-B scenarios. See also the **Tutorial** document for a step by step approach.

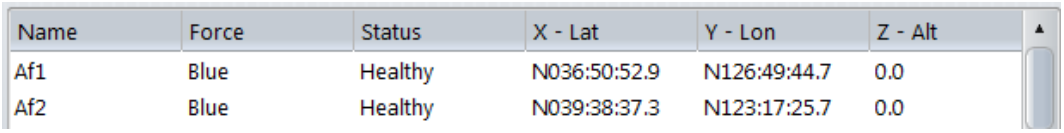
# Hook Window

At runtime, it is possible to open the scenario hook window to monitor data and mostly, the entities status.
To open the scenario hook window, **double click** the map background or use the right click **menu**:

# Entities



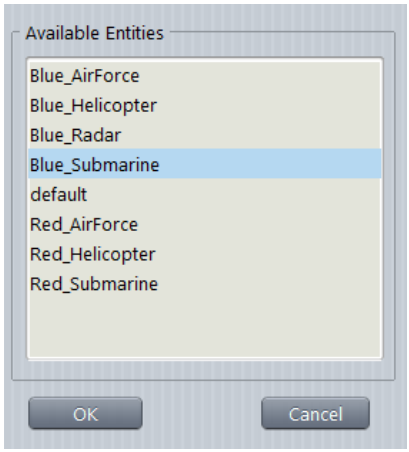

## 1 List




List all entities belonging to the scenario, including the disabled ones (status is showing Disabled and they are grayed out on the map).
Select one to grab focus (selection).
Double click to open the selected entity hook window.

## 2 Add

Use this button to add a new entity in the scenario.
A list of all catalog entities is then displayed.
Cancel will forget the add, selecting one in the list and pressing OK will change the mouse cursor to a + for the position. Click on the terrain map where the new entity should be placed.




## 3 Hook Window

Display the hook window of the selected entity (same as double clicking the list)

## 4 Disable

If the selected entity is running (activated), depressing this button will disable it (inactivate).

## 5 Enable

If the selected entity is not running (deactivated), depressing this button will enable it (activate).

## 6 Delete

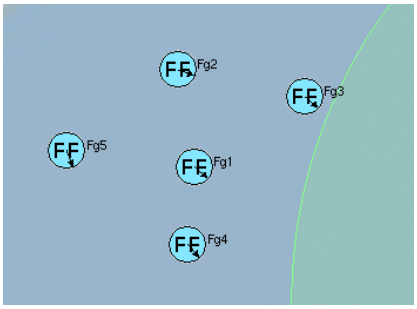
Use this button to remove the selected entity from the runtime scenario. A question dialog will request confirmation.

# Entities

Scenarios are normally populated with actors we call entities.
As an example, an entity can be a car, a tank, an aircraft or a boat, a submarine, a satellite or a soldier but can also be a factory or an airport.
Entities can be members or master of a unit (or both).

To act into the scenario, entities need behavior elements like Logic, Knowledge and Models.
They also can be represented on 2D map or 3D scene.

Entities are the living elements of the scenario and they model what matters. The simulation engine will give them life at runtime.



Entities organized into a Unit

A truck and a UAV on a terrain map

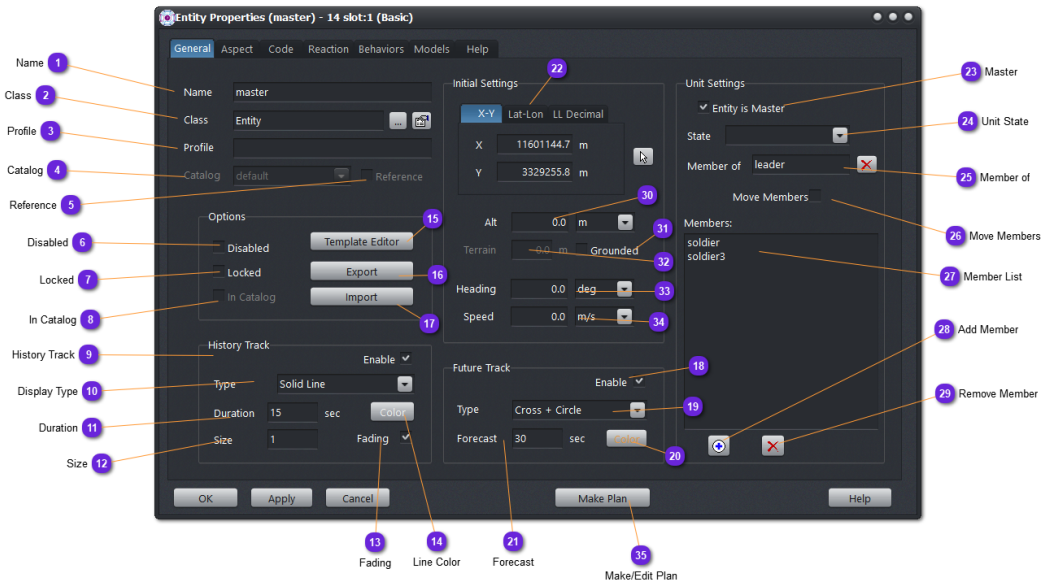

Ground entities on a 3D scene

- **Environment**

# Properties

Property window is called by double clicking on the entity itself, in Terrain mode, or from the contextual menu (mouse right click) or even using the  button once the entity is selected on the Environment list.

# General




## 1 Name

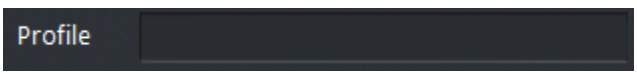

Entity name that must be unique. Cannot hold spaces or some special characters.
Unicode characters can be used here (Asian characters for example).

## 2 Class

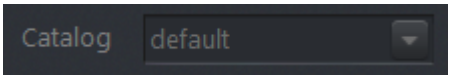

By default, an entity inherit from the Entity class. Select here another class for the entity using ...

Use  to open the class code window.

## 3 Profile

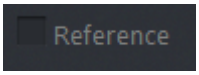


## 4 Catalog




If the class has been instanciated from a Catalog entity, the originating name will be displayed here. Cannot be changed unless Reference is selected.

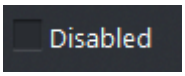
## 5 Reference




If Reference is checked, the entity will not be modifiable individually. Only the Catalog entity can be. At runtime, all the "reference" entities will be instanciated based on the specified catalog (same as for runtime entities). Only name and position of the entity can then be set at design time.
"Reference" entities are handful when they are numerous and depend on behavior definition that can change for all of them, instead of propagating the change individually which can be tedious and error prone.
When checked, as the entity is only a link, the Catalog name can be changed at anytime during design.

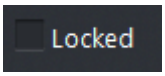

*Not available yet.*

## 6 Disabled



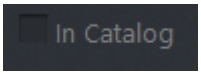

Check this to disable the entity without removing it from the scenario.
Disabled entities no more exist for the simulation even if they have a memory footprint and a data persistency in the runtime engine.
When an entity is disabled, it is created and is ready to be activated but will not exist from the scenario perspective.

## 7 Locked



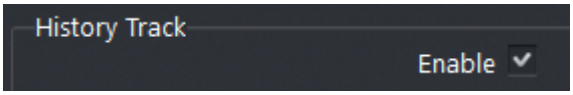

Check this to prevent the entity to be accidentally deleted or moved. Code and behavior cannot also be modified.

## 8 In Catalog



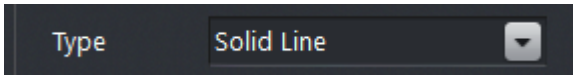

*Not used for now.*

## 9 History Track



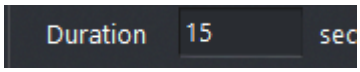

When Enabled, the previous entity positions recorded by the GUI will be recorded and displayed on the map according to Type.
History Track can be controlled from the code using the Entity API (entities.h)


## 10 Line Type



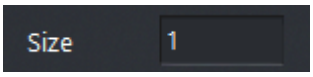

Chose here the style of the history line, from stream of dots to, dash or simply a solid line.

## 11 Line Duration



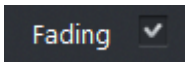

Specify here in seconds how long in the past will be the recording.
If the duration is 15 seconds, that means that the beginning of the history line displayed marks the entity position 15 seconds before the current time.

## 12 Size



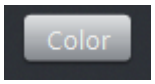

Size of the history line or width of the dot. Default is 1 for the line and 2 for the dot.

## 13 Fading



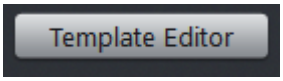

When enabled, the history line will fade from full brightness at start to black at tail. Default setting.

## 14 Line Color



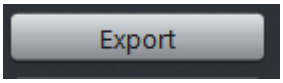

Line color of the history (track) line. By default or when the selected color is pure black (0x00) the entity color is used.

## 15 Template Editor



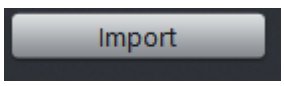

See explanations here.

## 16 Export



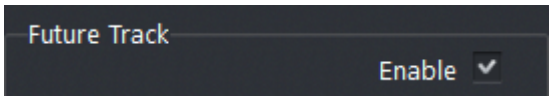

Will export the current scenario to an ASCII file for exchange (or re-use). Not all data is exported. Scenario does not export entities, only catalogs.

## 17 Import




Will import (by overwriting the current scenario) a previously saved scenario. Name will also be changed. All current scenario data will be lost.

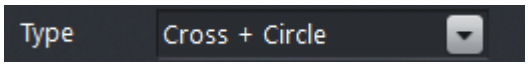
## 18 Future Track




When activated, will display a visual cue in front of the entity, forecasting the position the entity will reach in a specified number of seconds, according to the actual speed and heading.
Future track can be controlled from the code using Entity API (entities.h)


*The Future Track is not displayed when speed is around zero*

## 19 Display Type




Select the visual cue displayed for the forecasted position:

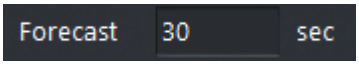
- X Cross
- Circle (empty)
- Circle with a X cross

A dashed line will connect the cue and the entity.

## 20 Future Track Color

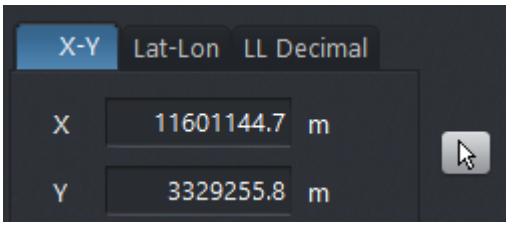

Set the color of the visual cue and dashed line for the Future Track.
Default is light orange which is good in both themes (dark and light).
Can be changed during runtime.

## 21 Forecast




Set here the number of seconds to use to compute the position ahead. If zero, the Future Track is not displayed.
Default is 30 seconds.

## 22 Position



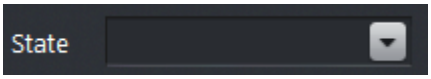

Position of the entity on the terrain coordinates. X-Y, Lat-Lon or Decimal Lat-Lon can be used for clarity. Use Apply when modifying the values in the text field.

If  button is used, the property window will vanish until user click on the terrain map to select the position. Property window will appear again with the new coordinates updated.

## 23 Master

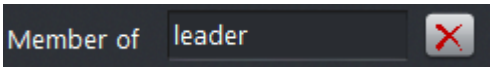

If checked, the entity will become Master of the unit (composed by itself plus the members).
See here for more details on how to work with Units/Aggregates.

## 24 Unit State



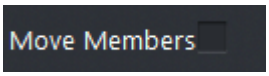

A unit can aggregate all its members, making them invisible and not selectable (but all of them will still be simulated).
If disaggregated, all entities will be visible and selectable but will be part of a unit and will be linked to the Master entity.

## 25 Member of



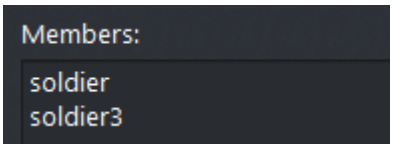

If the entity is a member of a Unit (and a member can also be a Unit itself for hierarchical aggregates), the name of the Master is notified here.
Selecting ✕ removed the current entity from the Master unit.

## 26 Move Members




When checked, disaggregated members will be displaced with the master, at design and runtime, when relocation is performed using the mouse or coordinates reposition.
This can be handful when a platoon or a battalion must be displaced from one terrain location to one another, without reorganization.
If not checked, the master entity will be relocated alone.

## 27 Member List




List of all entity members of this unit.

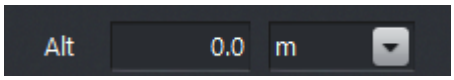
## 28 Add Member

Used to add another entity as a member of this unit. The list of available entities are not members of unit.

## 29 Remove Member

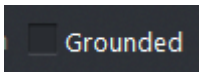
Select one or many entities from the above list and use this button to remove them from the unit.
Entities are not deleted, just moved out of this unit.

## 30 Altitude



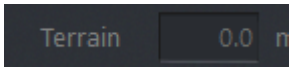

Initial altitude of the entity in the select unit. Changing the unit will also change the displayed value. Altitude is always above sea level. Shall be negative for submarines.

## 31 Grounded



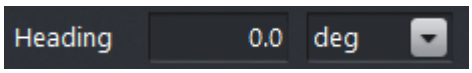

If checked, the altitude (Z factor) of the entity will match the terrain altitude (clamping). This is handful for land entities.
This is only for design time. A grounded entity will not necessarily be grounded by the dynamic model (if the model does not check this flag).
Checking Grounded insure that if the (land) entity will always be at the correct altitude even if moved during the scenario design.

## 32 Terrain Altitude



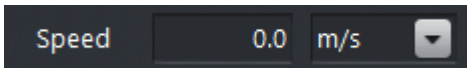

Display the terrain altitude (if any terrain layer provided into the scenario) below the entity position.

## 33 Heading



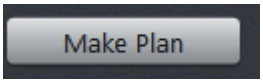

Initial heading of the entity expressed in the selected unit. Changing the unit will also change the displayed value.
Values are [-180,180] degrees.

## 34 Speed



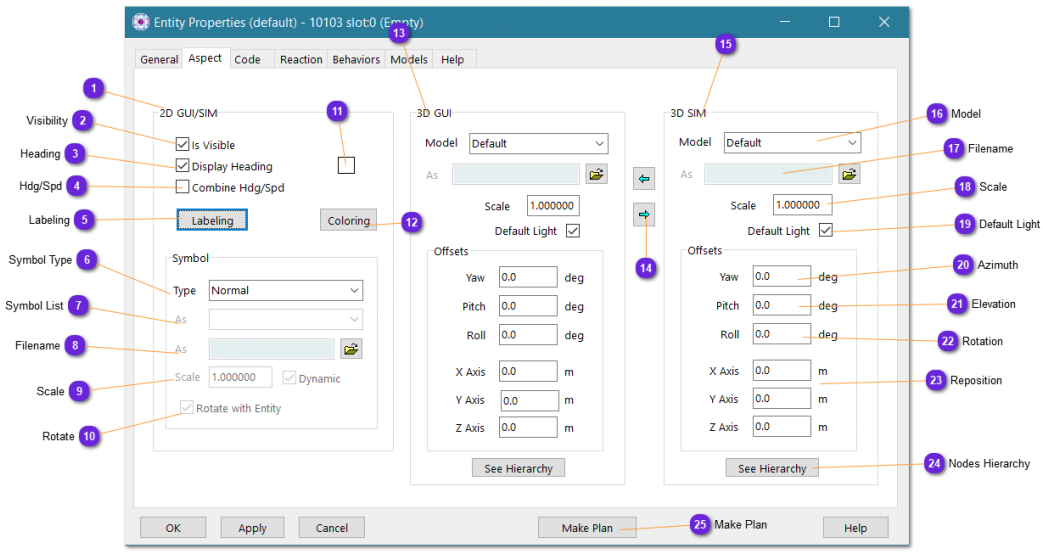

Initial speed of the entity expressed in the selected unit. Changing the unit will also change the displayed value.
The value is not bound by the dynamic model minimum and maximum values at design.

## 35 Make/Edit Plan

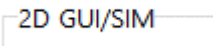



Use this button to create or update a Plan for this entity.
See here for how to work with plans.

# Aspect

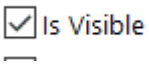



## 1 2D GUI/SIM



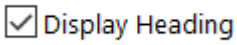

This block defines the aspect of the entity in the vsTASKER GUI terrain map.

## 2 Visibility



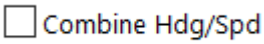

Check this to make the entity symbol visible on the map (otherwise, will not be visible).

## 3 Heading



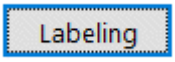

If checked, the heading will be represented as a little arrow aligned with the entity heading. Up meaning north.

## 4 Hdg/Spd




If checked, the length of the heading arrow will be proportional to the speed (the higher the speed, the longest the line).

## 5 Labeling




Set the font for the entity labeling and the display mode for both GUI and SIM map.
See here for a complete description.

## 6 Symbol Type


Select here the kind of symbol for the entity display:

- None: No symbol
- Normal: Default little empty square
- Normal Filled: Default little filled square
- Dot: OpenGL dot (square or round according to OpenGL smoothing mode)
- Bitmap File: Select it in (8)
- Symbol: Predefined symbols loaded in Preferences. See here.
- 2525B: List of Military standard symbols.
- Vessel: little oriented filled triangle.

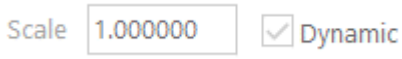
## 7 Symbol List


Select the kind of predefined symbols extracted from the list. See here for more details about the lists.

## 8 Filename

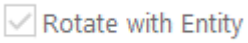

If Bitmap File is selected, enter here (or browse) for the bitmap file to load and apply as a texture.

## 9 Scale




Scale alteration of the loaded bitmap. Scale value will be applied equally on the width and height of the symbol.
If Dynamic is checked, the size of the bitmap will vary according to the zoom level. The higher the zoom, the bigger the symbol. If Dynamic is unchecked, size of the bitmap is unchanged whatever the zoom level.

## 10 Rotate

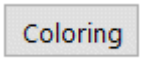



If checked, the symbol (bitmap only) will rotate with the heading. This has no effect on Normal, Normal Filled and Dot symbols.

## 11 Preview

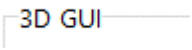
If possible, the selected symbol type will be displayed on the preview square.

## 12 Color




Color of the drawn symbol for types Normal, Normal Filled and Dot.
The color must not be confused with the force (IFF) of the entity.

## 13 3D GUI

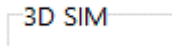



This block is for the Perspective and Globe views that uses the OpenGL library for display.
The specified 3D models must be loadable under OpenGL.

## 14 Copier

Use these buttons to copy all values from 3D GUI block to 3D SIM block or reverse. This can be useful when the Editor 3D is used as a tester for model and offset settings before forcing the simulation engine (stealth) to use the same.

## 15 3D SIM




This block is for the simulation engine output and will be used by the 3D library specified by the selected Viewer.
vsTASKER GUI will try to load these models into OSG when 3D Editor mode is selected.
If the model cannot be loaded by OSG plugins, it will not be displayed on the 3D Editor but might/will into the simulation engine if the correct 3D engine library supports it.

## 16 Model

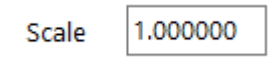

Type of the model (file) to load. Even if not mandatory, it is a good practice to make both the file and the setting correlated as the code generation might use the Model setting to call the proper loader, regardless of the file model type.

- None: No model to load
- 3DS Max: Autodesk 3ds format
- OpenFlight: Multigen/Presagis flt format
- OpenSceneGraph: osg or ive format
- Bitmap: Any raster file
- Delta3D: OSG format
- Collada: XML dae format

## 17 Filename

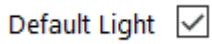

Enter here (or browse) the 3D model file (according to the chosen type above).
The full path will be stored. It is not possible yet to specify a relative path for 3D model files.

## 18 Scale




Enter here the scale factor that will be applied to the loaded 3D model by the graphic engine (1 means unchanged, 0.5 is two times smaller, 3 is three times bigger)

## 19 Default Light

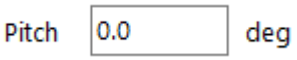



When checked, the model will be illuminated with the default light (if any defined). When unchecked, ambient lightning is used.
This setting is for OpenSceneGraph only. Sometimes, some objects appear totally white in some lightning conditions. Changing this mode restore the texture.

## 20 Azimuth


If the 3D model needs to be turned around the Z axis (heading effect), enter here the values in degrees. 0 means no offset.

## 21 Elevation

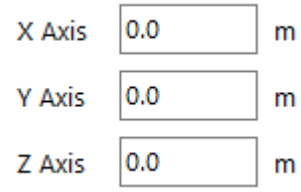



If the 3D model needs to be turned around the X axis (pitch effect), enter here the values in degrees. 0 means no offset.

## 22 Rotation

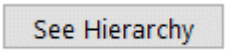

If the 3D model needs to be turned around the Y axis (roll effect), enter here the values in degrees. 0 means no offset.

## 23 Reposition



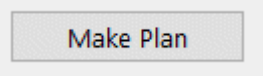

If the center of gravity of the 3D model must be displaced, enter here the offset value in meters.
In some models, the center of gravity is the lowest part of the model, in some, it is the middle part.
The Z value helps clamping visually a ground entity.

## 24 Nodes Hierarchy



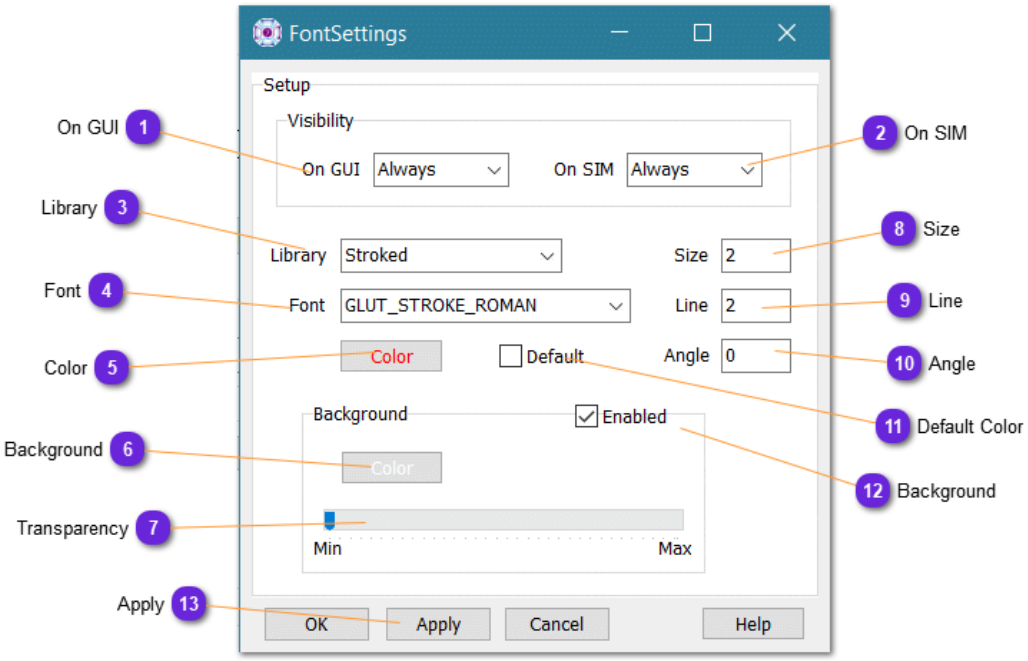

This button will call an OpenSceneGraph command which will load the model and print out on a console window the hierarchy of nodes.
This can be useful in case articulated parts have to be found for manipulation with a specific component.
Result depends on the model definition itself. Some has named nodes, some not.

## 25 Make Plan

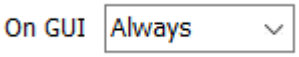



Call the Plan definition window. See here for complete description.

# Labeling



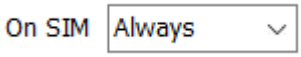

## 1 On GUI




Specify the display on the GUI map, in design and runtime.

- **Always**
- **On Select** - when entity is selected
- **Never**

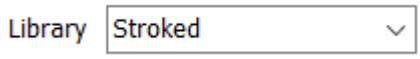
## 2 On SIM




Specify the display on the SIM map, OpenGL or HMI.

- **Always**
- **On Select** - when entity is selected
- **Never**

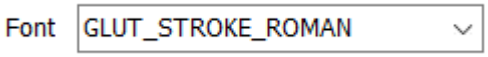
## 3 Library




Select which kind of font will be used:

- **Bitmap** - very efficient but fixed
- **Stroked** - more CPU intensive and limited to a set of fonts
- **Freetype** - the most aesthetic but with low performances

## 4 Font

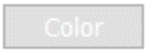



According to the library, select the font from the provided list.
With Freetype, the font must be loaded from an otf or ttf file. A file dialog window appears when clicking on the drop down.

## 5 Color


Select the font color. A Color dialog window will be used.

## 6 Background

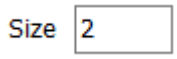



Select the font background color. A Color dialog window will be used.

## 7 Transparency

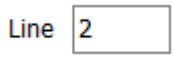

The font background can be fully **opaque** (**min**) or fully **transparent** (**max**).
Set here the desired transparency value.

## 8 Size



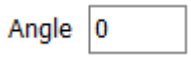

For Stroked font, specifies the scaling factor of the provided font. 1 == original size, 0.5 = half.
For Freetype font, it sets the size of the font in pixels.

## 9 Line



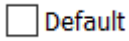

For Stroked font only. Thickness of the line used to draw the font.

## 10 Angle



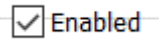

For Stroked and Freetype fonts only.
Sets the fixed orientation of the label on the plane. 0 for horizontal.
Negative values rotate the label clockwise.
In degrees.

## 11 Default Color



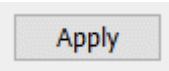

When checked, the font color will be the one set for the entity.
Background will not be available with default color.

## 12 Background




When checked, the font will have a background rectangle behind, for easy reading.
Default color must be unchecked.

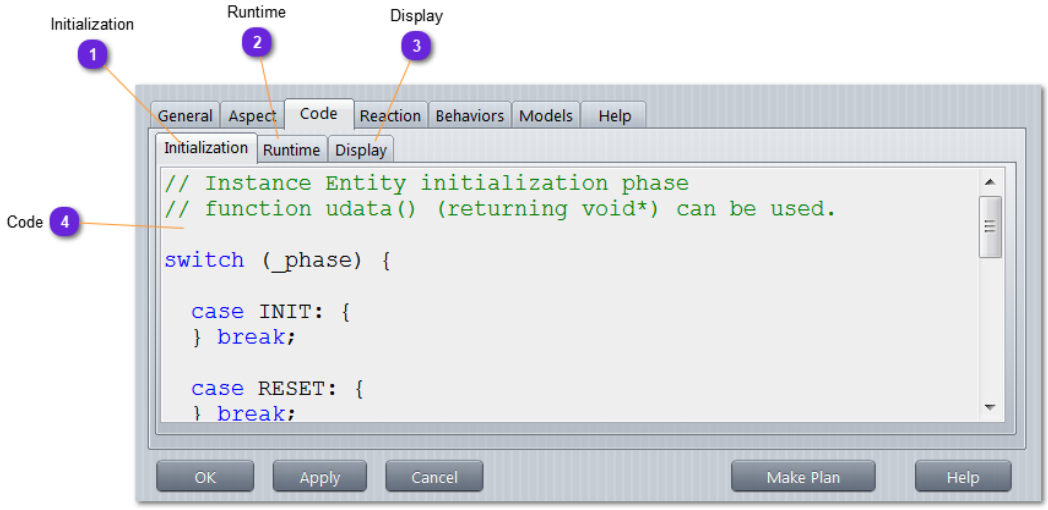
## 13 Apply



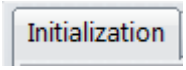

Whenever a change is made, press Apply to see the result on the map, for the selected entity.

# Code

Each entity can hold its own code that will be called after the Class code (it inherits). This allows specialization.
For example, the Class code can setup general parameters in built-in components and the entity code will tune them according to specific user code.
Two entities from the same Class will then share some same code but hold their own Initialization, Runtime and Display parts.



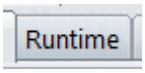

## 1 Initialization




This part of the code is called after the Class initialization code.
Put here what is specific to the instance entity.


*Refer to the Developer Guide for more details on system phases/events.*

## 2 Runtime

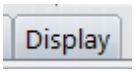



The Runtime part is called at every cycle. The code inside must be thought as from inside the simulation engine.
Put here whatever code you like, although it might be wiser to use either a Component or a Logic.
Do not put here any drawing function; use the Display part for this.
This part is called at the RTC frequency specified in Database (Runtime) but is not called in Freeze mode.
The entity implementation code is a function that must return a value:

- **AGAIN**: the entity is called again at next cycle.
- **DONE**: the entity finishes normally. All Logics and Knowledge are terminated. Entity is dormant.
- **EXIT**: the entity finishes normally and is removed from the scenario.


*The user can put whatever C/C++ code he wants there. This code can access all vsTASKER API, all included third party APIs and all user designed scenario and model data of the current database. The Runtime code comes from the first product versions and is mostly kept for compatibility.*

## 3 Display




The Display part is called at every cycle (except for the Console viewer).
It is called from inside the drawing loop (OpenGL, OSG, VegaPrime, Delta3D...) so, direct call to the third party graphic engine is allowed.

## 4 Code

```
// Instance Entity initialization
// function udata() (returning voi

switch (_phase) {
```

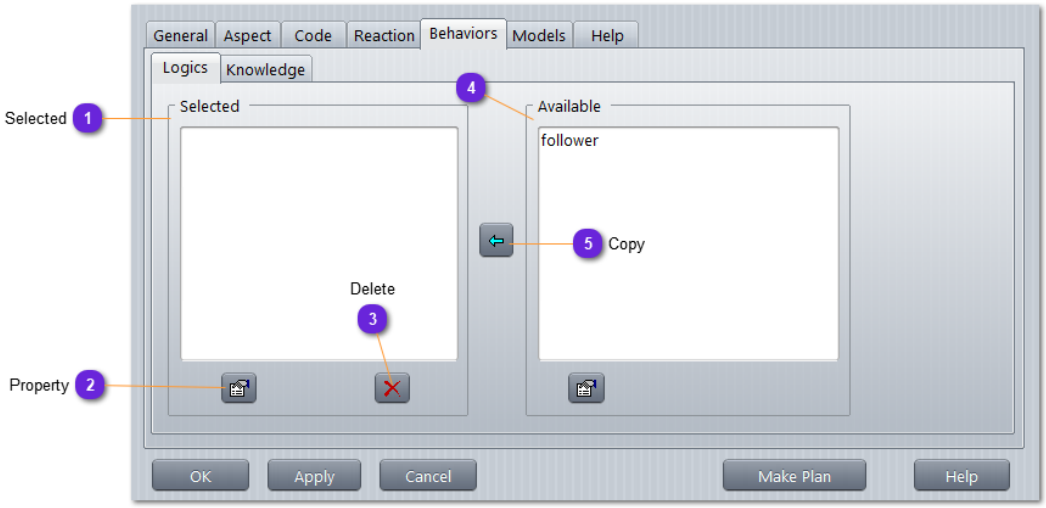
Write here your own code.

# Reaction

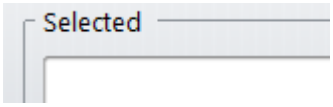
Similar to Scenario Reaction settings.
See here.

# Behaviors

Same explanations for Logics and Knowledge panes




## 1 Selected




This list all the logics belonging to the entity (behavior).

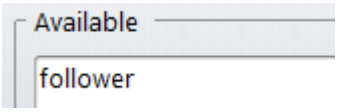
## 2 Property

Show the property window of the selected logic in the list.

## 3 Delete

Remove from the current entity behavior the selected logic.

## 4 Available

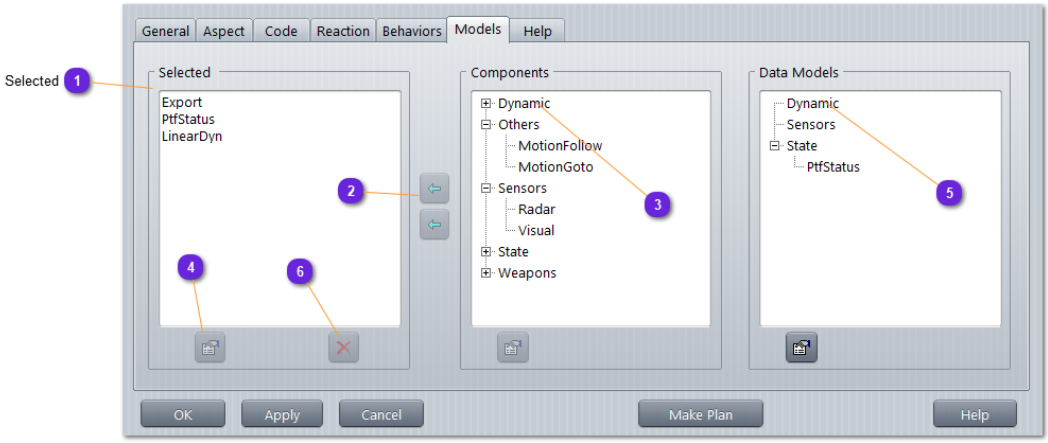



List of all logics available in the database.

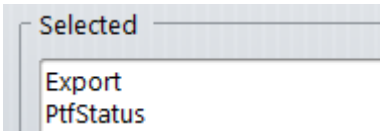
## 5 Copy

Add the selected logic (from the Available list) to the entity behavior.

# Models




## 1 Selected

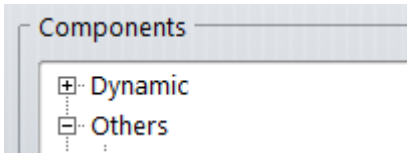



List here all the entity behavior Models (Components and Data-Models)

## 2 Copy

Copy to the entity behavior Model list the selected Component or Data-Model

## 3 Component List

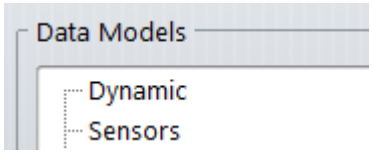



List all available Components of the database.

## 4 Property

Show the property window of the selected item in the list.

## 5 Data-Model List




List all available Data-Models of the database.

## 6 Delete

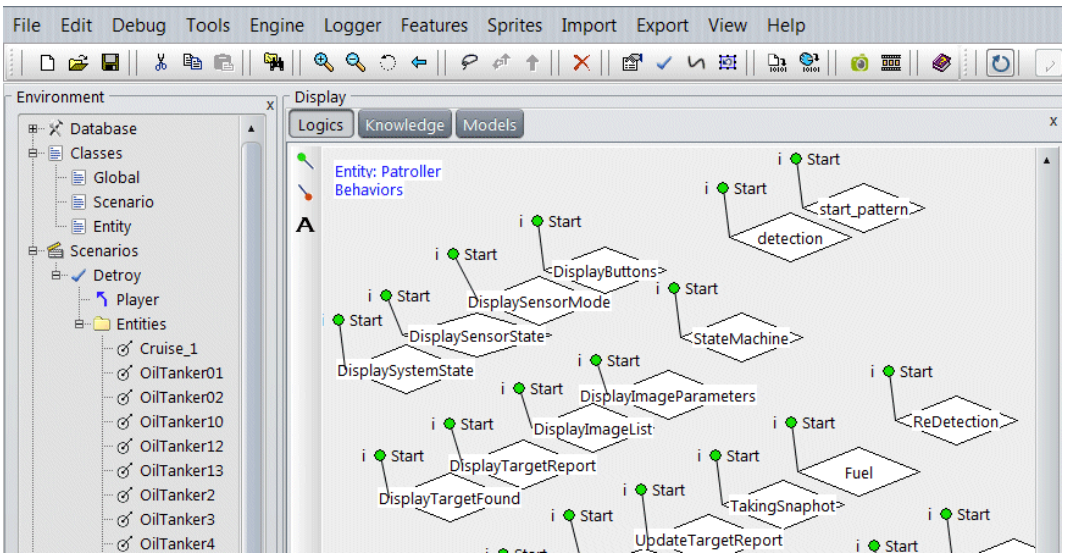
Remove from the entity behavior the selected item of the list (Component or Data-Model).

# Behaviors

An **entity** holds one or a set of Logics, Knowledge and Models defined in the database.

Behavior objects are references to logics, knowledge and components. These references will be instanciated at runtime so that each entity will hold their own copy of the logic, knowledge or model with their own runtime data. The behavior of one entity is thus defined by the combination of these elements.

To define or modify the behavior of one entity, first, selected in in the Environment tree-list (or the map). The Display diagram shows then the behavior objects and the panes for selecting the proper category (Logic, Knowledge, Models).




A behavior is defined by a set of logics or a knowledge or models associated with an entity. Without behavior, an entity can do much.

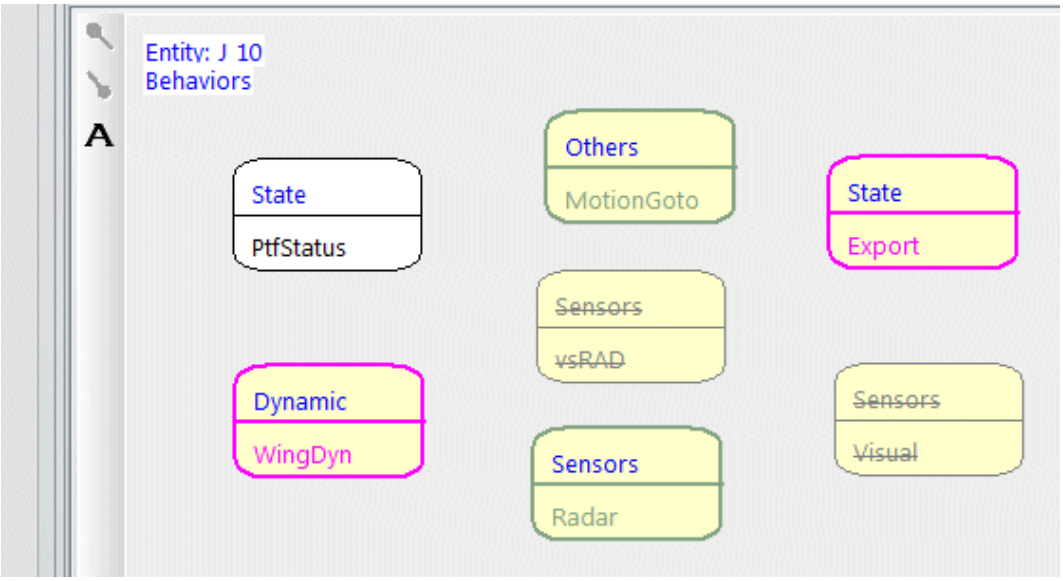
When a logic (knowledge or model) is defined globally in the database, it might need specific data when associated with a given entity. For i.e., if a logic activates a sensor upon reception of an event, then, each entity using this logic must specify some specific data for the logic, like the name of the sensor to be switched ON, as each entity might hold different sensors.

A behavior can only reference existing elements in the database. If a logic is needed for one or many entity behavior(s), it must be defined first in the Logic Environment, then used in the behavior logic panel for all entities that need it.

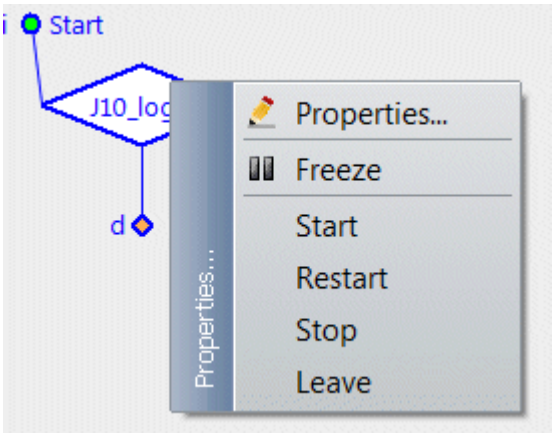
- **Runtime Controls**

### Behaviors

When an entity is selected, all running logic, knowledge or model are displayed in magenta.
The one in grey are disabled or not started.
The green one are pending (ready to start)




During **runtime**, any behavior object (logic, knowledge or model) can be manually **started**, **frozen**, **resumed** and **stopped**.
This can also be done from the behavior property window or from the object **popup menu** (or double clicking the object).




**Properties**: call the property window of the object

**Freeze/Resume**: pause/release the selected behavior (object).

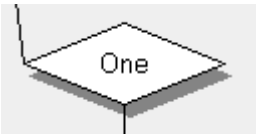
**Start**: If the object was pending, start it.

**Restart**: Stop the object (if it was running or paused) and start it again.

**Stop**: Stop the object but do not connect (Logic and Knowledge only).

**Leave**: Stop and connect to the next object (Logic and Knowledge only)

# Logics




**Entity** [behavior] **logic** is a **reference** to an existing Logic defined in the current database.
To **add** a new logic as an entity behavior, open the entity property window, select the behavior then logic panel, select a logic in the available list and use the button to move it to the selected logics list. To **remove** a logic from the behavior list, select it and use the button.

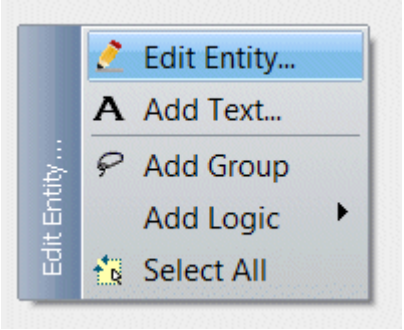
An entity logic can have Entry and Exit Points.
A logic is **started** when its EPoint triggers.
A logic is **stopped** when its XPoint triggers or when an object inside the Logic (not pertaining to a group) returns QUIT.

To **connect** one logic to one another, click the connector head with the mouse while maintaining down the left button then drag it toward the other Logic. When close enough, the object shows magenta anchors. Release the mouse while close to one of these anchors.

To **disconnect** one Logic, select the arrow head of its connector and move it away from any anchor, then release. The arrow changes to orange head (Done mode, default).


*Double clicking a behavior symbol pops up the setting property window.*

## • Popup menus




**Edit Entity**: call the entity property window

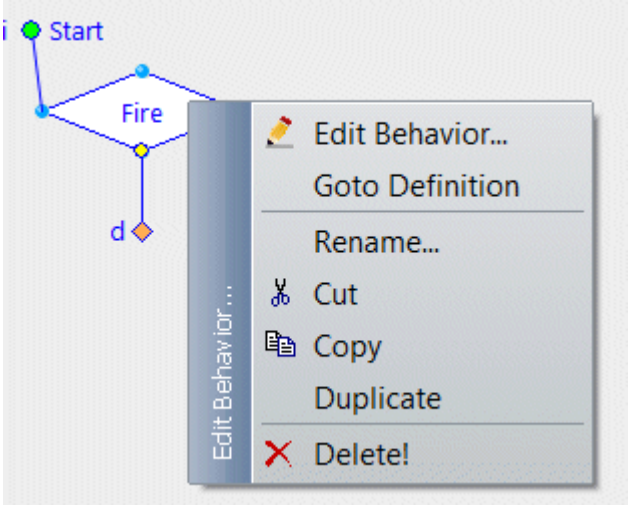
**Add Text**: add a label

**Add Group**: insert a behavior group that encapsulates logics

**Add Logic**: insert a logic from the database

**Select all**: select all behavior logics

### Logics




**Edit Behavior**: call the behavior property window

**Goto Definition**: jump to the logic description (same as selecting the logic in the Environment)

**Rename**: rename the behavior logic.

**Cut**: copy to clipboard with deletion flag raised

**Copy**: copy to clipboard

**Paste**: if a logic [behavior] is in the clipboard, menu will be available

**Duplicate**: copy/paste locally

**Delete**: remove the logic behavior (not the logic itself)

# Knowledge


**Entity** [behavior] **knowledge** is a **reference** to an existing Knowledge defined in the current database.
To **add** a new knowledge as an entity behavior, open the entity property window, select the knowledge panel, select a knowledge in the available list and use the button to move it to the selected knowledge list.
To **remove** a knowledge from the behavior list, select it and use the button.

An entity knowledge can have Entry and Exit Points.
A knowledge is **started** when its EPoint triggers.
A knowledge is **stopped** when its XPoint triggers or when a context inside the knowledge returns QUIT.

To **connect** one knowledge to one another, click the connector head with the mouse while maintaining down the left button then drag it toward the other knowledge. When close enough, the object shows magenta anchors. Release the mouse while close to one of these anchors.

To **disconnect** one knowledge, select the arrow head of its connector and move it away from any anchor, then release. The arrow changes to orange head (Done mode, default).

*Double clicking a behavior symbol pops up the setting property window.*

## • Popup menus

See here

# Models


**Entity** [behavior] **models** is a **reference** to an existing Model defined in the current database.
To **add** a new component into the entity behavior, open the entity property window, select the models panel, select a component in the available list and use the button to move it to the selected component list.
To **remove** a component from the entity behavior list, select it and use the button.

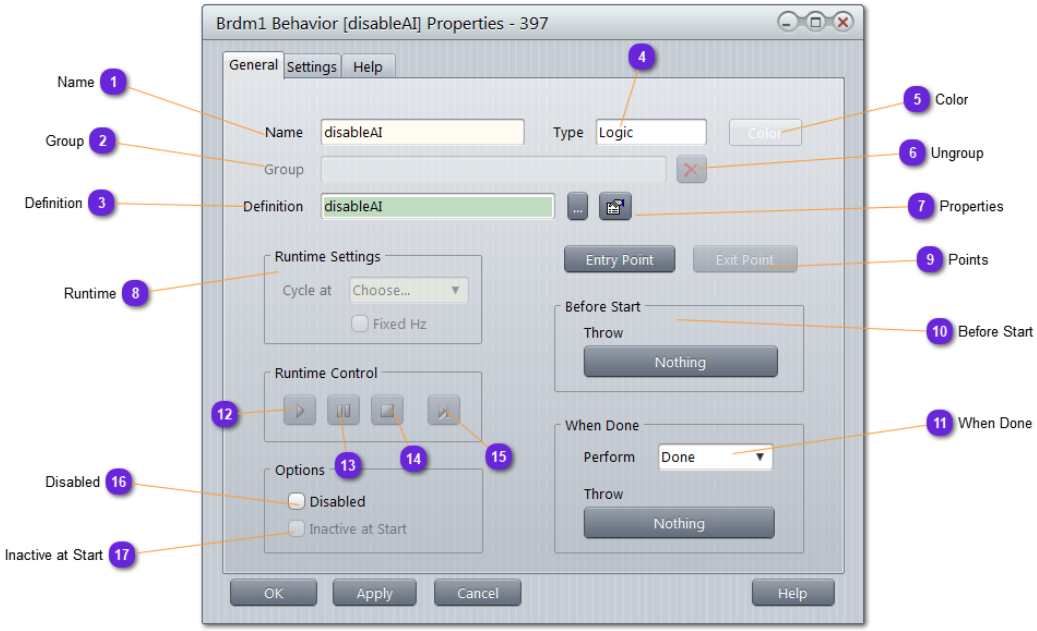
An entity component creates an instance at entity creation time and every time the simulation starts.
All Components are stored into a list and processed according to the general component runtime setting.
They are deleted at simulation stop.

*Double clicking a behavior symbol pops up the setting property window.*

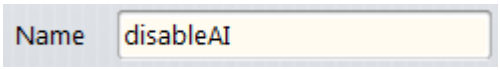
## • Popup menus

See here

# Properties




## 1 Name

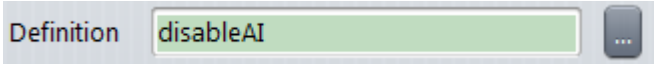



Name of the behavior object. Must be unique.

## 2 Group

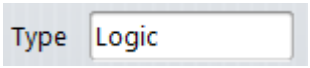

Name of the holding group if embedded (ie: group1::group2...)

## 3 Definition



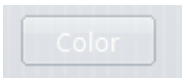

Name of the referenced object (logic, knowledge or model).
Use the button ... to select another one.

## 4 Type




Type (category) of the referenced object (Logic, Knowledge or Model)

## 5 Color




Open a color selector dialog to define the background color of the behavior object display on the diagram.
It is a good idea to use different colors per type of objects or behavior (groups can also clean the design).

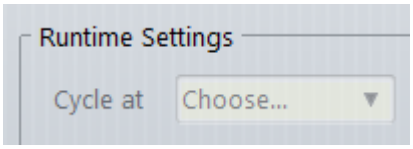
## 6 Ungroup

Use this button to ungroup (of one level) the behavior object, if embedded.

## 7 Properties

Open the referenced object property window.

## 8 Runtime

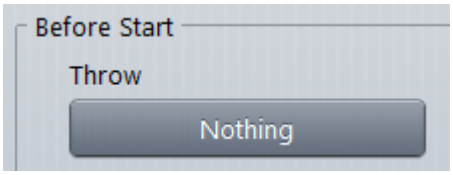



Change locally if needed the default component runtime heart beat. Although it might not be recommended to change the default settings for internal components, it may be interesting to have different frequency for different entities sharing the same component.
Let's imagine a component that checks the distance to a target. A high speed entity like an aircraft would have the component running at a higher frequency than the same one attached to a human character or a very slow entity.
See here for the frequency explanations.

## 9 Points

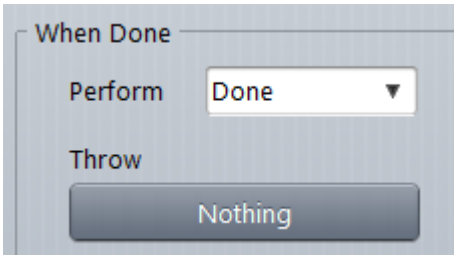

Setting of the Entry and/or Exit Points attached to the behavior object.
They must be created from the diagram panel first.

## 10 Before Start




Specify here the event or fact to be triggered just before the behavior object is activated.

## 11 When Done




Perform mode set the termination mode:

- Connect: The connection link is activated. Event/Fact are triggered.
- Done: The connection link will not be activated. Event/Fact are triggered.
- Quit: All behavior objects are stopped. If inside a group, all group objects are stopped and group is left. Event/Fact are triggered.
- Nothing: Like Done, but without Event/Fact triggering.

Specify here the event or fact to be triggered just after the behavior object is left.

## 12 Start

Start/Restart the behavior object.

## 13 Pause

Pause the behavior object if running.

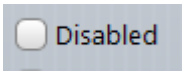
## 14 Stop

Stop the behavior object but do not connect (only for Logic and Knowledge).

## 15 Leave

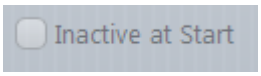
Stop the behavior object and connect to the next one (only for Logic and Knowledge).

## 16 Disabled




Check this option to disable the behavior object. When disabled, the object is not instanciated and code not generated.
A disabled object cannot be started at runtime.

## 17 Inactive at Start

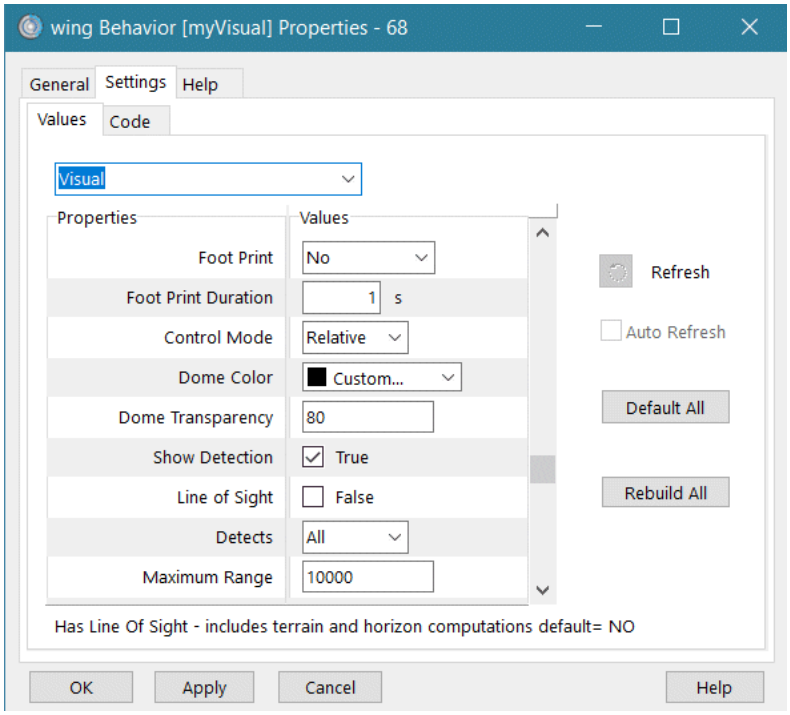



Check this option to suspend execution at startup of the behavior object. Manual start is mandatory (from the GUI or from the code).

# User Settings

If the Logic, Knowledge or Model definition includes a Dyn-UI, the published entries appears the in Value List editor below.

If blank values are set, default one will be used instead.
If user enters its own, they will be used after default values.
All entries are string values. They are converted in their proper types at runtime.
Toggle Default: this button retrieve the Dyn-UI default values as defined from the static interface definition. Pressing this button again removes entries that are identical to the default values.
For model Component or Data Model, if inheritance is used, the Inherit pull down menu will display user interface for each parent.

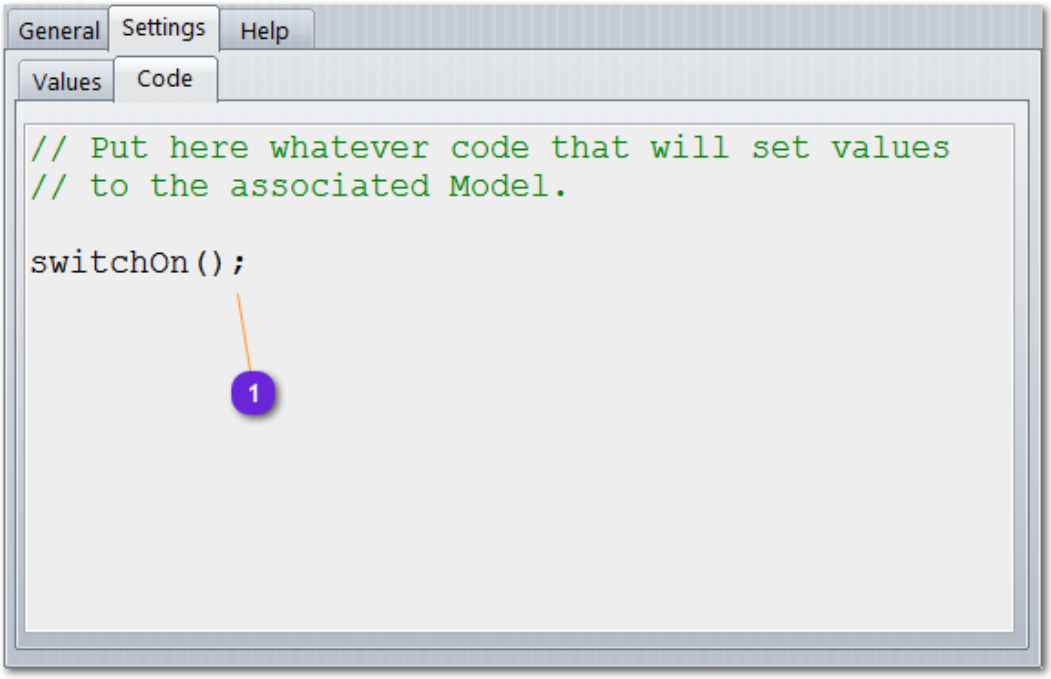



See here to learn how to use the interface.

## • Runtime Setting

Changing values during runtime can update the simulation engine with updated values by pressing the Send button.
Runtime (modified) values will be lost and replaced by design ones as soon as simulation ends.

# User Code




## 1 User Code

```
// Put here whatever code that will set values
// to the associated Model.

switchOn();
```

In this panel, you can put any C++ code to set data of the associated logic, knowledge or model, for public or protected data defined in each object (class), or using the published methods.
This code will be directly included into the object class that is referenced by the behavior object.

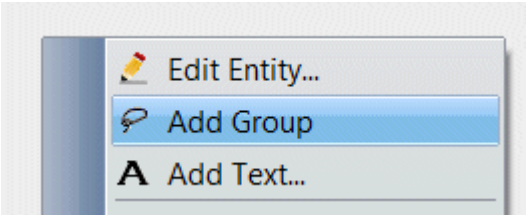
*For ie: Logic foo contains a public variable A and a method setup().*
*From a behavior object named foo, referencing logic foo, the User Code could hold the following code:*

```
A = 1;
setup();
```

# Group Behavior

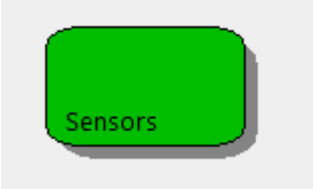
A **Group behavior** embed a list of logic, knowledge or models. It behaves as a big container that has control over its contained behavior objects.

To create a group, use the contextual menu (in a behavior pane):



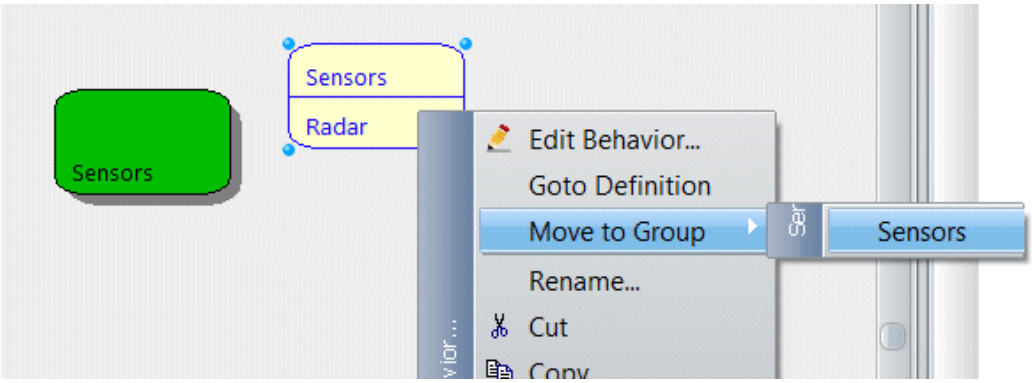

< select this menu

Once the group is created, the object symbol will be drawn with a shadow:



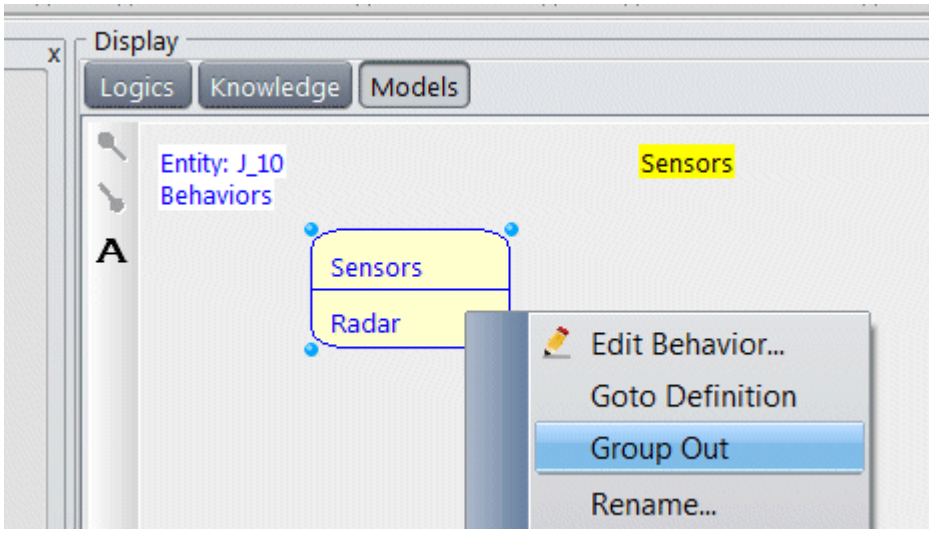

You can double click the object to open it and see its content.
A group can contain another group and so on (be reasonable in the depth)

From inside a group, you can add all logic, knowledge or models you want. They will all belong to the group.
To move a behavior object into a group (located in the same level), select the object and use the right click menu:

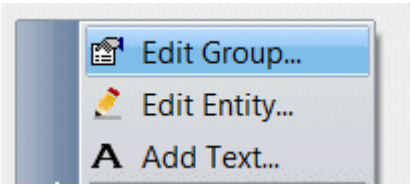



In the above example, Radar will be removed from the current level (might be a group or the root level) and moved to the Sensor (group).

To move out an object from the current group (and drop it outside), select the object and use the right click menu:

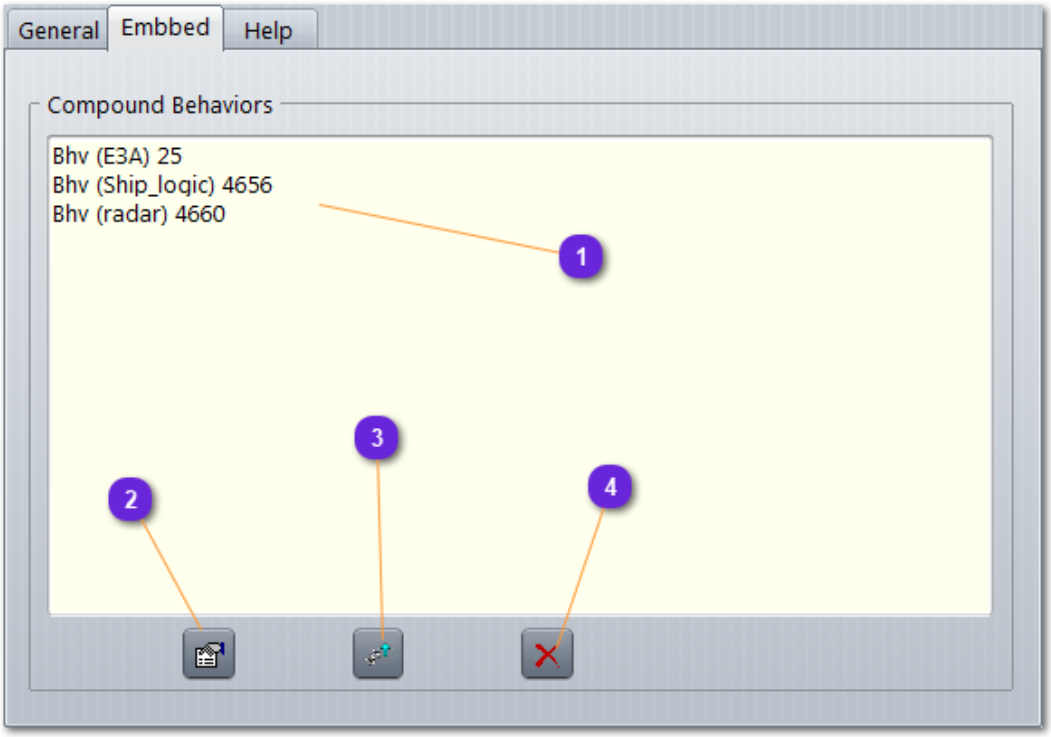



To edit a group behavior, select the group, then right click and chose Edit Group or from inside the group, use the right click menu:

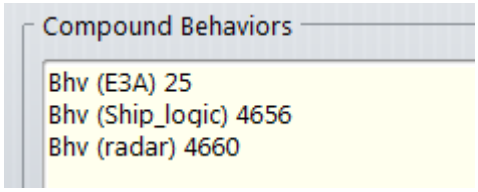


# Embbed

For the **General** panel of the Behavior Group properties, see here.




## 1 Behavior object list




List of all objects (including groups if any) included into the opened group.

## 2 Properties

Open the property window of the selected object of the list.

## 3 Ungroup

Removes from the current group all the selected objects of the list.

## 4 Delete

Delete all the selected behavior objects of the list.

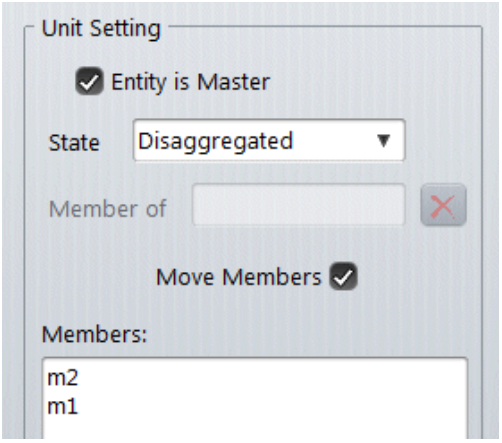
# Units / Aggregates

Entities can be organized as groups or hierarchical gathering called Units or Aggregates.
The benefit of such teaming is the ability to aggregate all members behind a unique master entity. This simplifies the representation on the map without cutting down the accuracy (aggregated entities can be simulated individually but without display on the map)

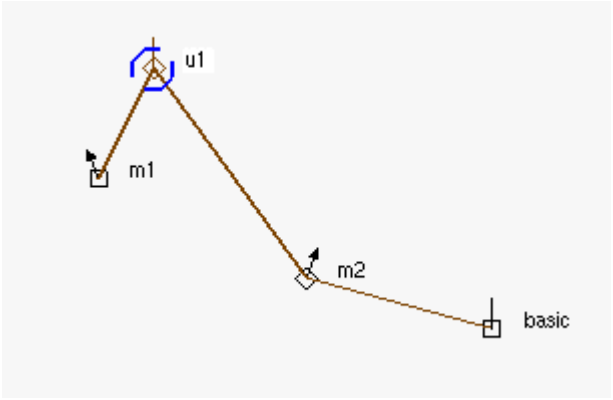
## • At Design

To create a Unit, one entity must be defined as the Master and others as Members.

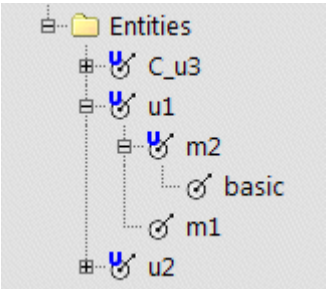

*A member can also be a Master of a sub-unit.*




On the terrain map, units are displayed with connecting brown lines (when a Master is selected) or a green line to the Master (when a Member is selected):



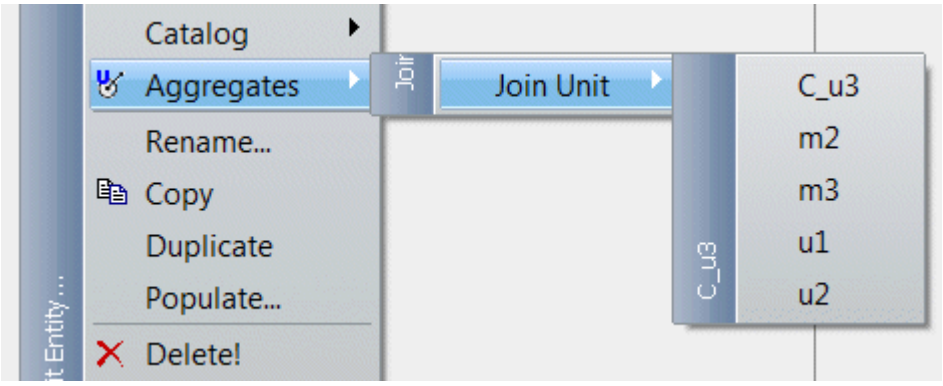
Units are also represented in a hierarchical way on the Environment tree list:




## • Using popup menus

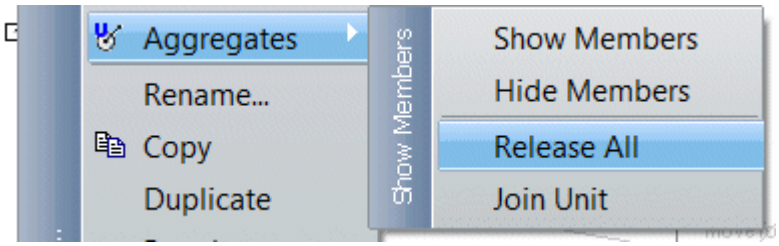
On the terrain map, when an entity is selected, it can be made a **master unit** by selecting the Make Master option:


To **add members** to this new unit, just select desired entities one at a time and make them Join the Unit:




For a **member**, to **leave** a unit, just use the option Leave Unit:

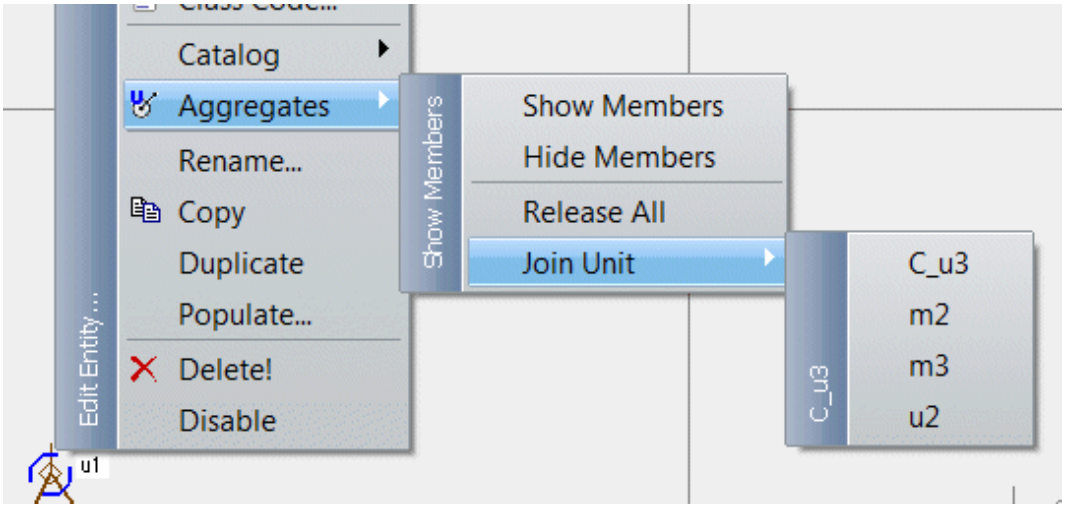

For a **unit,** to **release** all its members and become a single entity, just use the Release All option:




## • At Runtime

At runtime, a Master will be able to raise events only to its members. Conversely, a Unit Member will be able to raise an event only to its Master.
Communication channels can also used inside a Unit for exchanging data amongst members (report, synchronization, etc.)

From the GUI or using the API, it is possible for any entity, to join or leave a Unit.




Vt_Entity class Unit API:

```
 int  makeUnit(UnitState =U_Disaggregated); ///< Make an entity a
master unit.
 int  freeUnit();                           ///< Release all
members from the unit. Master becomes normal entity.
 int  joinUnit(Vt_Entity*);                 ///< Join the given
master unit
 int  leaveUnit();                          ///< leave the current
unit if any.
 int  isUnit();                             ///< Is entity a
master?
```

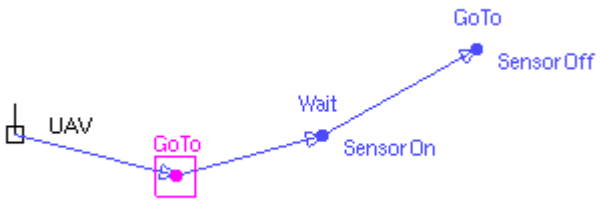
```
 int  isMember();                                ///< Is entity a unit
member?
```

# Entity Plan

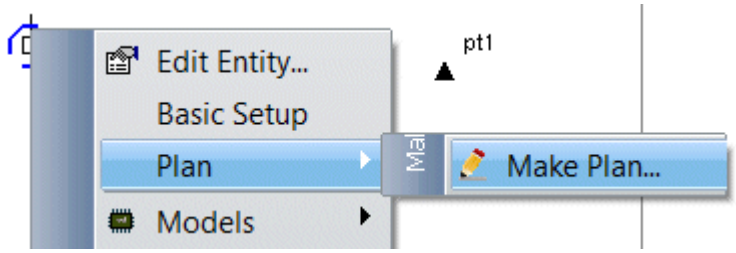
Beside of using logics and knowledge, it sometimes very useful to draw on the map where to go, how to get there or what to do when getting there.
Plan is covering this functionality.

A Plan is a string of Plan-Points, normally geolocalized (but some are not) that refer to existing routines.
The same way as behavior objects are referring to existing logics (knowledge or models), plan-points refer to existing routine.

When the simulation starts, the first routine of the entity plan is also executed (at the specified frequency).
The routine can perform any process (checking the speed, the position of the entity regarding the plan-point position, raising events or whatever computing).
As soon as the routine returns `Done`, the next following routine is automatically started, until this one returns `Done`, etc, until the end of the plan.

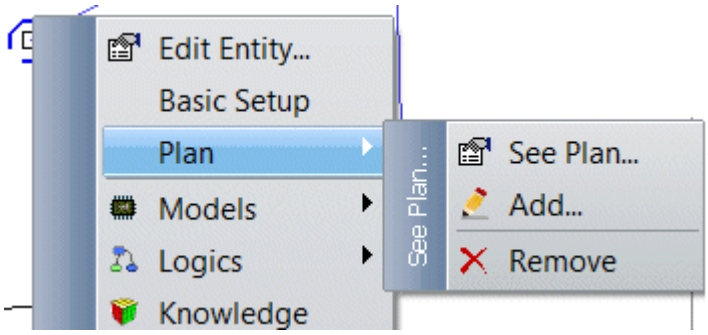



- First, create a plan by selecting one Entity, then right click:




Once the mouse cursor changes to + select on the map the first plan-point to create.

- Once a Plan is created, you can:

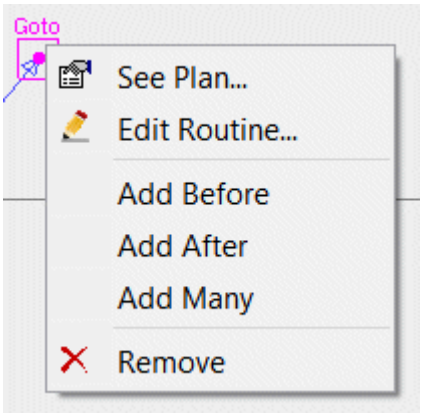



**See Plan**: property window of the Plan

**Add**: append a new plan-point at the end of the actual plan. Click on the map to create.

Remove: delete the whole plan.

- Any plan-point can be selected, displaced, edited or deleted. Right click the plan-point:




**See Plan:** property window of the Plan

**Edit Routine**: property window of the plan-point routine

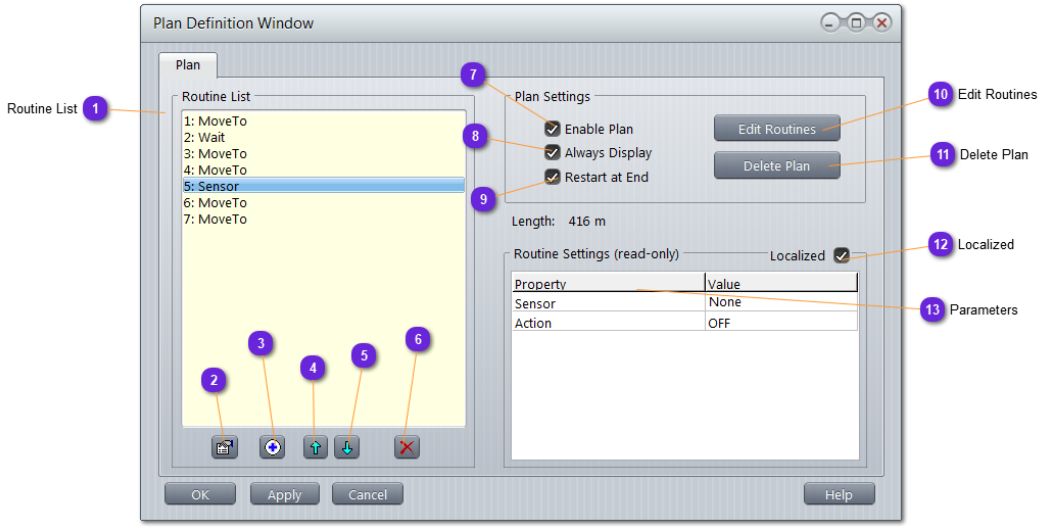
**Add Before**: insert a new plan-point before the selected one (click on the map to create)

**Add After**: insert a new plan-point before the selected one (click on the map to create)

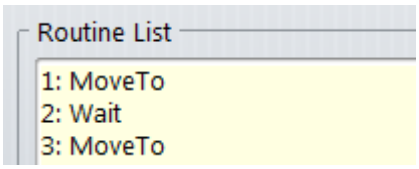
**Add Many**: duplicate the selected one as many times as clicked on the map. Right click and Done to stop or use keyboard ESC key.

**Remove**: suppress the selected plan-point

# Plan Definition




## 1 Routine List




List all routines from first plan-point to last one, that are making the actual plan.

## 2 Properties

Display the plan-prop editor window for the selected routine.

## 3 Add

Add/Insert a plan-point after the select one in the list (or at the end if none selected).
The plan-point position will be the one of the selection (if inserted) or the last plan-point of the list (if added).

## 4 Move Up

Move the selected plan-point one position up (towards the beginning of the plan).

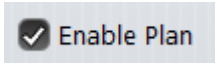
## 5 Move Down

Move the selected plan-point one position down (towards the end of the plan).

## 6 Delete

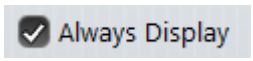
Remove the selected plan-point.

## 7 Enable



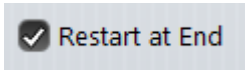

Activate the current plan when checked.
If unchecked, the plan will not be followed by the entity at start.

## 8 Display



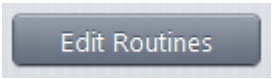

If checked, the plan will always visible on the map.
If unchecked, the plan will only be visible when the associated entity is selected.

## 9 Restart



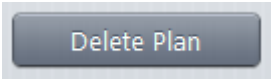

If checked, the plan will loop after the last routine leaves.
If unchecked, the plan will normally finish at the last routine.

## 10 Edit Routines



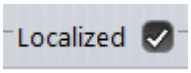

Call the routine editor window.

## 11 Delete Plan



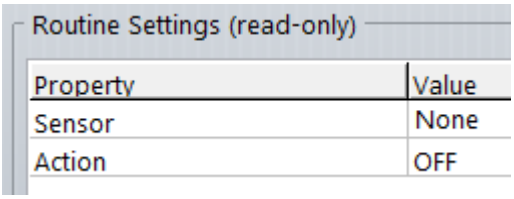

Removes all plan-point and delete the whole plan.

## 12 Localized



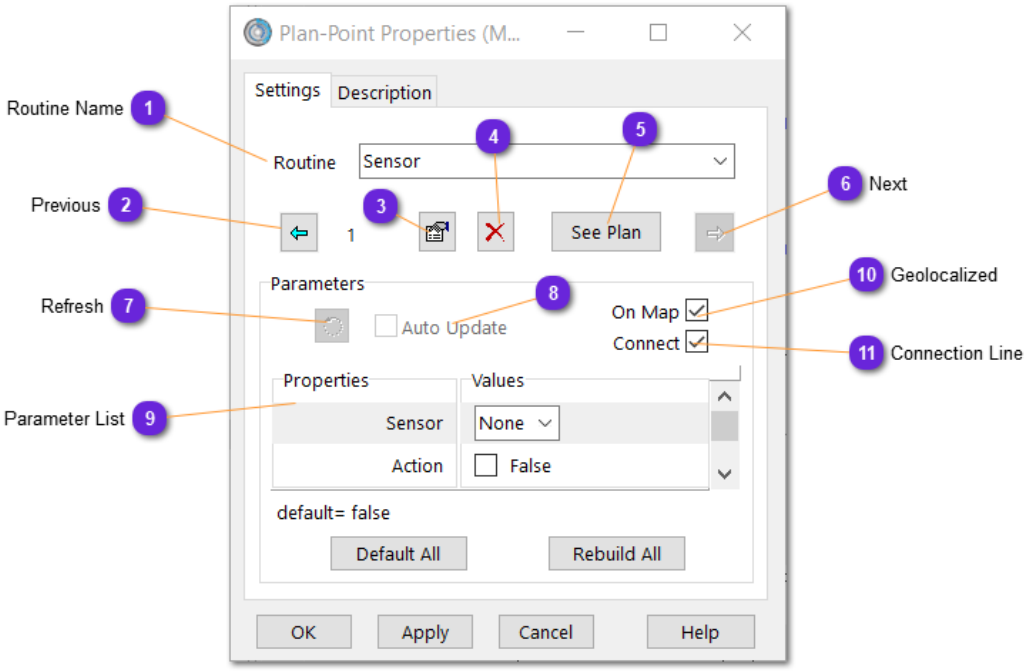

Same as On Map in the plan-point editor.

## 13 Parameters




Data interface of the selected routine. User can modify the values in this window.

# Plan-Point




## 1 Routine Name


Select the routine to attach to the actual plan-point.
The list is extracted from the actual routine list attached to the entity class.

## 2 Previous

Switch to the previous plan-point (if any)

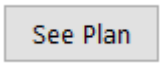
## 3 Properties

Show the routine property window for editing

## 4 Delete

Remove the actual plan-point from the plan.
If the plan-point was the last one, the plan is also deleted.

## 5 Plan Window




Display the actual plan window for editing. See here.

## 6 Next

Switch to the next plan-point (if any)

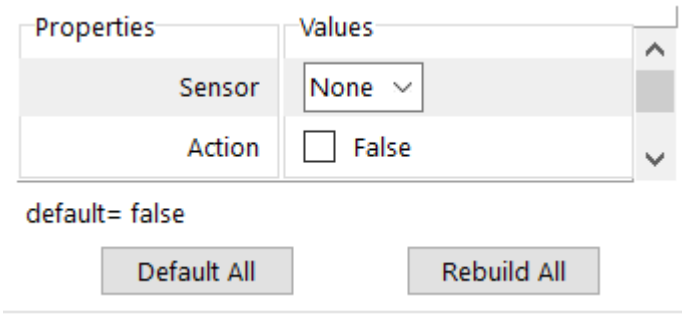
## 7 Refresh

Ask the simulation engine to refresh the routine values (9) with the runtime ones.

## 8 Auto Refresh


Force the refresh at 1hz for all time the window is kept opened.

## 9 Parameter List




List all the routine parameters defined as Dyn-UI. See here to know how to use.

## 10 Geolocalized

On Map ☑

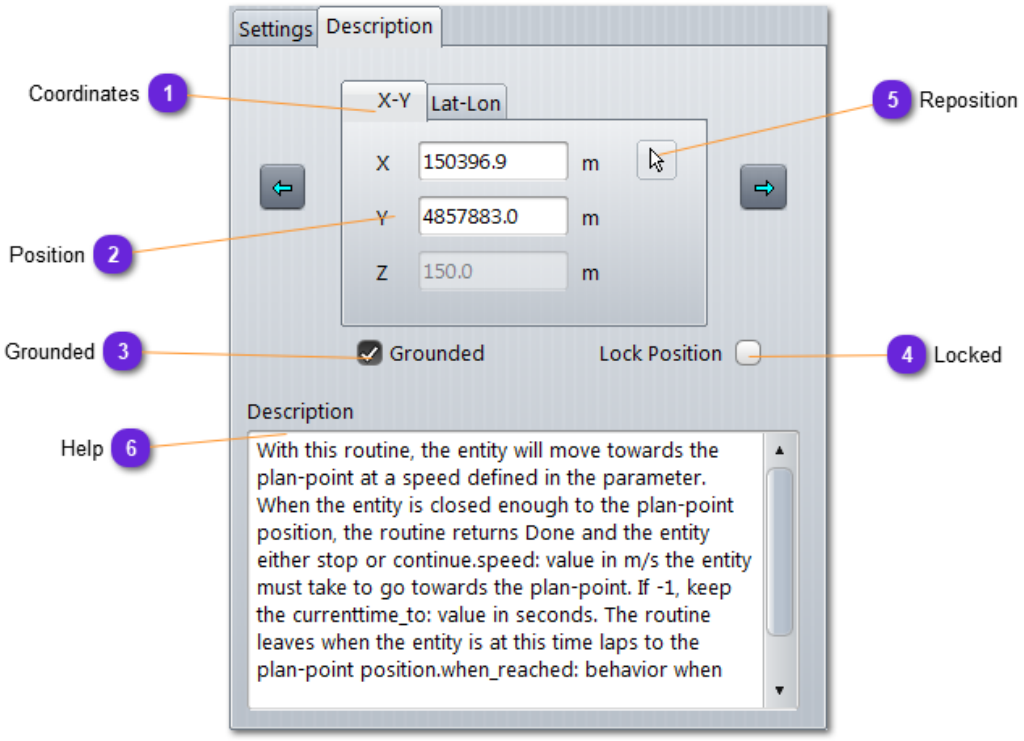
When checked, the plan-point will be geolocalized (and selectable) on the map and the routine will have the capability to retrieve its position.
This is mandatory for some routines that needs to compare the actual entity position with the plan-point position to control the navigation.
Conversely, a routine that will only activate some components of the entity will not need to be geolocalized.

## 11 Connection Line

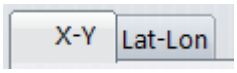
Connect ☑

When checked, the plan-point will be **visually** connected to the previous plan-point (on the map).

# Description

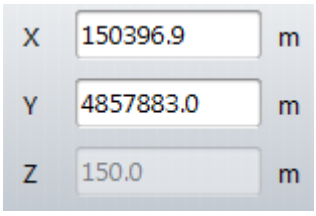



## 1 Coordinates




Select here the tab preferred for the coordinates units.
XYZ or Latitude, Longitude, Altitude

## 2 Position

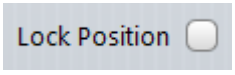



When the routine is localized on the map, the position is displayed here.
This position is available on the routine code using the `WCoord coords` variable.

## 3 Grounded

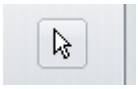

If checked, the routine is **clamped** on the ground below it.
Otherwise, the altitude can be set manually (for aircraft or helicopter).

## 4 Locked



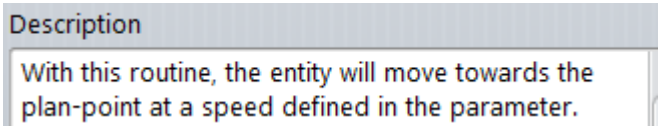

If checked, the position cannot be modified using the mouse. This insure that the plan will not accidentally be modified.

## 5 Reposition



Use this button than click on the map to update the routine position.

## 6 Help




Display the help part of the routine, as defined in the routine definition.

*Read-only text.*

# Routine Properties

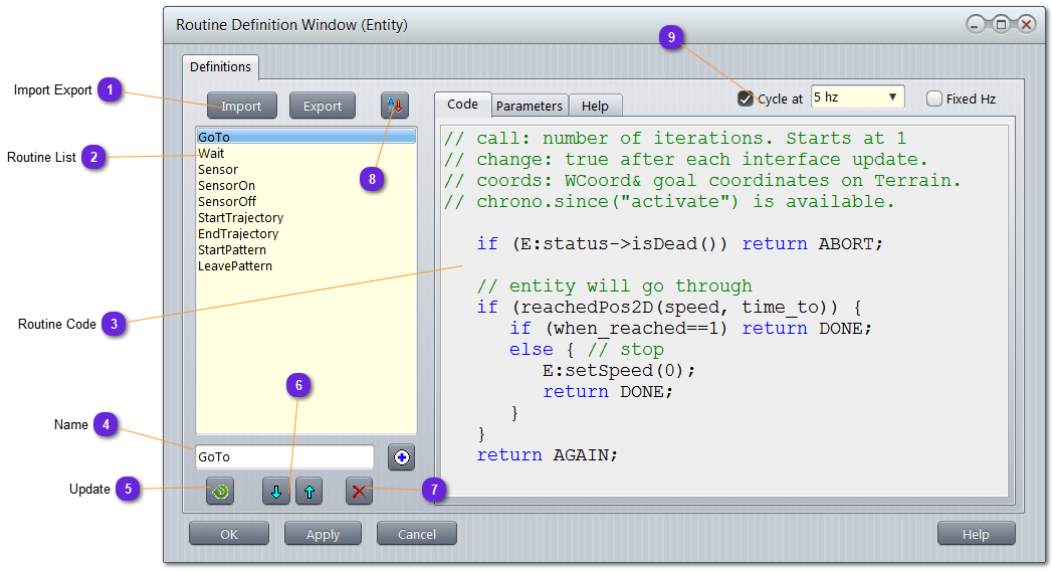
Routines are named functions that can be given quickly and easily to entities during design or runtime. They are used in Plans.
A routine is normally geolocalized (associated with a position on the map), like MoveTo.
Then, selecting an entity and giving the routine MoveTo with a position on the map will immediately triggers the MoveTo code and make the entity moves towards the position.

Routines are mostly user-defined although vsTASKER comes with built-in routines (refer to the *Developer Guide* for the list). There is no limitation in the number or complexity of a routine as long as the designer follows basic runtime coding rules, meaning that the routine does not capture too much processing time at each call to not slow down the simulation frame rate.

A routine can be replaced with a logic or a component model but offers a more convenient way to associate sequence of actions and terrain coordinates.

Refer to the *Developer Guide* and the *Tutorial Manual* to learn the built-in routines and how to create your own.

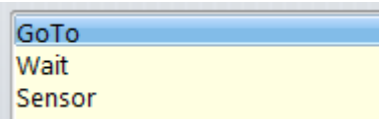
*Routines are piece of C++ code and are saved into the database. If you write some general purpose routines, try to export them into Data/Shared/Routines directory for later use or add them into you favorite database template to always have them listed.*



## 1 Import Export

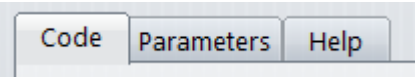

Routines can be **Imported** or **Exported** one by one.
To import, just select one from the Data/Shared directory (or anywhere else).
To export, select one from the list, use the Export button, then give the routine a name.

## 2 Routine List




All database defined Routines are listed here. They also belong to the Template.

# 3 Routine Code




## Code

Enter here the C++ code that will be called repetitively at the specified frequency, as soon as the Routine will be triggered.
Returning AGAIN insure that the piece of code will be called again at the next cycle (if cycle at selected)
Returning DONE will stop the Routine to run.

Have in mind that loops or complex computations at every cycle might impact the overall CPU performance. If you must parse a long list of data, think about cutting it into chunks (10 or 100) at each cycle to distribute the load on several cycle and not on time (to avoid CPU peak like regular beat with visible impact on visual for example).

## Parameters

Define here variables used by the Routine.
If you want the user to change the value using the automatic interface generator, use `//&&` after the definition:
ie:

```
int   my_data;
float my_speed;  //&& DEF[20]  UNIT[m/s]  HINT[Speed]
```

Here, `my_speed` will be declared as a public variable with an entry in the Routine Dyn-UI `my_data` will remain private.

## Help

Add here all information needed by the end user to know how to use the Routine and what are Parameters used for.

# 4 Name


To **create** a new Routine, set the name in the field and press ⊕ button

## 5 Update

To **rename** a Routine, select it in the list, change the name then use this button.
You can also use it to force the update of the routine with the *Code/Parameters/Help* text values.

## 6 Reorganize

Select one routine from the list and use the up or down arrow button to move the routine in the list.

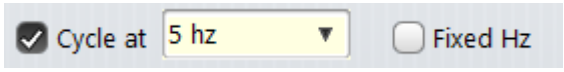
## 7 Delete

Select the Routine to delete and press this button.

## 8 Sort

Sort alphabetically all routines.

## 9 Cycle at




Select here the frequency at which the Routine shall be processed.
If the checkbox is not set, base RTC frequency is used by default.
When the checkbox is ticked, the frequency of the dropdown list will be used.

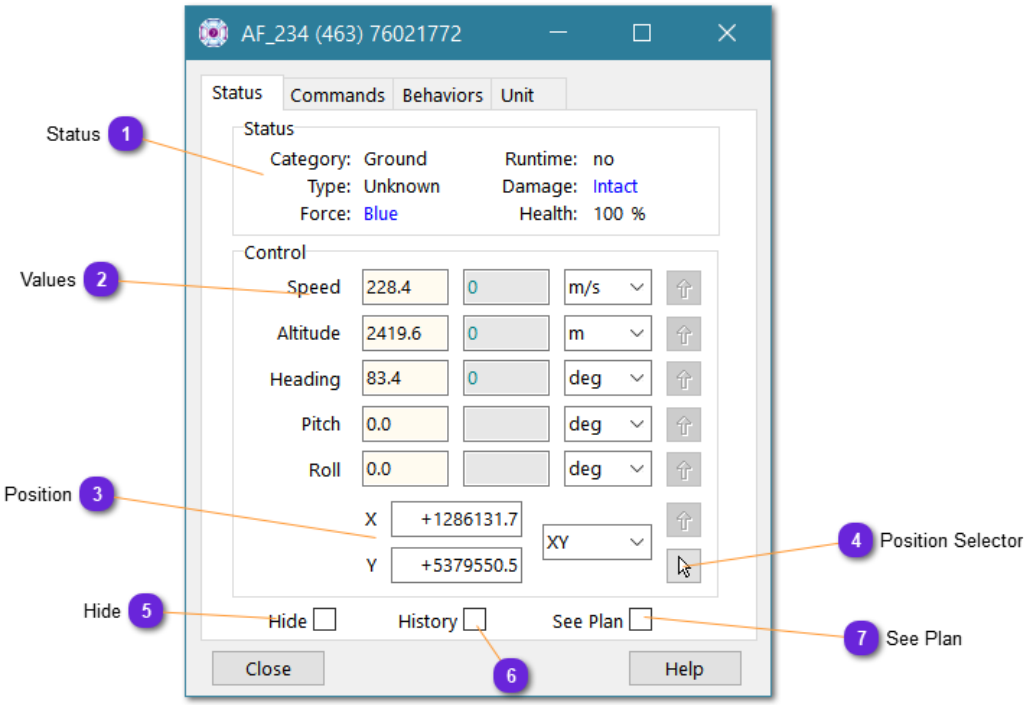
If **Fixed Hz** is selected, the frequency will be the wall clock frequency and will remain steady even if the simulation speed is changed.

# Hook Window

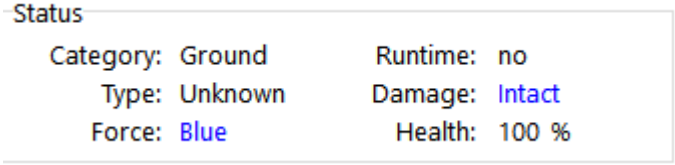
At runtime, it is possible to control entities from a hook window.
Several hook windows can be displayed at the same time.

To open a hook window, double click the entity or use the right click menu.

# Status




## 1 Status

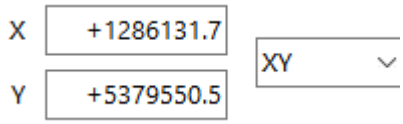



Real time status of the hooked entity. Read only

## 2 Values


Fields (Speed, Altitude, Heading, Pitch, Roll) display current and target values of the hooked entity, in the unit chosen in the selector.
To **change** the value, just click on the grayed target field, enter the new target value, then press ⇧.

## 3 Position

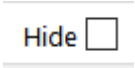



These two fields display the ground position of the hooked entity (in X-Y or Lat-Lon or decimal Lat-Lon).
To **relocate** the entity, just click on one text field (both update will stop) and enter the new target value, then press ⇧.

## 4 Position Selector

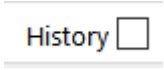
For immediate relocation of the entity, use this button, then click on the map to grab the position (fields are updated) then use ⇧ to validate the change.

## 5 Hide



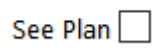

Check/uncheck this option to hide/show the entity on the map.

## 6 History



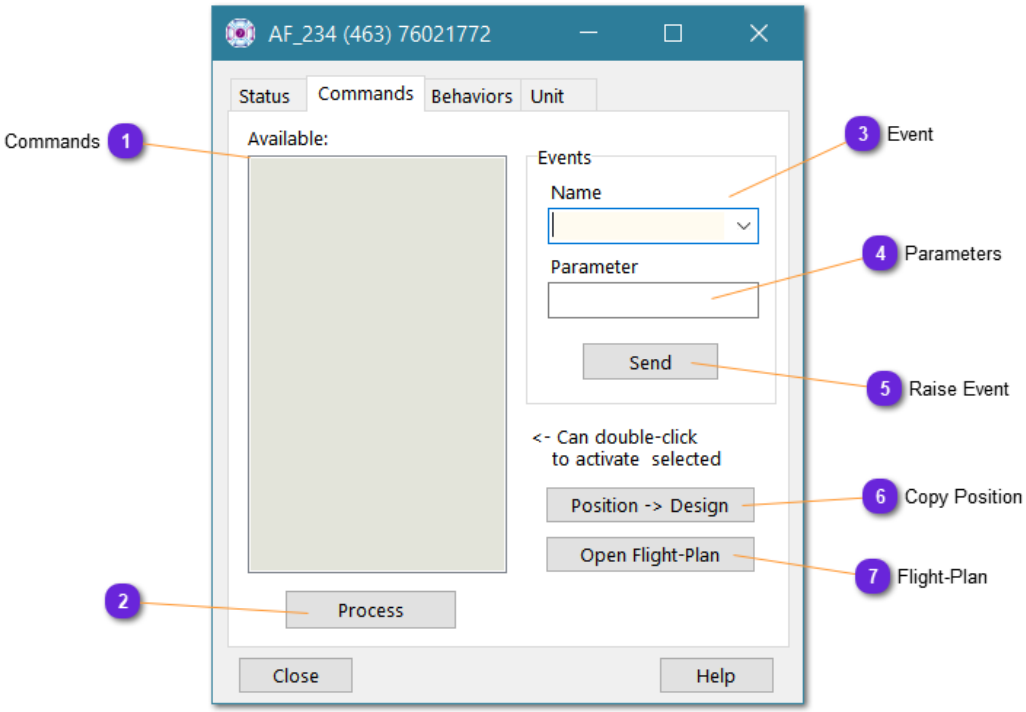

When checked, the history track will start (or restart) and will be displayed on the map (if entity moves).
When unchecked, history track is removed (and cleaned).

## 7 See Plan

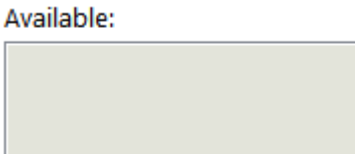



Check/uncheck this option to hide/show the entity plan (if any) on the map.

# Commands

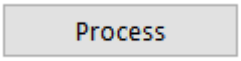



## 1 Commands



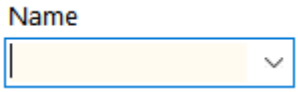

List of all available commands for the hooked entity.

## 2 Process



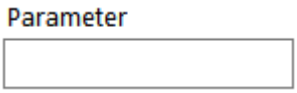

Select a command in the list then press Process to activate it (if actions are requested by the command, like selecting on the map, the cursor will change to a +. If values must be entered, a text field window will appear, otherwise, command will immediately by processed).

## 3 Event




Name of the event to raise at the entity scope.

## 4 Parameters




Optional parameters to attach to the event.

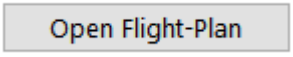
## 5 Raise Event


Click this button to raise the above event.

## 6 Copy Position

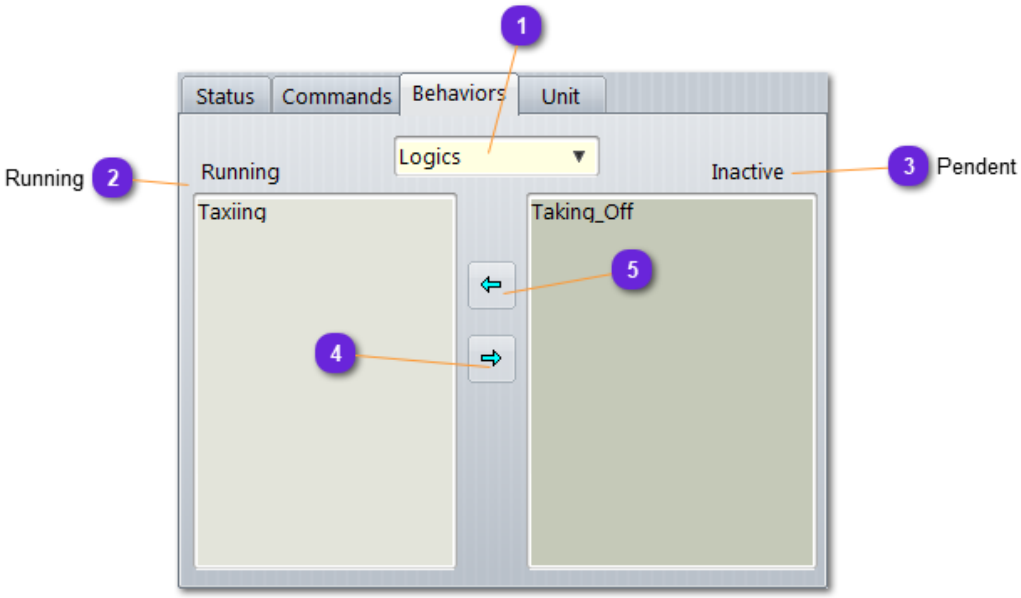

Press this button to copy the current runtime position of the entity to the design counterpart.
This functionality is also available from the runtime entity popup menu.
Very useful to accurately position an entity using a 3D viewer only available during runtime.

## 7 Flight-Plan




If the entity is flying a flight plan, use this button to display the Flight Plan property window for runtime.

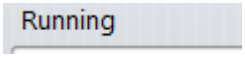
# Behaviors




## 1 Selector

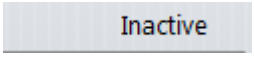

Select here what behavior category must be displayed on the lists below.
Logics, Knowledge, Models.

## 2 Running




List of all actual running behavior object for the selected category.

## 3 Pendent




List of all actual stopped or not active objects for the selected category.


*Disabled objects are not listed here.*

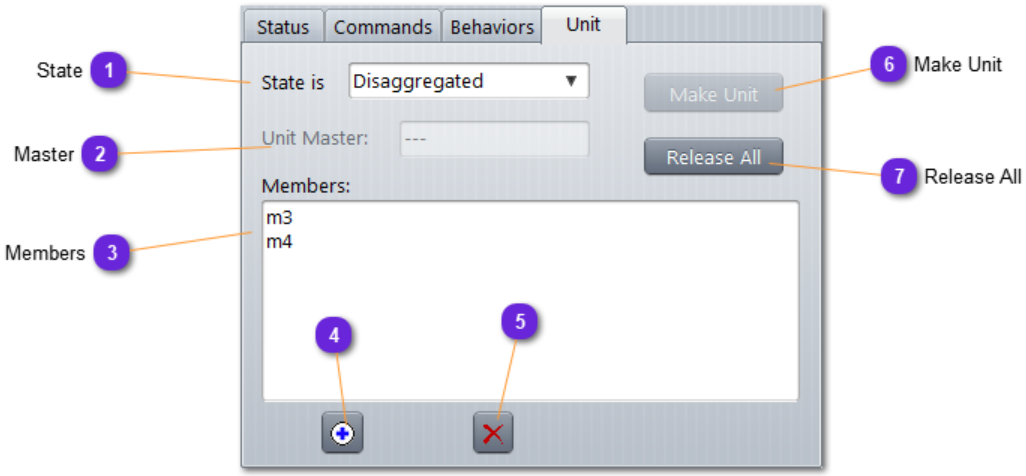
## 4 Deactivate

Move to the inactive list the selected Running behavior. Equivalent to stop.

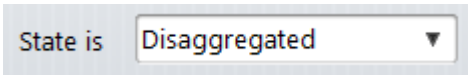
## 5 Activate

Move to the running list the selected Inactive behavior. Equivalent to start.

# Unit




## 1 State



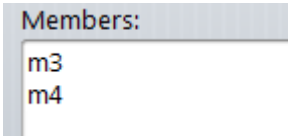

Shows the status of the master (unit).

- Aggregated: all members are hidden on the map
- Disaggregated: all members are visible on the map (and simulated).

## 2 Master


Read only field that gives the name of the master (unit) if hooked entity is a member. Empty (---) otherwise.

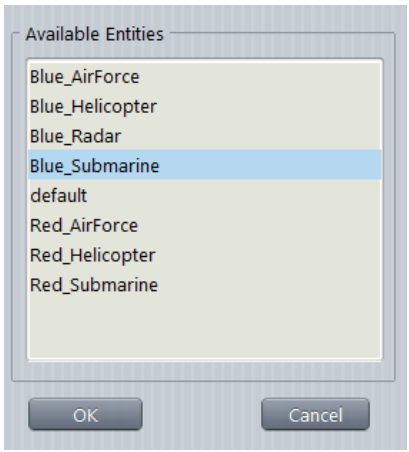
## 3 Members




List of all entity members of the Unit (blow the actual master).

## 4 Add Member

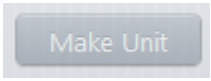
Use this button to select from a list of available entities (see image below) which one to add as a member of this unit.
The list of proposed entities does not belong to any unit (but can be unit master).




## 5 Remove

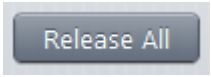
Select one member in the above list and use this button to exclude it from the unit (will not delete the entity).

## 6 Make Unit




If the hooked entity is not a Unit, depress this button to make it unit Master (then start adding members).

## 7 Release All

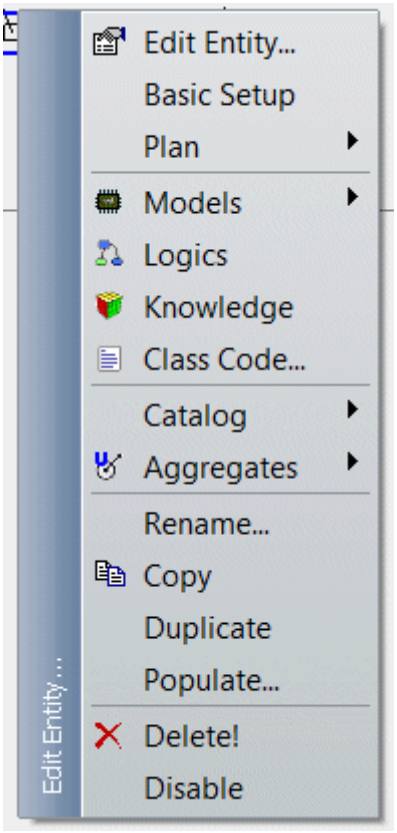



Use this button to free all members from the current Unit. The hooked entity will no more be a unit master (but can still remain member of another unit).

# Popup Menus

You get these popup menus by selecting the Entity on the map and mouse right click:

## • At Design




**Edit Entity**: display the property window.

**Basic Setup**: same as using tool . See here (end of page).

**Plan**: create or edit the entity plan.

**Models**: display the interface setting of all attached components or data-models.

**Logics**: display the interface setting of all attached logics.

**Knowledge**: display the interface setting of all attached knowledge.

**Class Code**: display the entity class code.

**Catalog**: copy or move the selected entity to the catalog list.

**Aggregates**: see here.

**Rename**: change the name of the entity.

**Copy**: copy the entity into the clipboard.

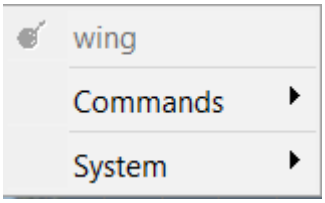
**Duplicate**: copy the current entity and paste it aside.

**Populate**: see here.

**Delete**: suppress the entity from the scenario.

**Disable**: make the entity inactive (disabled entities no more exist for the simulation even if they have a memory footprint and a data persistency in the runtime engine).

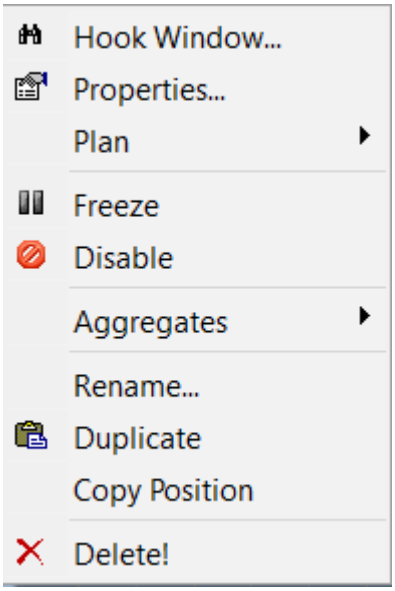
## • At Runtime




**Name**: of the entity, not selectable

**Commands**: list all user-defined commands

**System**: see below




**Hook Window**: display the hook window.

**Properties**: display the property window (read-only during runtime).

**Plan**: allow runtime creation and edition of a plan during runtime.

**Freeze**: stop all running behaviors of the entity (logics, knowledge and models)

**Disable**: stop and virtually removes the entity from the scenario (disabled entities no more exist for the simulation even if they have a memory footprint and a data persistency in the runtime engine).

**Aggregates**: see here.

**Rename**: change the name of the entity.

**Duplicate**: will copy and paste the selected entity.

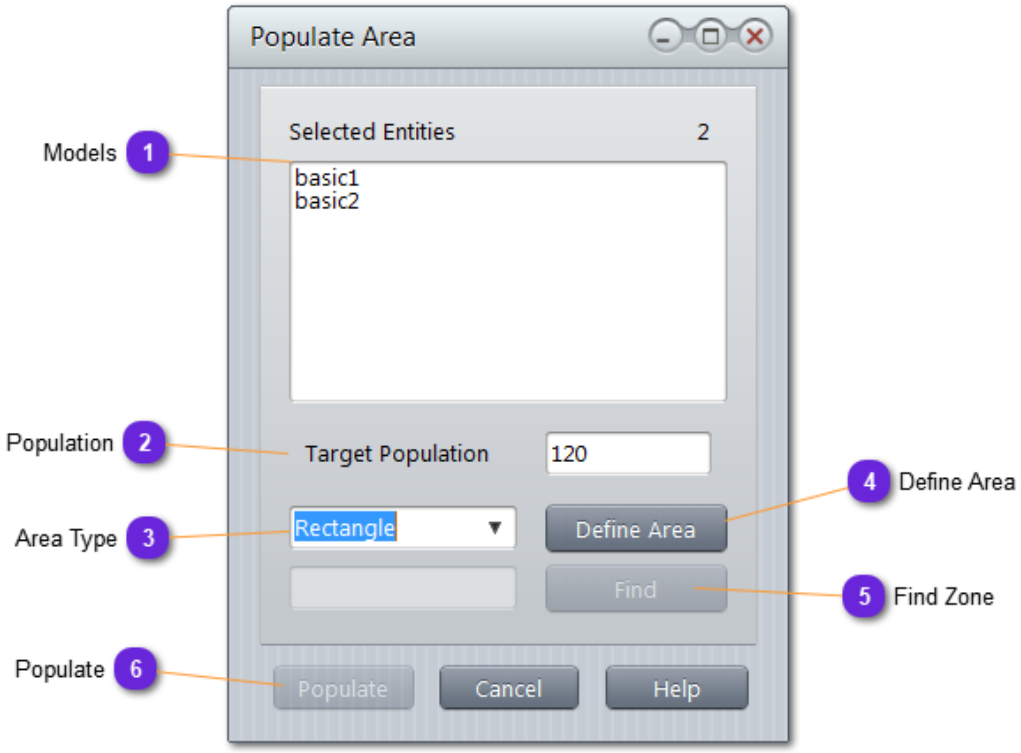
**CopyPosition**: copy the actual position of the entity to the design entity counterpart.

**Delete**: remove the entity from the scenario (and from the runtime engine).

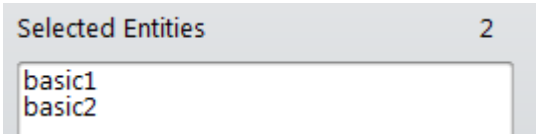

*All modifications in the scenario at runtime will have no effect on the design counterpart (except for CopyPosition). At simulation stop, the scenario is reverted with design values.*

# Populate

Populate window is a handy way to duplicate and distribute randomly one or several entities on a user-defined area.
To access the feature, user must select one or several entities then, right click and select Populate function.

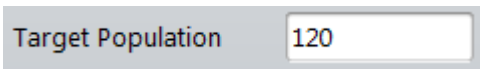



## 1 Models



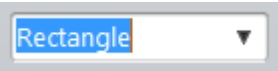

Lists the entity that will be cloned.
If several selected, cloning will select randomly any of them at each iteration.

## 2 Population




Specify how many clones of the selected models will be generated in the area.

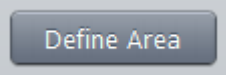
## 3 Area Type




Chose in the list the shape of the area that will be populated.

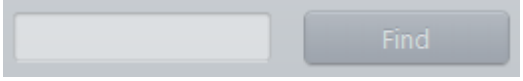
- Segment: clones will be distributed along a single line
- Rectangle: clones will be distributed into a rectangle horizontal area
- Circle: clones will be distributed into a circular area
- Special Zone: clones will be distributed into the shape(s) of the selected special zone (field below) that must exist (5)

## 4 Define Area



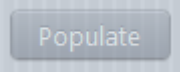

Use this button to define the area on the terrain map.
Populate window will be hidden until the area is defined (mouse change to +. Click on the map than with mouse down, define either the second end of the segment, or the bottom right corner of the rectangle, or the radius of the circle.

## 5 Find Zone




When type is defined as Special Zone, specify in the field the name of the special zone and use the Find button to retrieve it (shall be done as a test)

## 6 Populate




Once properly set, the button will be activated. Press it and see immediate response on the terrain.
Every time the button is depressed, target population will be created in the defined area.
Close the window when done.

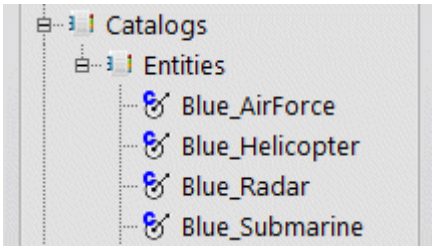
# Catalogs

**Catalogs** are a list of predefined templates, belonging to the database, whose main purpose is to be duplicated and copied into scenarios.
At the moment, only one catalog type is defined: **Entity**.

Although it is not mandatory to use catalogs at design time, as an entity can be created from a template or imported from the shared directory, using catalog entities is mandatory at runtime: adding a new entity into the scenario at runtime can only be done using duplication of an existing or catalog entity.

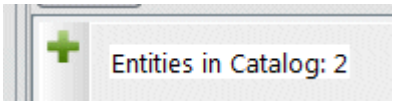
## • Environment

To access the catalog list, expand it in the Environment tree-list panel.

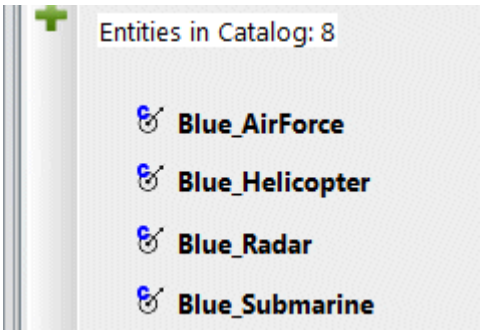



According to settings, catalogs can belong to the database (one for all scenarios) or to each scenario (catalog list will change with scenario focus).

To add a new entity in the catalog list, select Entities and use the ✚ button of the tool bar:

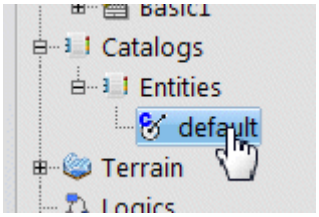


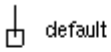

Entities belonging to the catalog are listed into the Diagram panel:



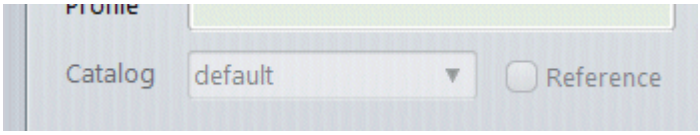
## • Drag & Drop

To move an entity from the catalog to the current scenario, just select the entity in the list and drag it into the scenario (while holding down the mouse button) at the desired position.
Release the mouse to instanciate.




When an entity is coming from a catalog (copied from), it will appear as such in the property window:




*For the moment, it is not possible to make it as a reference to a catalog entity. This will come in a future version.*

## • Design Popup

In the scenario map, select any entity and right click:

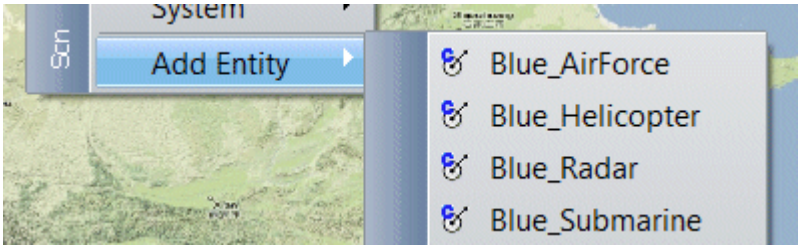

**Copy**: copy the selected entity into the catalog list, using the same name.

**Move**: do a copy first (see above) then remove the selected entity from the scenario. This is another way to build the catalog list.

## • Runtime Popup

In the scenario map, right click the background at the location you want to put a new runtime entity:

### Catalogs




**Add Entity**: at runtime, the only way to add a new entity is to instanciate it from the catalog list. Select the one you want, rename it and it will be created at the mouse location.

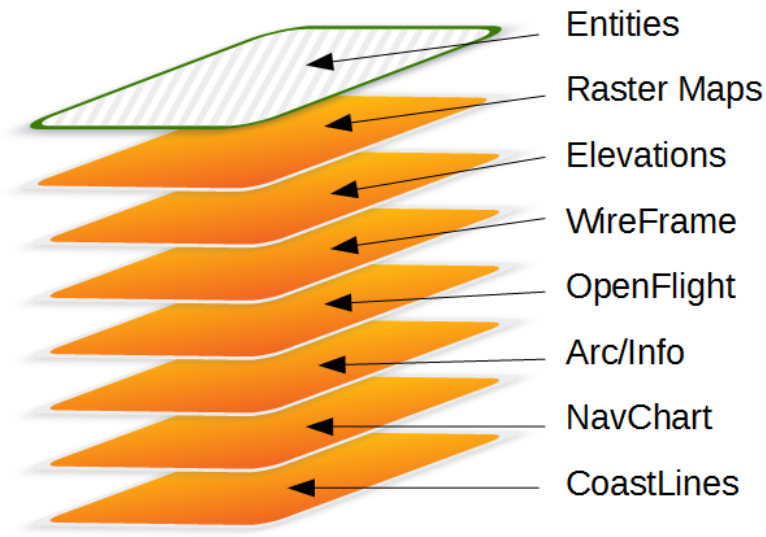
## • Runtime Code

Use such code into logic or any code using vsTASKER libraries to instanciate a catalog entity, at runtime, in the scenario:

```
 WCoord pos(WC_LLA, 31.911104, 115.653213, 3000);
 Vt_Entity* blue_airforce = new Entity("Blue_AirForce", "F16_1",
pos);  // catalog, instance name, position
```

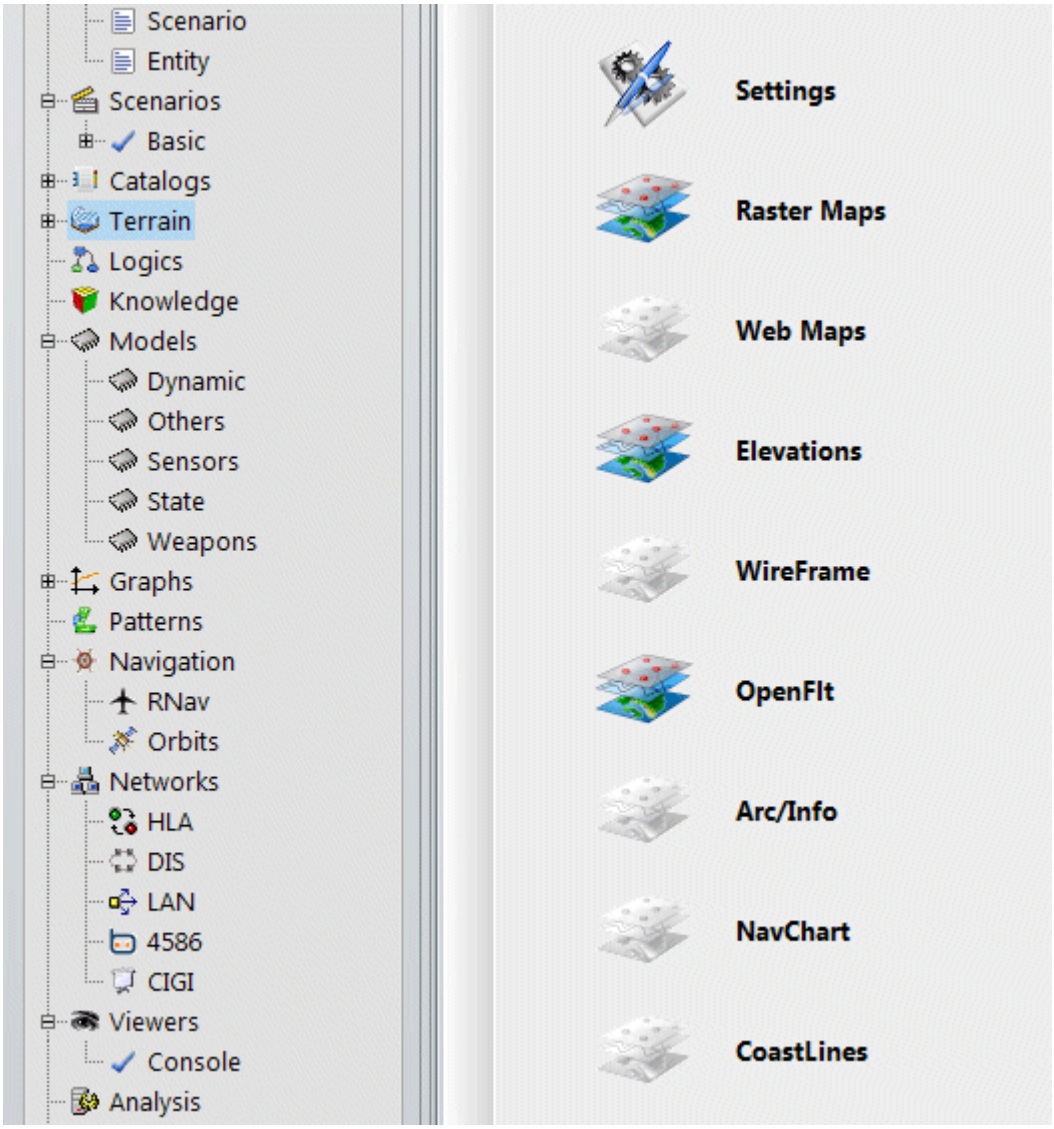
# Terrain Layers

Scenario map is made of a pile of layers.
The entity layer that has focus on Terrain Plan View
The terrain layer pile that is displayed from the first to the last one.

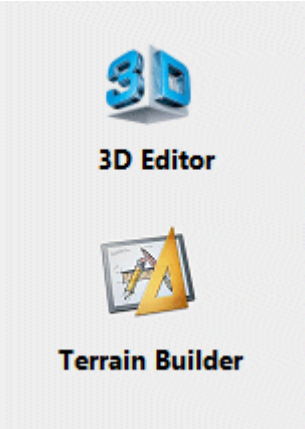



- **Environment**

### Terrain Layers



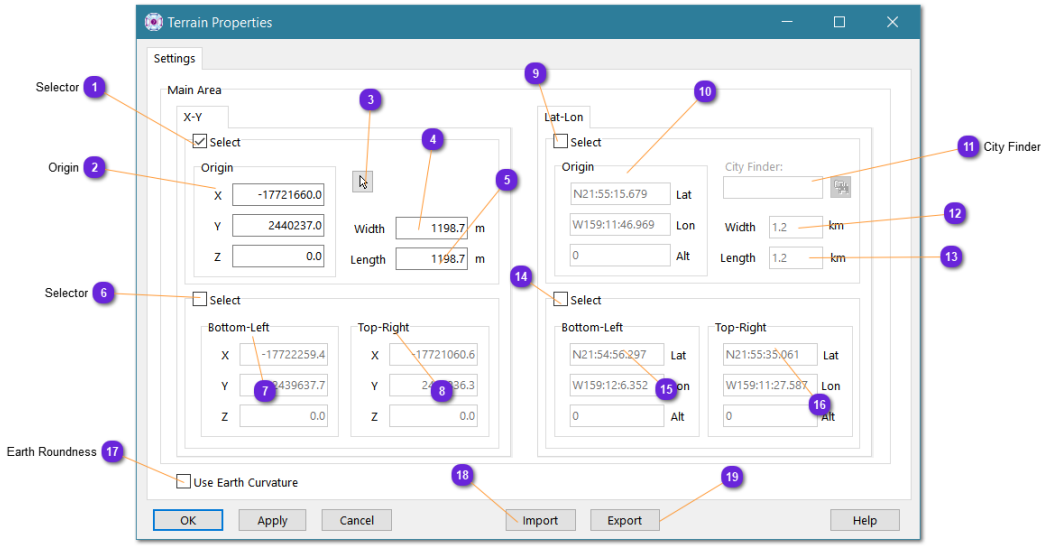

You can select any of the listed layers to display their property window. When the icon is grayed out, it means the layer is empty. When the name of the layer is grayed out, it is not available according to credentials or product version.



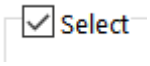

The 3D Editor icon is to set the OpenSceneGraph terrain viewer for design.
The terrain Builder icon is to call the external application to convert some terrain formats into proprietary vsTASKER formats.

# Settings

Use this window to setup the dimensions of the area covered by the scenario (in case of multiple terrains) or by the database (in case of unique terrain).

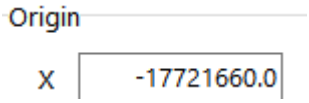



## 1 Selector




Use this selector for defining the terrain area by giving the **center** position in XYZ and width, height in meters.
This format is good for small to medium size areas that are not correlated with real map.

## 2 Origin




**Center** of the terrain, in meters.

## 3 Location Select

Use this button to manually recenter the terrain by selecting a new location on the map.
Window will hide, mouse cursor will change to +.
Click on the terrain map to get the new center position.

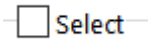
## 4 Width


**Width** of the terrain (east - west), in meters, centered on origin.

## 5 Height


**Height** of the terrain (north - south), in meters, centered on origin.

## 6 Selector




Use this selector for defining the terrain area by giving the **two corner** positions in XYZ. Width and height are automatically computed.
This format is good for small to medium size areas that shall be correlated with real map or position.


*Flat earth coordinate system is used to compute width and height.*

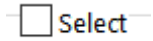
## 7 BL Corner


Specify in XYZ the coordinates of the **bottom-left** corner of the terrain (south-west).

## 8 TR Corner

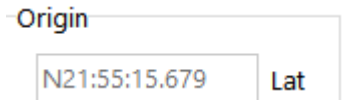

Specify in XYZ the coordinates of the **top-right** corner of the terrain (north-east).

## 9 LatLon Selector




Use this selector for defining the terrain area by giving the **center** position in latitude, longitude and width, height in kilometers.
This format is good for big size areas that are correlated with real map.

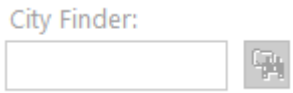
## 10 Origin




**Center** of the terrain, in latitude, longitude.

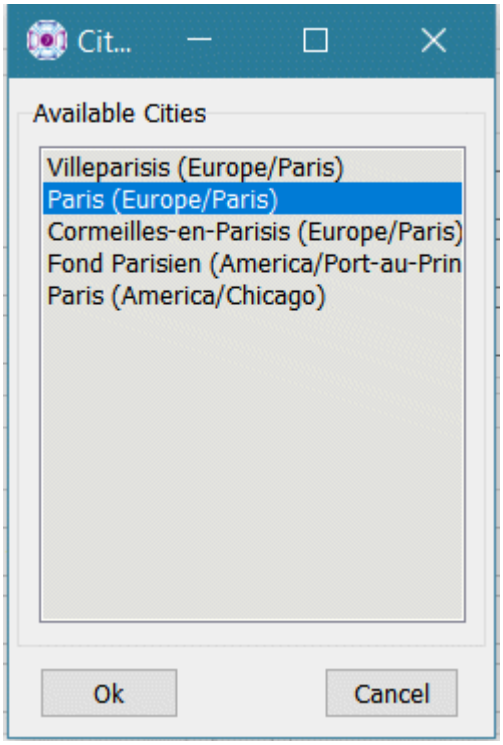

*Literal format (N/W/E/S000:00:00.0) or decimal format (0.00) can be used for Lat-Lon values.*

# 11 City Finder

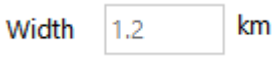



Sometime, we need to setup a scenario around a known city, mostly when WebMap is intended to be used. Instead of zooming and panning inside the world map (which takes time as tiles must be downloaded from Internet), better to set the terrain directly around a given city, with a predefined area (width and height), then Apply and switch ON the WebMap layer.
vsTASKER knows 15000 cities of the world, so, it is likely you will find yours.

Enter the name of the city in the field then use the button.
If the city is found, it will be displayed in the Available Cities list (window below). If several cities exist under the same name, just select the one you want and click OK.
The coordinates of the selected city will replace the data of the Origin fields.
Apply and OK.

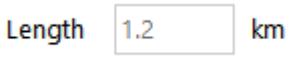


## 12 Width



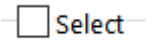

**Width** of the terrain (east - west), in kilometers, centered on origin.

## 13 Height




**Height** of the terrain (north - south), in kilometers, centered on origin.

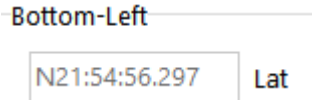
## 14 Selector




Use this selector for defining the terrain area by giving the **two corner** positions in latitude, longitude and altitude. Width and height are automatically computed.
This format is good for big size areas that shall be correlated with real map or position.

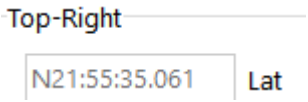

*WGS84 coordinate system is used to compute width and height.*

## 15 BL Corner



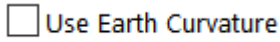

Specify in Lat-Lon the coordinates of the **bottom-left** corner of the terrain (south-west).

## 16 TR Corner



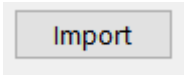

Specify in Lat-Lon the coordinates of the **top-right** corner of the terrain (north-east).

## 17 Earth Roundness



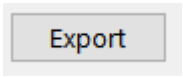

When set, earth curvature is used in line of sight computations by components and wherever it's needed.
From the code, use `terrain->useEarthCurvature()` to know if this option has been checked or not.

## 18 Import




**Import** all definitions of the terrain, including all the layers definition (see formats), previously saved in /data/shared.
This reduce the time of a setup or when keeping the same database but changing the terrain setting.

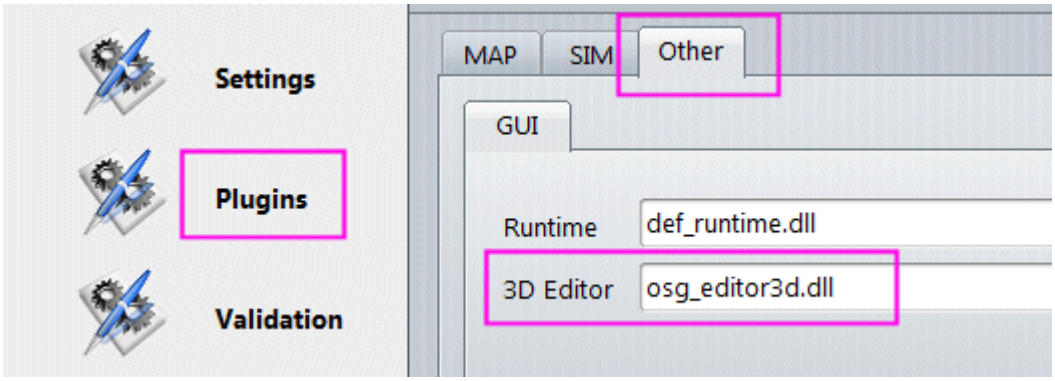
## 19 Export



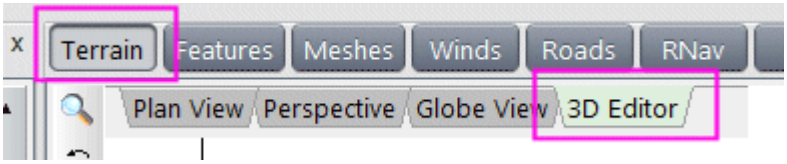

**Export** all definitions of the terrain, including all the layers definition (see formats), in /data/shared.

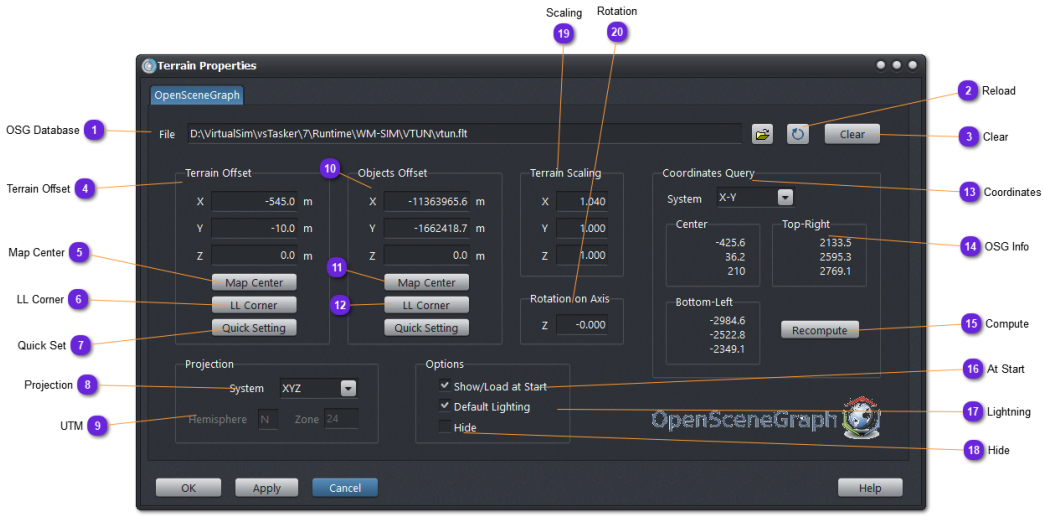
# 3D Editor

This setting window is for the 3D scenario editor that relies on the built-in plugin (def_3deditor.dll).
The OSG terrain database to be loaded here will also be used by the OSG Viewer (if defined).
First, make sure the plugin is set:

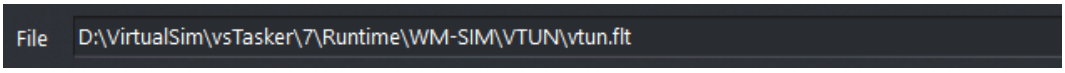



Focus on the 3D editor:



## 1 OSG Database

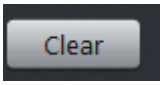



Select here the **OSG** terrain database in either ive or osg formats.

## 2 Reload

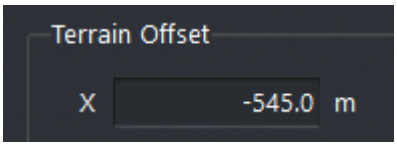
Clear the current OSG scene node and **reload** the selected database in text field (1) to rebuild the scene node.

## 3 Clear



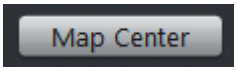

**Clean** the current OSG scene node.

## 4 Terrain Offset




Set here the terrain **offset** to apply to the current OSG database bottom-left corner.
This can be useful to offset to (0,0,0) an OSG terrain whose coordinates are huge (UTM for example) or to place a (0,0,0) OSG terrain to a specific XYZ coordinates terrain.
These values are used in the `setPosition()` function of the `osg::PositionAttitudeTransform` object holding the terrain node.

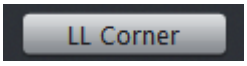
## 5 Map Center




Use this button to **compute** the **offset** to apply to the current OSG database to make it (0,0,0), relatively to the actual terrain center.


*The offset is applied on the lower-left corner of the terrain.*

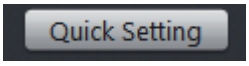
## 6 LL Corner




Use this button to **compute** the **offset** to apply to the current OSG database to make it (0,0,0), relatively to the actual terrain lower-left corner.

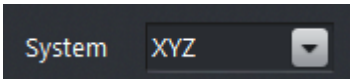

*The offset is applied on the lower-left corner of the terrain.*

## 7 Quick Set



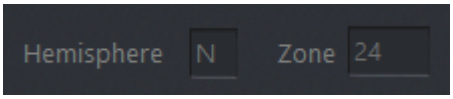

Use to visually reposition and orient the terrain.
For more details, go here

## 8 Projection



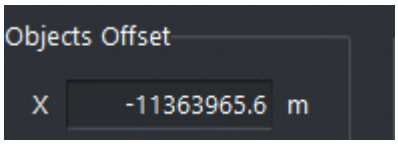

Set here the projection system used in the loaded database. This will insure correct correlations between the 3D world and the 2D layers of the Map terrain.

## 9 UTM




Only available when Projection is UTM

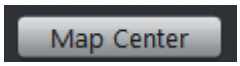
## 10 Objects Offset




Set here the objects **offset** to apply to all entities and object coordinated from the scenario in order to position them into the OSG database.
For example, if the scenario terrain origin is (-100,100,0) and the OSG database origin (bottom-left corner) is (1000,1000,0), then, all objects will have to be offset by (1100,900,0) to appear on the 3D scene at the correct position.
These values are used in the `setPosition()` function of the `osg::PositionAttitudeTransform` object holding the 3D model node.

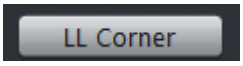

*If the OSG terrain is offset, then the objects normally do not have to be also offset. It is often one or the other.*

## 11 Map Center



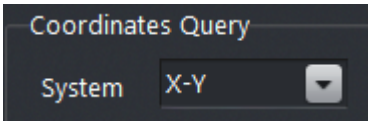

Use this button to **compute** the **offset** to apply to all entities and objects to make them (0,0,0) on the OSG terrain, relatively to the actual terrain center.

## 12 LL Corner




Use this button to **compute** the **offset** to apply to all entities and objects to make them (0,0,0) on the OSG terrain, relatively to the actual terrain lower-left corner.

## 13 Coordinates



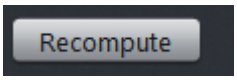

Choose the coordinate **system** for displaying the OSG terrain database coordinates:

- XYZ
- Lat-Lon literal
- Lat-Lon decimal

## 14 OSG Info


Display here the actual **OSG** database **coordinates** for the center of the scene, the bottom-left and the top-right corners.

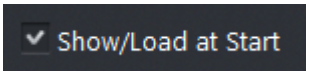
## 15 Compute




Extract from the OSG database the coordinates to display in (9).

*The scene is loaded into a special process that is called by vsTASKER. Wait for the process to finish. Sometimes, the process crashes but the extracted coordinates are correct and will be shown in (9).*

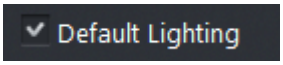
## 16 At Start




If this is checked, the OSG database is loaded and the scene is created in the plugin. In the Terrain Map, 3D View will automatically be selected so that the 3D editor will be shown as soon as the scenario is loaded.

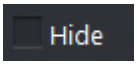
*If unchecked (default), the scene node is created only when the 3D Editor view is selected an plugin loaded.*

## 17 Lightning



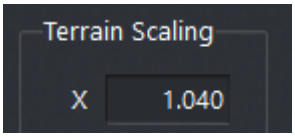

When checked, the terrain light will be set to default uniform ambient light. When unchecked, a spot light will be positioned 1000 m above the terrain center.

## 18 Hide



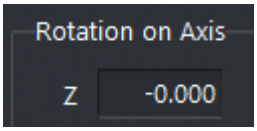

If checked, the terrain will not be visible (but entities and other objects will be).

## 19 Scaling



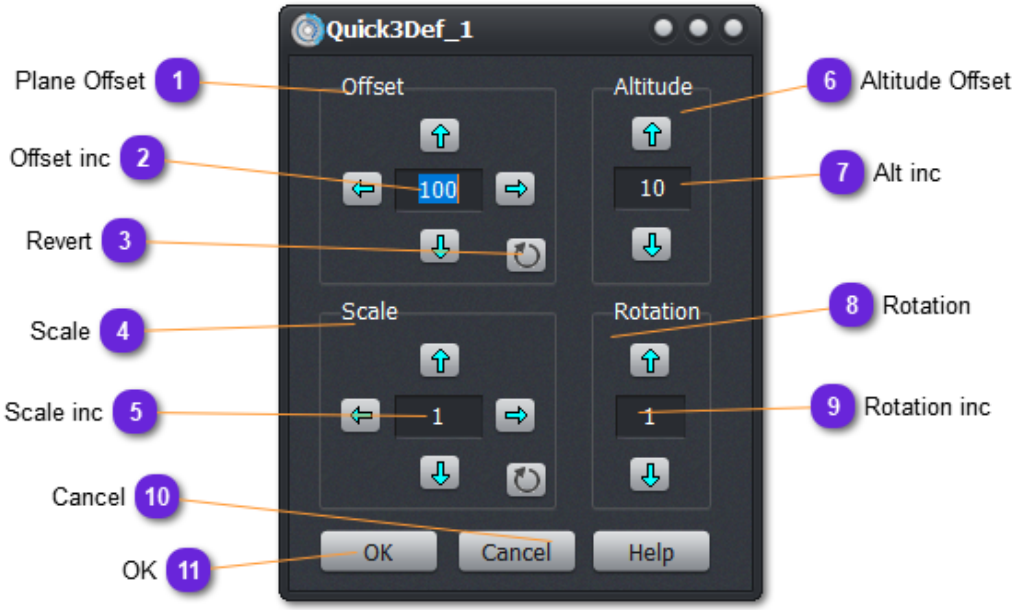

Display the node object scaling on the three axis.
These values are used in the `setScale()` function of the `osg::PositionAttitudeTransform` object holding the terrain node.

## 20 Rotation




Display the node object rotation in degrees around the Z axis.
These values are used in the `setAtt()` function of the `osg::PositionAttitudeTransform` object holding the terrain node.

# Quick 3D Setting




## 1 Plane Offset


Use the four buttons to displace the terrain along the X (up and down buttons) and Y (left and right) axis.
Each press on the button add or subtract the offset value.

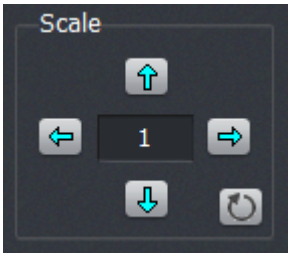
## 2 Offset inc


Offset increment in meters

## 3 Revert

Use this button to revert to the Offset or Scale saved values (when windows was opened)

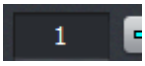
## 4 Scale




This panel controls the scaling of the terrain on the X (up and down buttons) axis and the Z (left and right buttons).
Each click on each buttons adds or subtract the scale inc value.


*Value cannot be negative and cannot drop below 0.01*

## 5 Scale inc

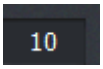



Scale increment value to be added to the current value. If 1 is used, scale will change like 1, 2, 3, etc.

## 6 Altitude Offset


Use the two buttons to move the terrain up or down along the Z axis.
Each press on the button add or subtract the offset value.

## 7 Alt inc




Altitude increment in meters

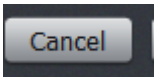
## 8 Rotation


Use the two buttons to rotate the terrain around its center. Up arrow increases angle, down arrow reduces, by the offset value.

## 9 Rotation inc

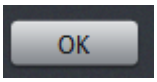

Rotation increment in degrees

## 10 Cancel




Revert to the old values and cancel the settings.

## 11 OK




Validate the setting

# Formats

vsTASKER only support the following formats.
Terrain formats are proprietary once converted by the TerrainBuilder tool.
Layers can directly load raw format files, at design, from the GUI and at runtime using the `vt_terrain.h` API.

*Proprietary format shall be privileged over raw formats as conversion is not necessary, thus, processing time is faster.*

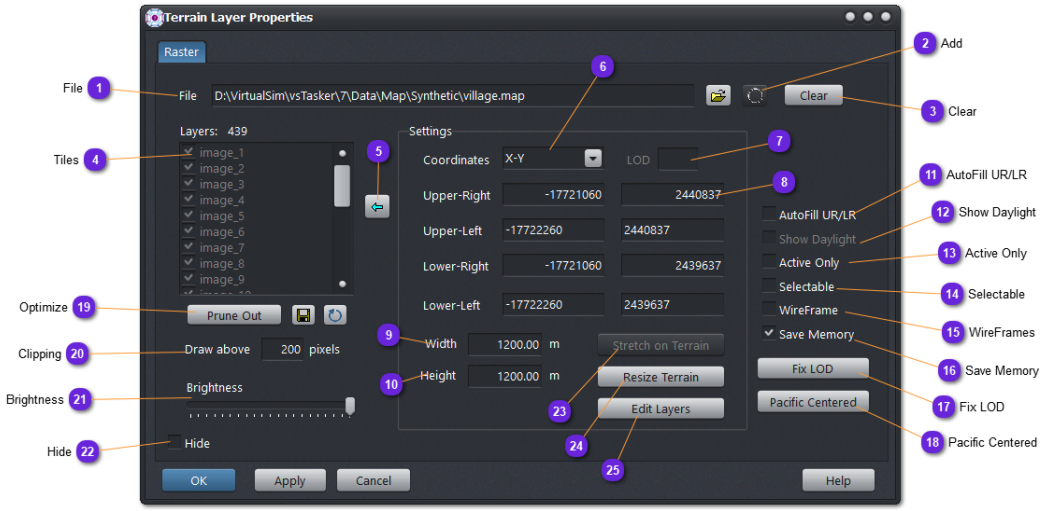
# Raster Maps

Raster maps are made of **textured tiles**.
The simplest map is made of only one tile that can cover the full or a part of the terrain area.
A tile is made of four corners (flat) that can shape a four sides polygon. For this simple map, several texture/raster formats are supported.

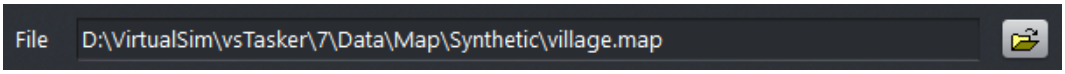
Tiled maps are another format provided by vsTASKER. They are a collections of textured tiles, organized as layers (based on LoD) that are automatically displayed and hidden according to the zoom level.


*Raster maps do not model any elevation. They are used with flat terrains.*

# Properties




## 1 File




Name of the raster **file** or tiled map.
Supported formats:

- map: tiled maps, proprietary format from TerrainBuilder converter tool.
- bmp,
- gif,
- tga,
- png,
- jpeg: raw formats.

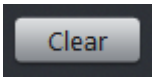
## 2 Add

Use this button to add another tile to the current list.

This addition must be save using button in order not to be lost.
The added tile must also be resized accordingly (8).

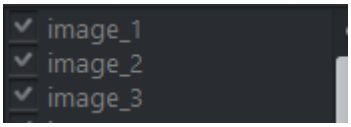
*Available only when a map is already loaded.*

## 3 Clear




Free the memory from the actual loaded raster map.

## 4 Tiles




If a tiled map is loaded in (1), list all the tiles used for the dynamic map.
If only one raster is loaded in (1), the unique tile is listed here.
When an entry is **checked** (x), it will be displayed on the terrain map.
When an entry is **selected** (i.e.: t_1024_1024), Settings box (right) will be filled with the current tile dimension (corners coordinates).

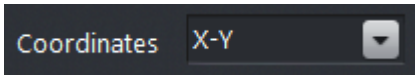

*Selected tiles are outlined in blue on the terrain map.*

## 5 Update

When data of Setting box has been changed, use this button to overwrite the selected tile with the modified data (lost otherwise).


*When the text fields are grayed out or read-only, user cannot change the values of the tile.*

## 6 Coordinates

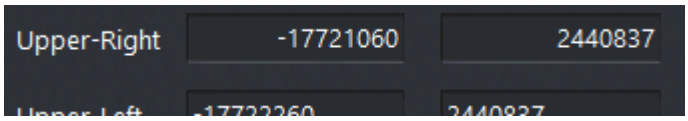



Select here the coordinate system format you prefer for the data of Settings box to be displayed.
When it gets to update the data (if allowed), XYZ or LL Decimal are the most convenient format.

## 7 LoD

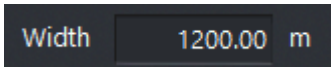

Tiled maps can be layered with Level of Details (LoD) that will hide or display them according to the zoom level (and the effective size on the display for the tile).
You can change the LoD of the selected tile here.

## 8 Corners



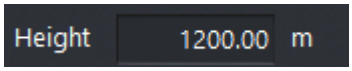

For the selected tile in (4), list here the coordinates of the four corners of the tile, in the coordinate format selected in (6).
The **first** text field (on the **left**) is the X or the Latitude, the **second** (on the **right**) is the Y or the Longitude.
According to the type of raster loaded, corners can be edited or read-only.
Do not forget to update with (5) if any change.

## 9 Width



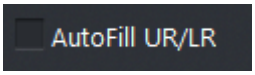

**Width** of the selected tile. Read-only.

## 10 Height



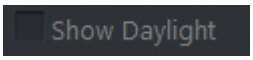

**Height** of the selected tile, Read-only.

## 11 AutoFill UR/LR



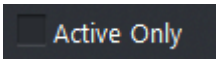

When checked, setting the lower-left and upper-right corners are enough to define a perfect rectangle area for the tile as the two other corners are automatically computed.
If unchecked, all corners must be set.

## 12 Show Daylight



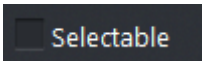

Not available yet.

## 13 Active Only



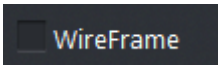

If checked, will rebuild the list to keep only the tiles displayed on the map, either due to the zoom level or to the clipping.
Will filter out invisible tiles from the list.

## 14 Selectable



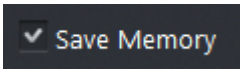

If checked, any tile on the map can be selected with the mouse.

## 15 WireFrames



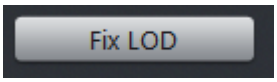

When checked, tiles will not be textured, only outlined.

## 16 Save Memory



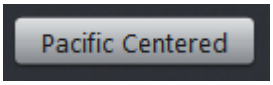

When checked, tiles textures that are not displayed (hidden, clipped out, etc.) are removed from the memory. This option is saving the memory but can slow down slightly the display at zoom and panning as the textures will be loaded from the disk for all new fresh tiles.

## 17 Fix LOD



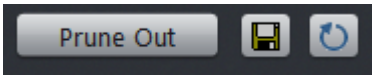

Use this button when the map seems screwed up, mixing big tiles with small one. This can happen when the building of a layered tiles map has not followed the rules.
Save the map file (18) after the fix.

## 18 Pacific Centered



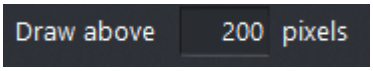

Use this button when the terrain is on the pacific side, around the counter-meridian.

## 19 Optimize



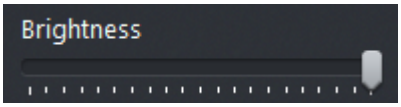

Will remove all tiles that fall outside the terrain area.
Use the button to overwrite the database file (1) after optimization.
Use the button to revert to the database file (1). Same as Cancel.

## 20 Clipping



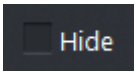

Set here the maximum size of a tile on the display before being replaced with higher resolution tiles (if available).
This is only for multi-layers tiled maps.

## 21 Brightness



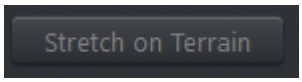

Set the overall brightness of the raster map or tiles. Left is dark, right is full bright.

## 22 Hide




If checked, the raster layer will not be displayed on the terrain map.

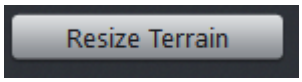
## 23 Stretch on Terrain




Use this button to resize the raster map (outer corners) to the actual terrain size.
This is useful to apply a precisely cut map whose corner coordinates are known, on a terrain.

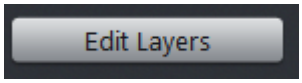

*Disabled with tiled maps.*

## 24 Resize Terrain




Use this button to resize the terrain to the actual size of the raster map (if single) or by the outer corners of a tiled map.

## 25 Edit Layers




Popup the raster map editor window.

# Web Maps

vsTASKER can display map tiles from different providers once the service is activated and Internet connection available
It is suggested to open an account in each providers and register to get a key, otherwise, limitations in the number of tiles one can download per year will apply.

Also, vsTASKER keeps the tiles in a cache directory for offline use and bandwidth reduction. User can empty this cache at any time for space or copyright reasons. During normal use of Web Maps, tiles are first searched into the cache before being requested to the server.

## • How to Use

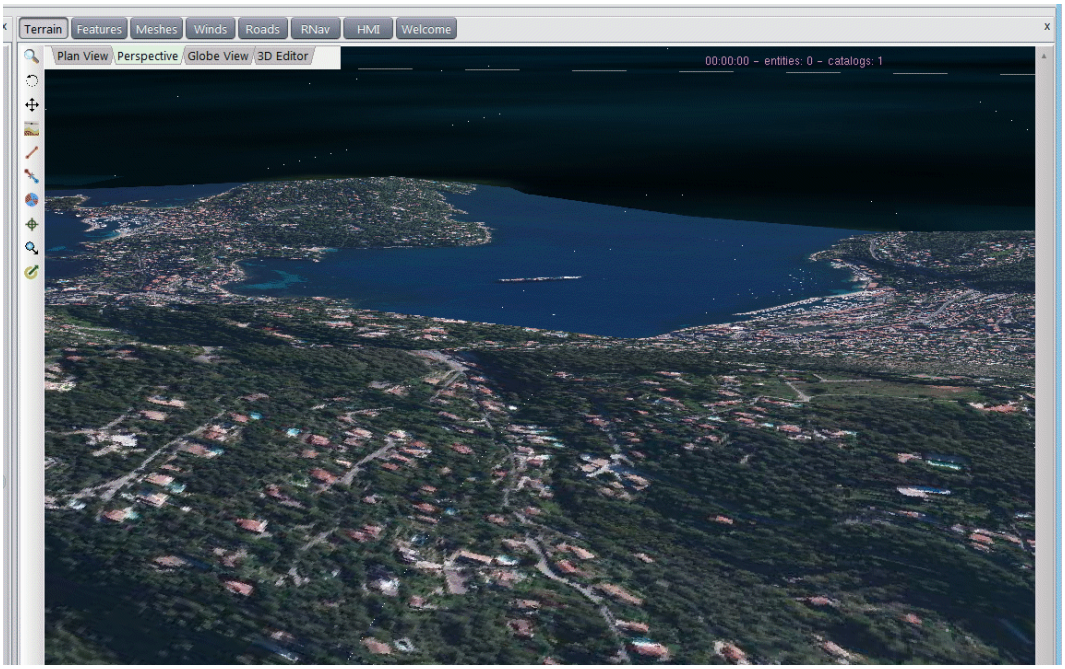
First, set the terrain origin and dimension to the area you want your simulation to take place, then activate the service. Tiles will automatically be downloaded at various resolution according to your zoom level.


*When setting a new scenario somewhere on the world map, it is advisable to use the City Finder button to get the proper Lat/Lon coordinates of the city and have the scenario origin properly setup. Then just specify the area around the city center and you will be set.*

You also can use the world coastline (or select the template WebWorld) and start zooming at the position you want to setup your simulation.
Use the mouse wheel to zoom in and out. Tiles will automatically increase resolution and download textures from the web.

Although plan mode is commonly used, Perspective mode can provide a 3D view with elevations (if available).

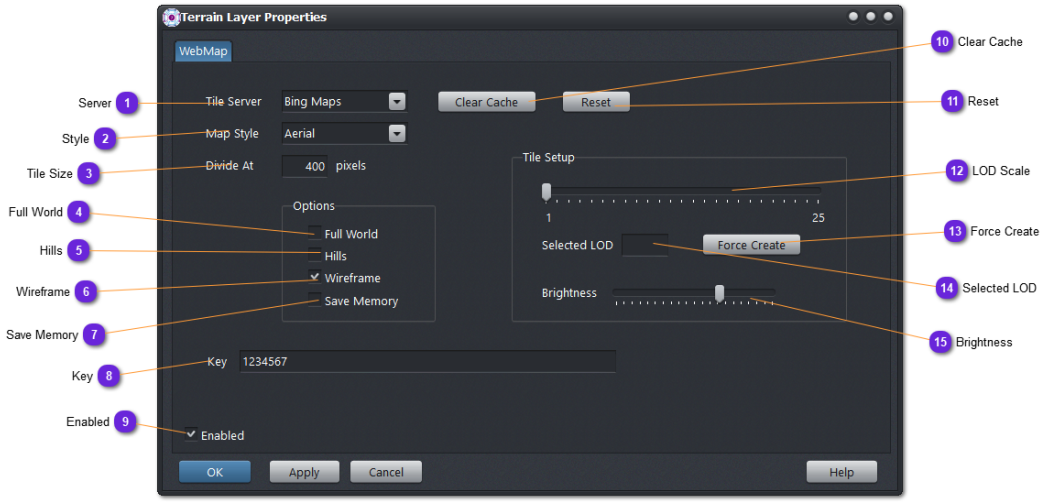
## Web Maps



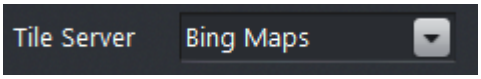


*Once a tile has been downloaded, it remains available locally in the cache and will not be downloaded again unless the cache is freed.*

# Properties



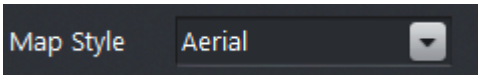

## 1 Server




Select the tile server to be used.

- Bing (Microsoft) is only supported for the moment.

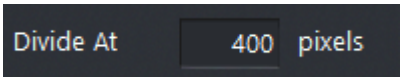
## 2 Style




Select here the kind of map to be used. Not all are available from the selected server.

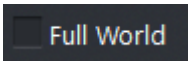
- Aerial: satellite picture
- Aerial + Labels: satellite pictures with some roads and symbols (borders)
- Roads: road maps
- Roads + hills: roads maps with terrain elevation shaded

## 3 Tile Size



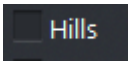

Specify here in pixels the maximum diagonal size of a tile (when zoomed) before it is divided into 4 pieces of higher level of details (LOD).

## 4 Full World



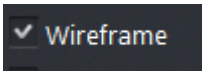

Select this option to force the WebMap server to set the root at level 1, whatever the size of the terrain.

## 5 Hills




When this option is checked, tiles corners will retrieve elevations from the database terrain (if provided in the layers) and not from the tile provider. Perspective view will be able to show elevation in 3D. All tiles must have the same LOD to avoid artifacts between tiles of different sizes. See Force Create to avoid that.

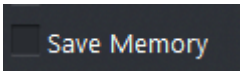
## 6 Wireframe




When checked, displayed tiles are drawn with a magenta border. The tile LOD is mentioned on the lower left corner.

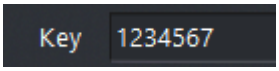

*Does not work in perspective view.*

## 7 Save Memory



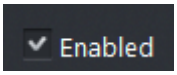

When checked, unloaded tiles (during zoom out or when the tile gets out of view) are freed from the memory and the texture is released.
This saves memory but require reloading the texture each time the associated tile must be created and displayed.

## 8 Key




Most tile servers need a registration to track the tile usage. Enter here your own key (provided by the tile server for your account - create one if needed) to avoid the limitation in the number of tiles allowed per IP.

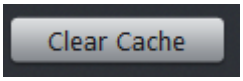
## 9 Enabled




Enable the Web Maps here. The terrain setting will be used to retrieve the appropriate tiles.


*Before enabling the Web Maps, it is better to setup a terrain origin and dimensions properly, using Lat Lon values.*

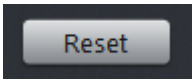
## 10 Clear Cache




This button will remove all the downloaded tiles from the cache.
Only the cache of the selected server and style is cleared.
You need to change servers and styles and repeat the process to clear all tile files.

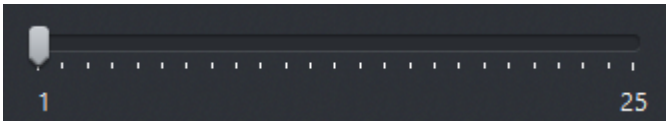

*Cache can be found in /Data/Map/Servers. In Bing directory (for i.e), you will find one directory for each style (a,r,h...).*
*In each of these style directories, you will see the LOD sub directories (numbers).*
*You can manually delete all LOD directories (1,2,3,4..).*
***Do not remove the _floor.png file !***

## 11 Reset



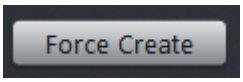

Force the rebuild of the center node, according to the new terrain setting or if Full World option has been activated or removed.

## 12 LOD Scale




When Web Maps is enabled, according to the terrain dimensions, the LOD of the root tile is given here.
You can use the slider to select a bigger LOD (selecting a lower LOD will have no effect).

## 13 Force Create



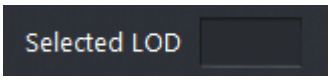

This button will increase the accuracy of all tiles down to the selected LOD. To reduce the number of tiles to be created (raw estimate is 4^n, with n = selected LOD - root LOD), only the tiles falling into the terrain area will be processed.

If you want to have a good and uniform resolution of an area for perspective view, you can first manually setup the terrain dimension for the desired area, uncheck Save Memory (to avoid releasing the hidden tiles), then Force Create the tiling at the specified LOD.

Select Hills options and if you have loaded an elevation layer for the same area, tiles will be clamped to terrain surface. In perspective mode, you will see the highest LOD tiles (artifacts will be visible at the border with tiles of lower LOD).


*If you think the number of tiles is too high, you will need to kill vsTASKER to abort the process. Do not let it run as it will empty your quota of tiles quickly.*
*For a world area, LOD 1, trying to Force Create up to LOD 25 will require more than 1125899906842624 files. You do not want that (and your computer neither).*

## 14 Selected LOD



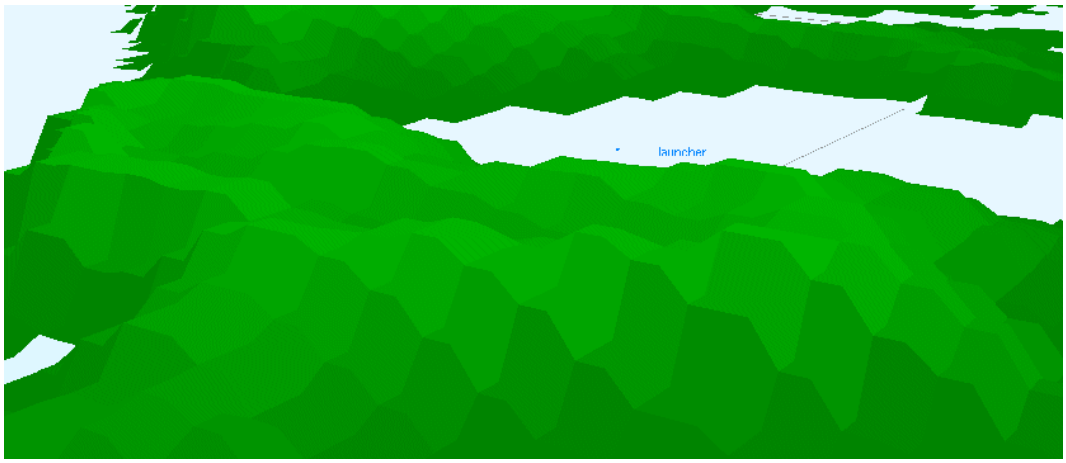

Shows the actual LOD selected by the slider above.
Can also be set manually if the value must be higher than 25.

## 15 Brightness

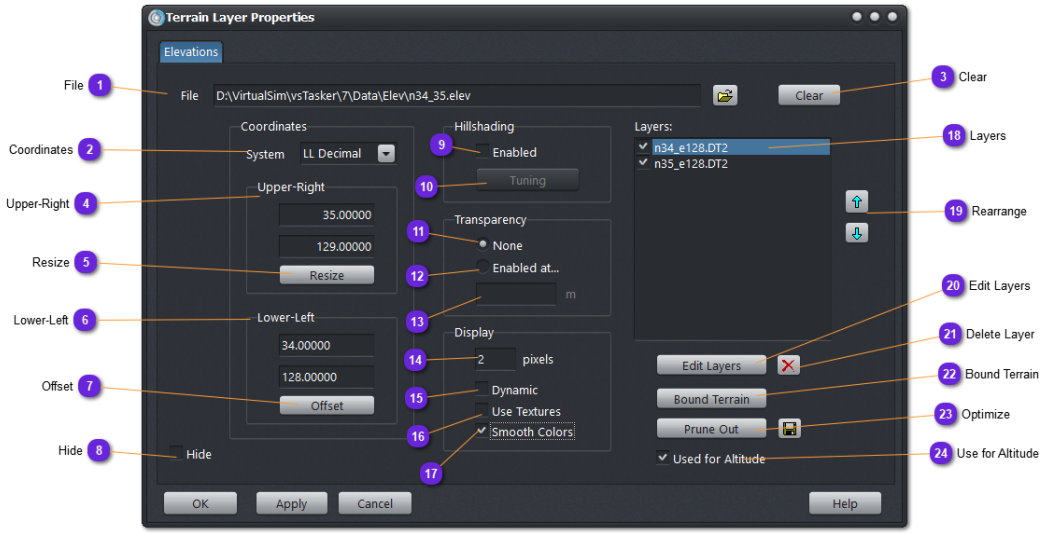

Set the overall brightness of the raster map or tiles. Left is dark, right is full bright.

# Elevations

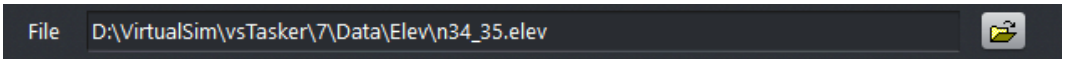
Elevation databases are based on **DTED**, **DEM, HGT or ASCII Grid** source data.
They are organized as tiles of elevation plots organized in rectangular grids.
Each corner of the tile is geo-localized and plots coordinates are linearly interpolated between corners.



# Properties




## 1 File

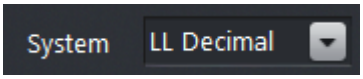



Name of the elevation **file**.
Supported formats:

- elev: grid of elevation plots, proprietary format from TerrainBuilder converter tool.
- e00:
- dt0:
- dt1:
- dt2:
- dt3: raw formats.

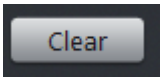

*Use the TerrainBuilder tool to import other elevation data types (DEM, HGT, ASC...)*

## 2 Coordinates



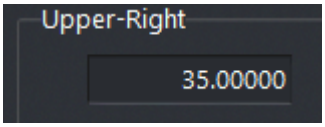

Select here the **coordinate** system preference for displaying the elevation database area corners.

## 3 Clear




Free the memory with the loaded elevation database.

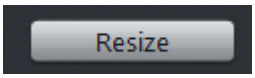
## 4 Upper-Right




Coordinates Upper-Right corner of the elevation database or the selected layer (15).

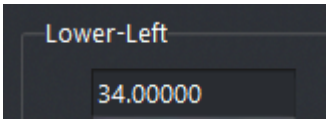

*Read-only.*

## 5 Resize



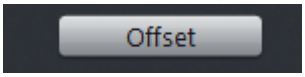

Use this button in order to resize the selected layer with new upper-right coordinates (the lower-left corner remains unchanged).

## 6 Lower-Left



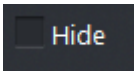

Coordinates Lower-Left corner of the elevation database or the selected layer (15).
These values are user modifiable.

## 7 Offset



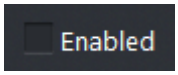

Use this button to relocate the selected layer by moving its Lower-Left corner to the new coordinates.
If no layer is selected, all will be relocated according to the difference between the elevation database corner and the new one.
The operation does not change the size of the terrain.

## 8 Hide



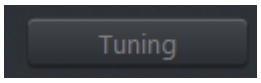

If checked, the elevation database layer will not be displayed on the terrain map.

## 9 Hill Shading



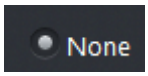

When checked, the hill shading algorithm will apply on the elevation database. See the tuning (8)

## 10 Tuning



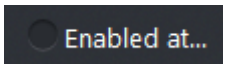

See here.

## 11 No Transparency



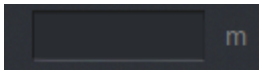

If checked, all plots are drawn, regardless of their altitude.

## 12 Transparency



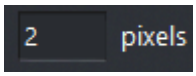

If checked, any plot altitude equal to the value set in the below field (11) will not be drawn.

## 13 Transparency Altitude



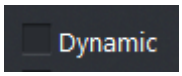

Set here the exact value in meters for plots to be transparent (not drawn).

## 14 Accuracy



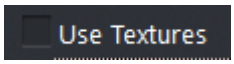

Set here the **minimum size** of a visible plot. The lower the value, the more accurate the drawing and the more CPU is required (drawing might be jumpy for big elevation database).
This value does not influence the altitude query. It is only for display.

## 15 Dynamic




If checked, the pixel size will be automatically computed according to the size of the database and the zoom level. The higher the zoom, the higher the accuracy for the display.

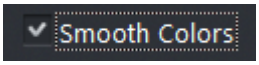
## 16 Use Textures




If checked, the elevation plots for each tile are not drawn. A texture file is used instead. Use this with big elevation database to increase drastically the drawing process.

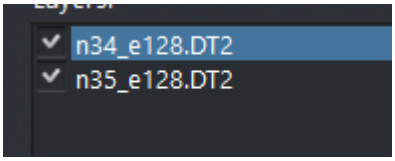

*First time the option is used, vsTASKER will automatically build the textures for all the elevation tiles (15). Might take some time. Done once.*

## 17 Smooth Colors




When this flag is raised, the plot colors are interpolated according to altitudes of the bounding layers.
ie: if layer 1 for altitudes [0..10[ is green and layer 2 for altitudes [10..20[ is yellow, plots for altitudes [5..10[ will have colors interpolated between green and yellow (layer 1 & 2) while plots for altitudes [0..5[ will have colors interpolated between layer 0 & 1 defined ones.

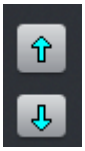
## 18 Layers




List here all the plots layers making the elevation database.
When a layer is **selected**, the corner coordinates are updated (4,5)
When a layer is **unchecked**, it will not be displayed on the terrain map.

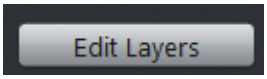

*To deselect, just click outside the list.*

## 19 Rearrange



Move up or down the list the selected layer.
This is useful to reorganize the list with the higher accuracy layers at the top and the lowest and widest ones at the bottom.

## 20 Edit Layers

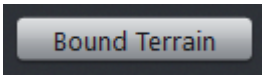



Call the elevation editor.
See here.

## 21 Delete Layer

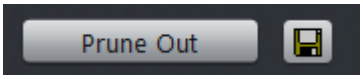
Delete the selected layer of the list above.

## 22 Bound Terrain



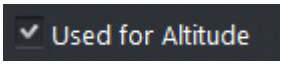

Resize the terrain dimensions to match the elevation database ones.

## 23 Optimize




Remove all tiles (if any) that fall entirely outside the terrain area.
Use the button to overwrite the elevation file.

## 24 Use for Altitude

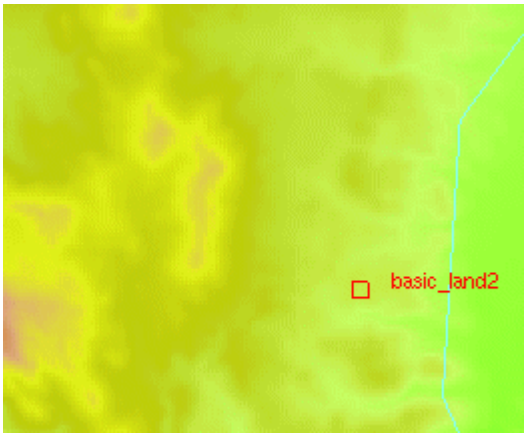


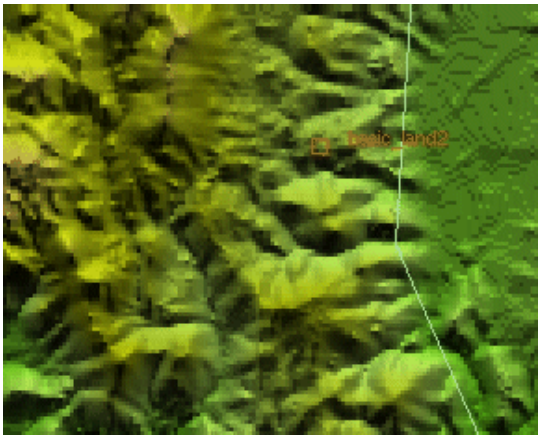

When this option is set (default), the Terrain will use this layer to get the altitude at a given position: function getAlt(WCoord);
If disabled, the altitude will be taken from another layer (if any) or return the global Z/Alt value set in Terrain Settings window.

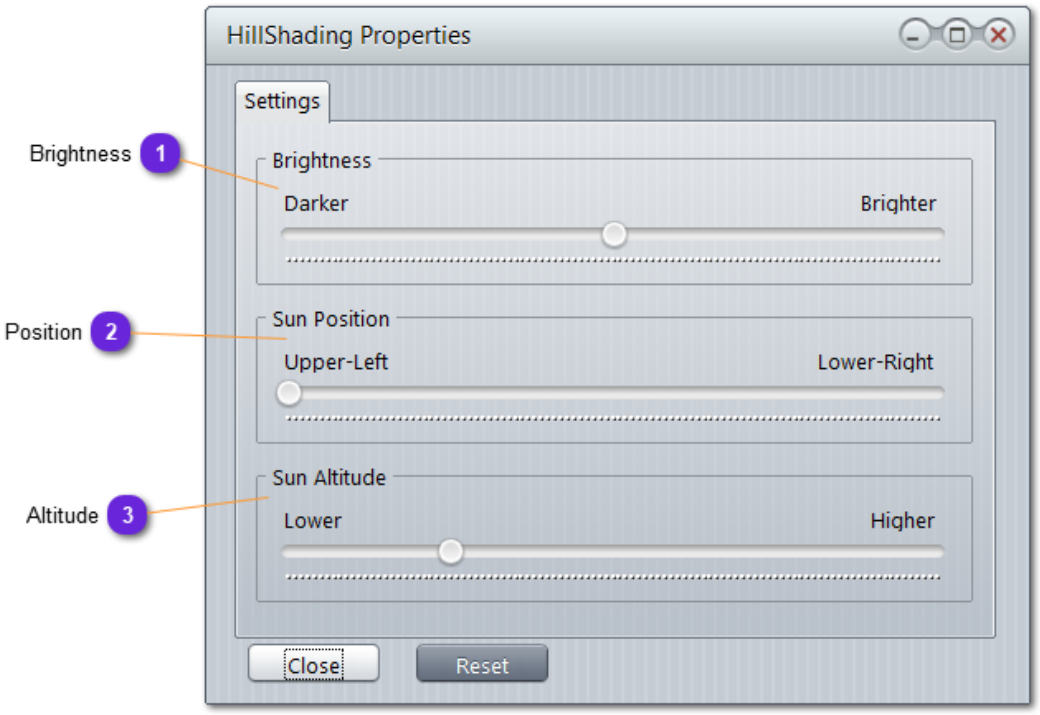
# Hillshading

Hill shading algorithm paint apply a brightness to each elevation plot according to general ambient light, sun position regarding the two opposite corners and the altitude above the terrain.
The formula compute the angle between the normal of the elevation area and the vector pointing to the sun. The smaller the angle, the brighter the plot.

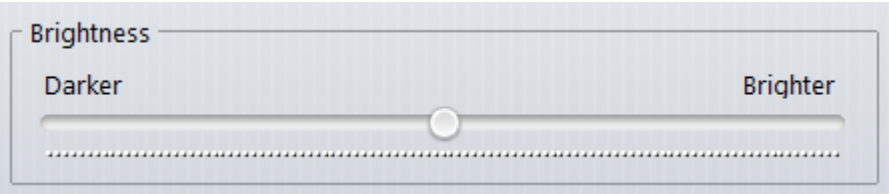

*Occultation is not taken into account. There is no shadow, even if the sun is set as low altitude.*



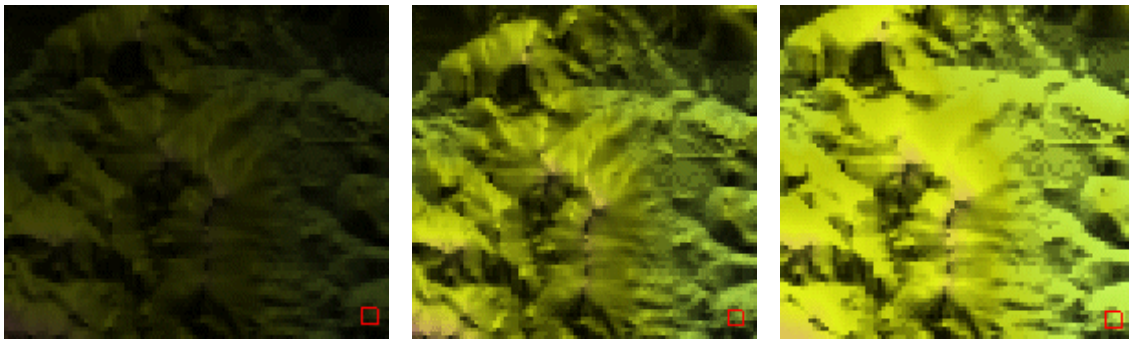





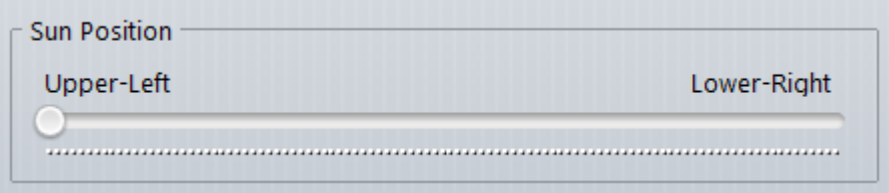
## 1 Brightness




Set the ambient light **alpha** of the sun.

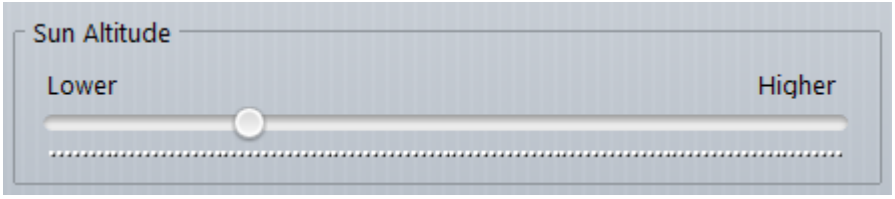


## 2 Position




Set the **position** of the sun from the top left corner of the terrain down to the bottom right corner.

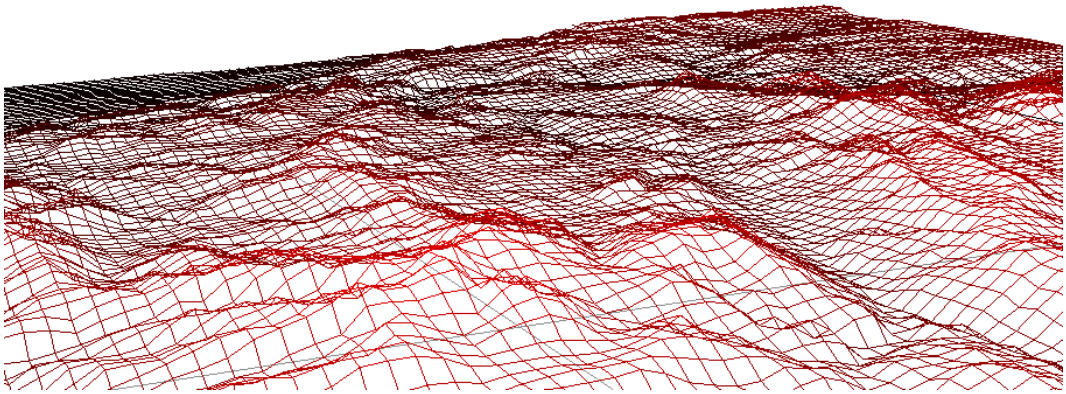
## 3 Altitude




Set the altitude of the sun from 100 m to 50,000 m

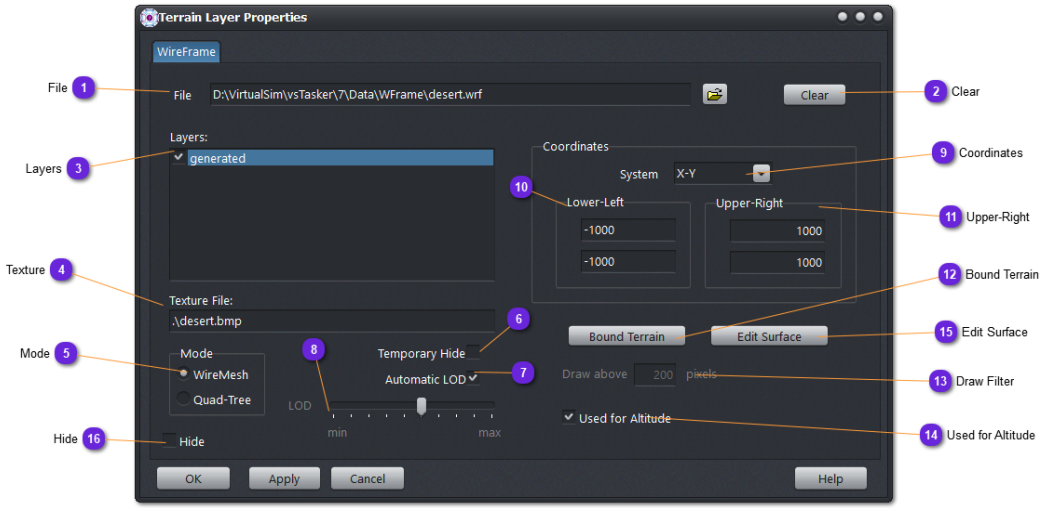
# Wire Frames

Another way to represent a terrain with elevations is to use wireframes.
A wireframe are like a mesh made of polygons. A texture can be applied on the full wireframe like a skin.
The more polygons, the more accurate will be the result on the display and regarding the altitude request.

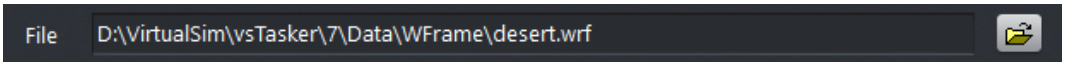



*Wireframes are built using the TerrainBuilder tool.*

# Properties



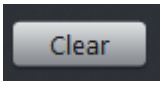

## 1 File




Filename of the wireframe **database**. Supported formats:

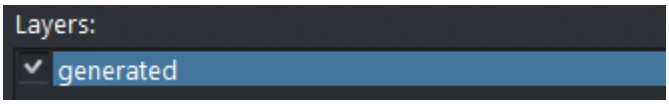
- wrf: proprietary format of TerrainBuilder tool.

## 2 Clear




**Remove** the wireframe from the memory and scenario.

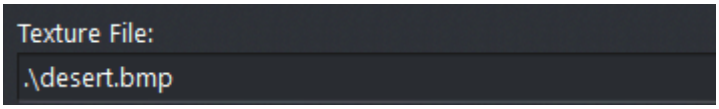
## 3 Layers




List of all **wireframes** (layer) included into the database file (1).
User must select one to view coordinates and the associated texture skin.
When a layer is **selected**, the corner coordinates are updated (10,11)
When a layer is **unchecked**, it will not be displayed on the terrain map.

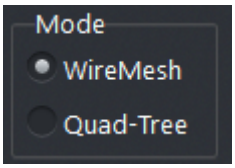

*Read only for now. All modifications are lost.*

## 4 Texture




Texture **file** associated with the selected wireframe.

## 5 Mode

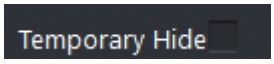



Show here the **representation** mode of the selected layer:

- WireMesh: single grid with a fixed number of meshes.
- Quad-Tree: multiple grids where all meshes can be divided into 4 sub meshes, recursively.

*A layer need to be selected into the list (1). Read only.*

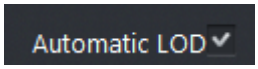
## 6 Temporary Hide




If clicked, the **texture** mesh will be **hidden**.

*A layer need to be selected into the list (1).*

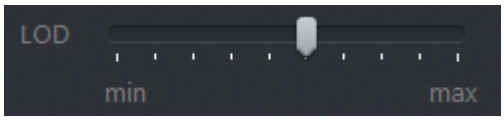
## 7 Automatic LoD




If clicked, the number of meshed displayed will be **computed** according to the zoom level. This is only for speeding up the drawing process. Internally, the number of meshes is maximum and depends on the wire mesh definition.

*A layer need to be selected into the list (1).*

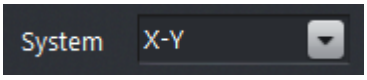
## 8 LOD Scale




When Automatic LOD is not checked, use this slider to **manually** set the grid quality that will be kept whatever the zoom level.

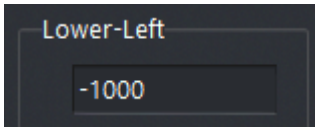
*A layer need to be selected into the list (1).*

## 9 Coordinates



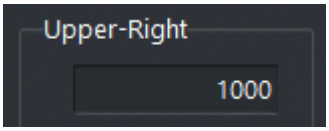

Select the coordinate units for the displayed values of the corners (10 & 11).

## 10 Lower-Left



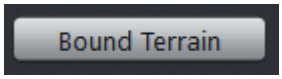

Coordinates of the **bottom left** corner of the wireframe database or just the selected layer in (1).

## 11 Upper-Right



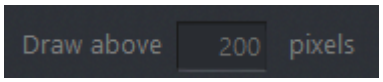

Coordinates of the **top right** corner of the wireframe database or just the selected layer in (1).

## 12 Bound Terrain



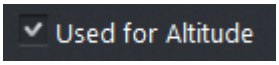

Recalculate the terrain settings to match the wireframe (database) area (utter most corners).

## 13 Draw Filter



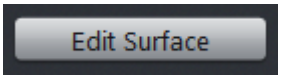

In case of quad-tree wireframe, the **mesh level** will be chosen when the size of the inner mesh will be greater than the value set in this field.
The bigger the value, the more zoom will be requested before switching to higher definition (if any defined in the database).

## 14 Used for Altitude



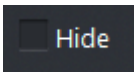

When this option is set (default), the Terrain will use this layer to get the altitude at a given position: function getAlt(WCoord);
If disabled, the altitude will be taken from another layer (if any) or return the global Z/Alt value set in Terrain Settings window.

## 15 Edit Surface




Call the surface **editor**.
See here.

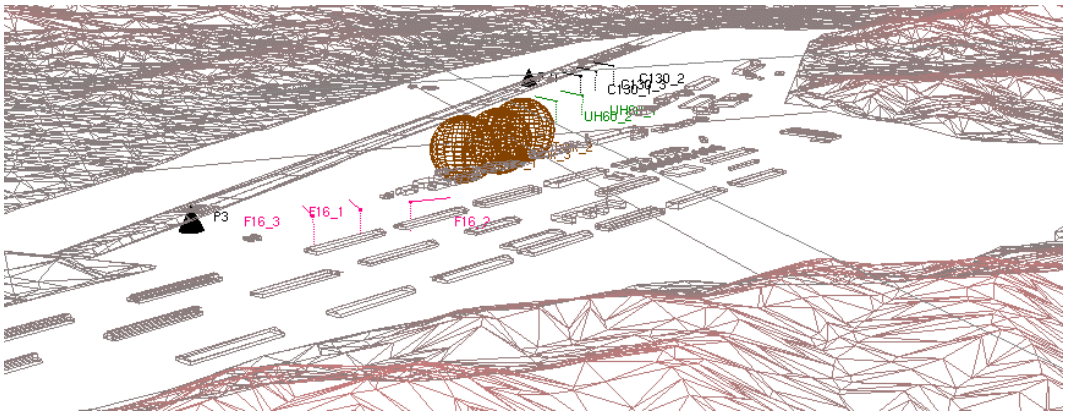
## 16 Hide




If checked, the wireframe will not be displayed on the terrain map.

# Open Flight

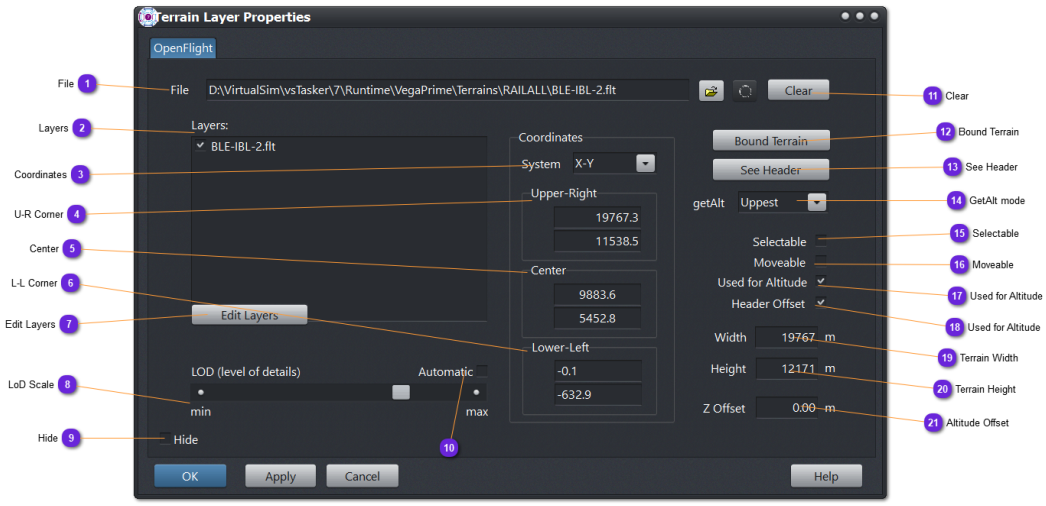
vsTASKER can import Open-Flight database format and once converted by the TerrainBuilder tool, can be displayed on the terrain map **without texture**.
The imported converted terrain will be used as a collection of triangle meshes.
It is still useful for setting up accurately a scenario with entities.



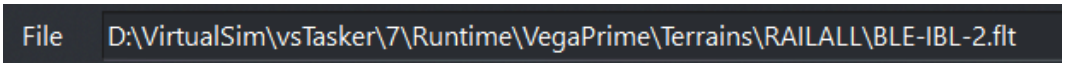


*3D Editor can be used with Open-Flight terrain database.*

# Properties



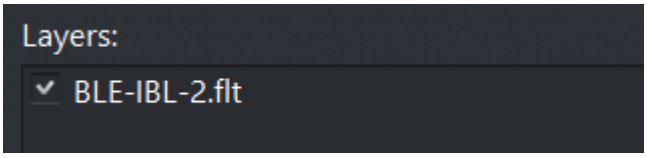

## 1 File




Select here the converted **Open-Flight database**. Supported formats:

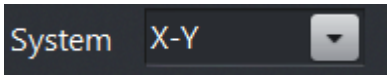
- opf: proprietary format of TerrainBuilder tool.

## 2 Layers



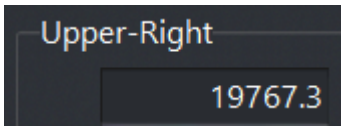

List of all the layers and objects included into the imported database.
When a layer is **selected**, the corner coordinates are updated (5,6)
When a layer is **unchecked**, it will not be displayed on the terrain map.

## 3 Coordinates



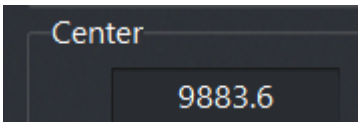

Select the coordinate units for the displayed values of the corners.

## 4 Upper-Right



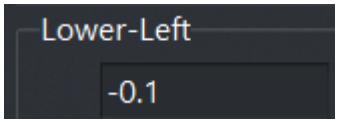

Coordinates of the **top right** corner of the Open-Flight database or the selected layer (1).

## 5 Center



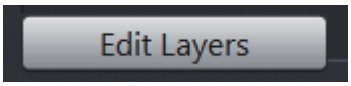

Coordinates of the **center** of the Open-Flight database or the selected layer (1).

## 6 Lower-Left



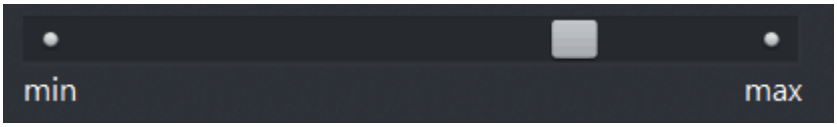

Coordinates of the **bottom left** corner of the Open-Flight database or the selected layer (1).

## 7 Edit Layers




Popup the local editor.
See here.

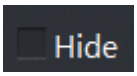
## 8 LoD Scale




When Automatic LOD is not checked, use this slider to **manually** set the drawing quality that will be kept whatever the zoom level.

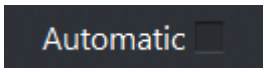
*A layer need to be selected into the list (1).*

## 9 Hide




If checked, the Open-Flight database will not be displayed on the terrain map.

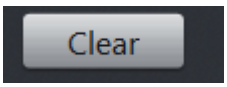
## 10 Automatic LoD




If clicked, the number of triangles displayed will be **computed** according to the zoom level. This is only for speeding up the drawing process. Internally, the number of triangles is maximum and depends on the database definition.

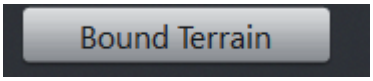
*A layer need to be selected into the list (1).*

## 11 Clear



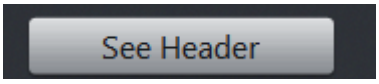

**Remove** the Open-Flight from the memory and scenario.

## 12 Bound Terrain



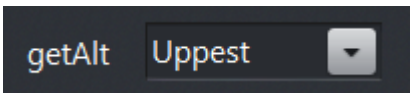

Recalculate the **terrain settings** to match the Open-Flight (database) area (utter most corners).

## 13 See Header




Popup the header information window.

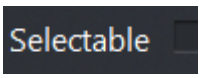
## 14 GetAlt mode




Select how the altitude retriever shall work:

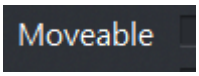
- **Uppest**: select the uppest altitude from all polygons at a given position (default)
- **Lowest**: select the lowest altitude from all polygons at a given position
- **First**: select the altitude of the first polygon found below the given position (old)

## 15 Selectable



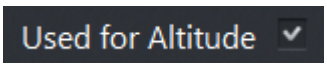

If checked, use the mouse to **select** any **layer** of the Open-Flight database. Useful to identify some shapes for edition or removal.

## 16 Moveable



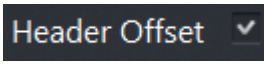

If checked, the full database (including all layers) can be **offset** by values. Use the layer editor window (offset) for that purpose.

## 17 Used for Altitude



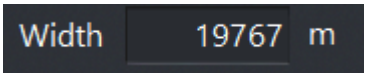

When checked, the OpenFlight layers will be used to retrieve the altitude at a given coordinates.

## 18 Used for Altitude



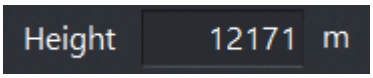

When this option is set (default), the Terrain will use this layer to get the altitude at a given position: `function getAlt(WCoord);`
If disabled, the altitude will be taken from another layer (if any) or return the global Z/Alt value set in Terrain Settings window.

## 19 Terrain Width




Total **width** (east west distance in meters) of the database of the selected layer in (1).

## 20 Terrain Height




Total **height** (south north distance in meters) of the database of the selected layer in (1).

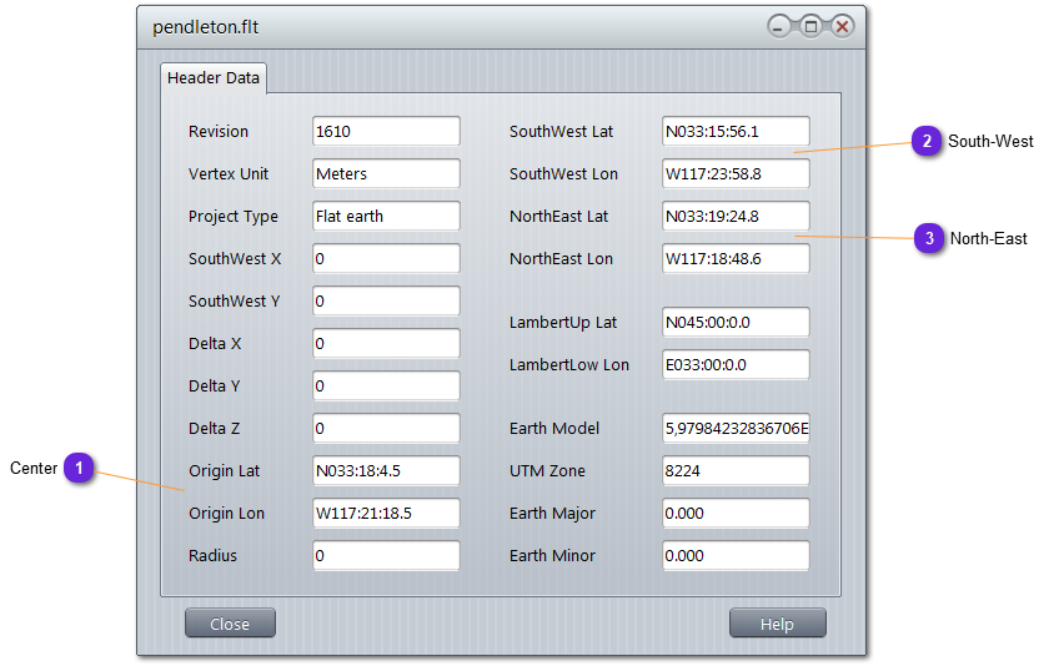
## 21 Altitude Offset


Put here a global value for offsetting the returned altitude when this layer is used to return an altitude at a point (see: **Used for altitude** option above)

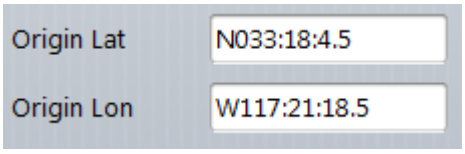
# Header Data

List here all the extracted data from the original Open-Flight terrain database.

*Read only panel.*

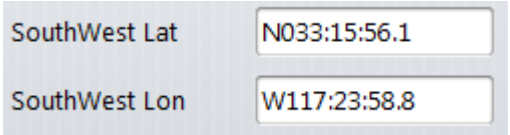



## 1 Center



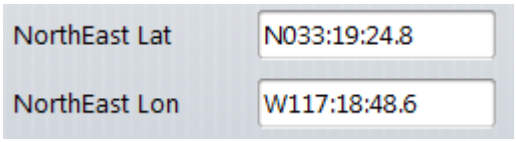

**Center** coordinates of the Open-Flight database, in literal Lat-Lon.

## 2 South-West




**Bottom-Left** corner of the open-flight database, in literal Lat-Lon.

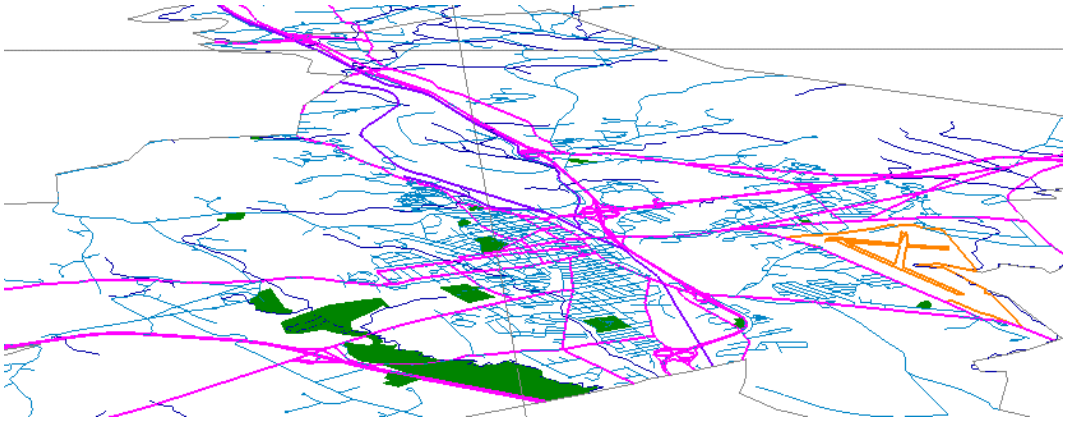
## 3 North-East



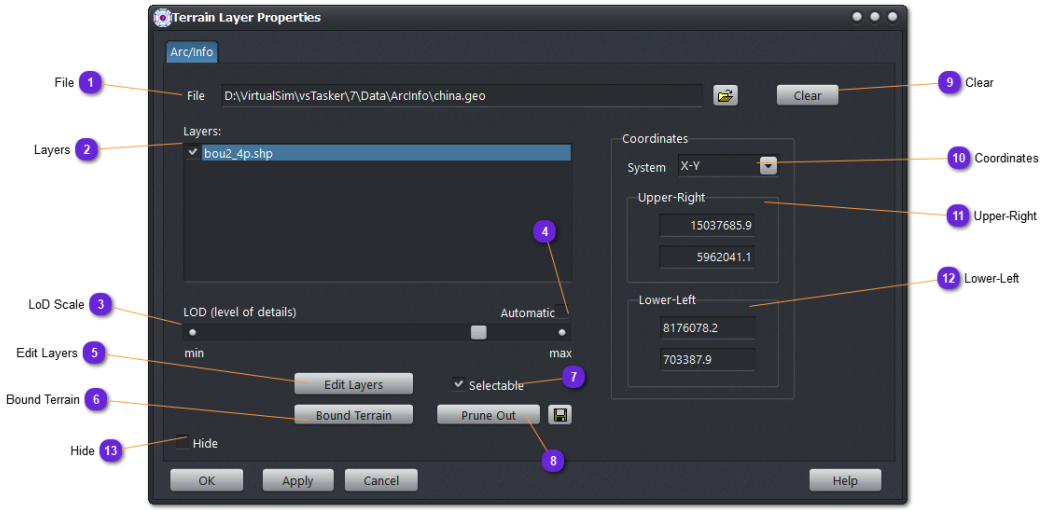

**Upper-Right** corner of the Open-Flight database, in literal Lat-Lon.

# Arc Info

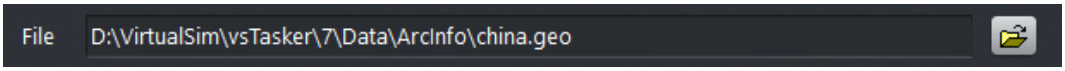
vsTASKER can import ESRI **Arc/Info** database format and once converted by the TerrainBuilder tool, can be displayed on the terrain map with lines, colors and filled polygons.
Any content (shape) of the database can be queried from the code and using the mouse.



# Properties



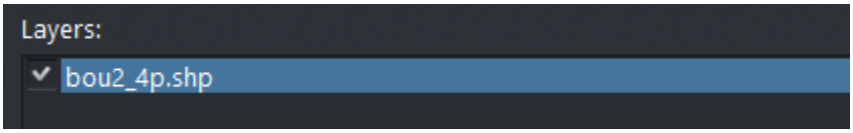

## 1 File




Select here the converted **Arc-Info database**. Supported formats:

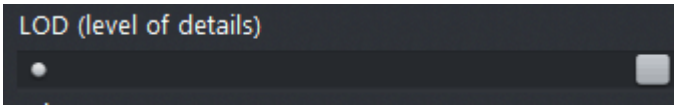
- geo: proprietary format of TerrainBuilder tool.

## 2 Layers




List of all the layers included into the imported database.
When a layer is **selected**, the corner coordinates are updated (11,12)
When a layer is **unchecked**, it will not be displayed on the terrain map.

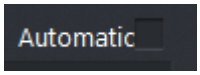
## 3 LoD Scale




When Automatic LoD is not checked, use this slider to **manually** set the drawing quality that will be kept whatever the zoom level.

*A layer need to be selected into the list (1).*

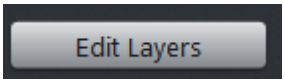
## 4 Automatic LoD




If clicked, the number of segments (lines, polygons) displayed will be **computed** according to the zoom level. This is only for speeding up the drawing process. Internally, the number of segments is maximum and depends on the database definition.

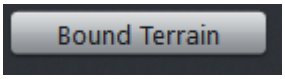
*A layer need to be selected into the list (1).*

## 5 Edit Layers



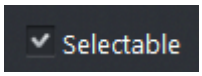

Popup the local editor.
See here.

## 6 Bound Terrain

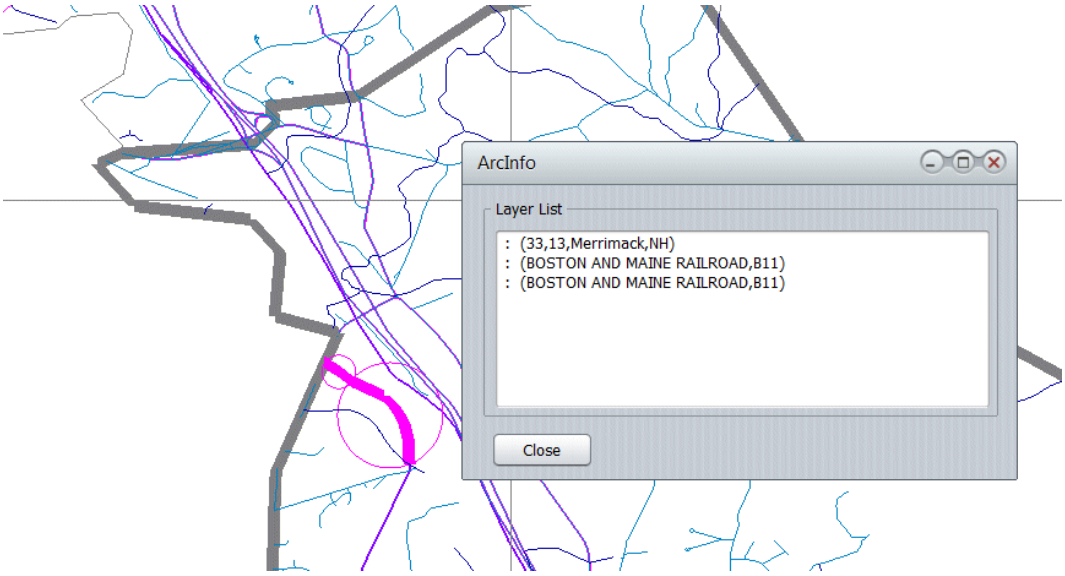



Recalculate the **terrain settings** to match the Open-Flight (database) area (utter most corners).

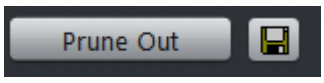
## 7 Selectable




If checked, use the mouse to **select** any **layer** of the Arc-Info database. Useful to identify some shapes for edition or removal.
Selected layers are displayed in bold.

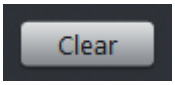



## 8 Optimize



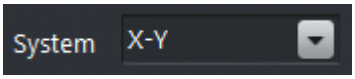

Removes all shapes (if any) that fall entirely outside the terrain area.
Use the button to overwrite the Arc-Info file.

## 9 Clear



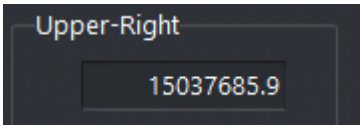

**Remove** the Arc-Info from the memory and scenario.

## 10 Coordinates



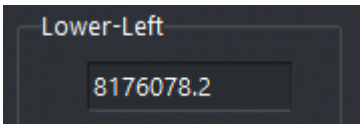

Select the **coordinate units** for the displayed values of the corners.

## 11 Upper-Right



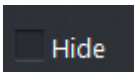

Coordinates of the **top right** corner of the Arc-Info database or only the selected layer (1).

## 12 Lower-Left




Coordinates of the **bottom left** corner of the Arc-Info database or only the selected layer (1).

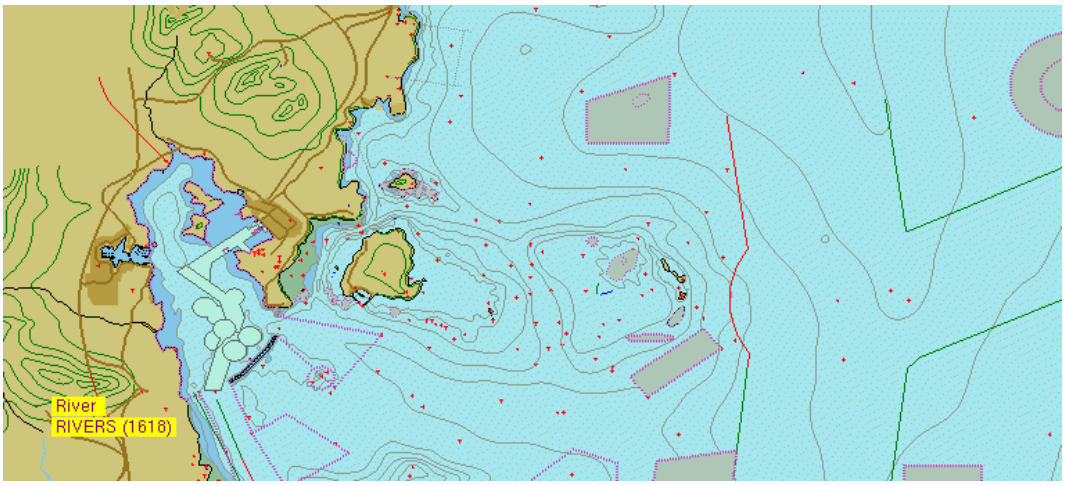
## 13 Hide



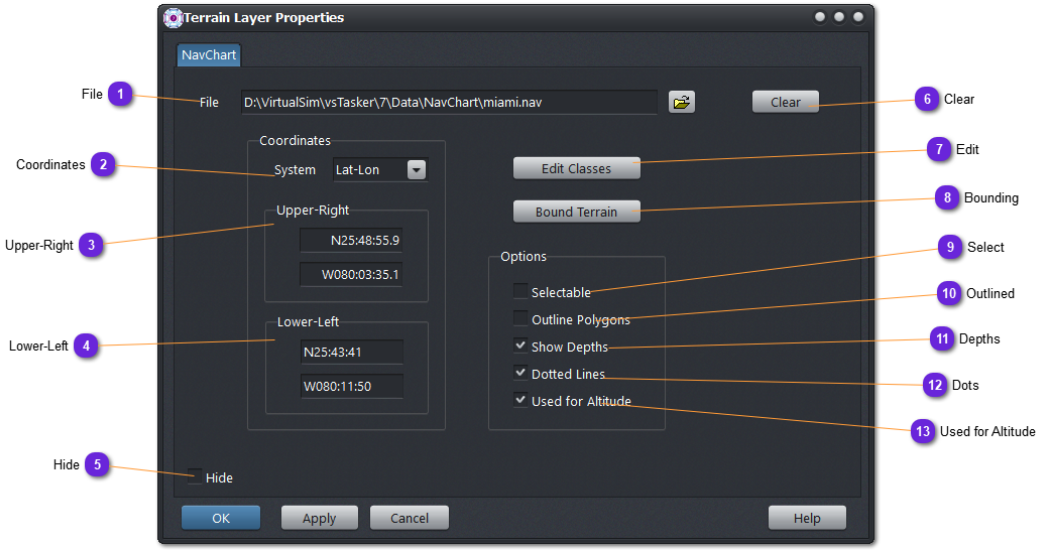

If checked, the Arc-Info database will not be displayed on the terrain map.

# Nav Charts

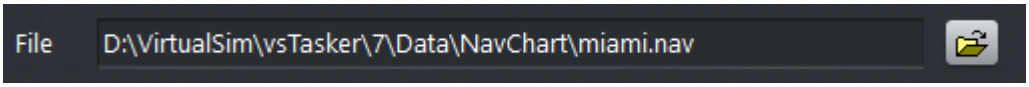
Navigation charts are converted from **S57 maritime** chart format using the TerrainBuilder tool.
They are used for displaying maritime area for navigation.



# Properties



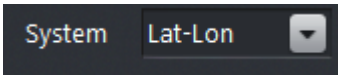

## 1 File




Select here the converted **navchart database**. Supported formats:

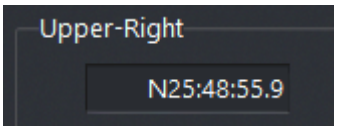
- nav: proprietary format of TerrainBuilder tool.

## 2 Coordinates



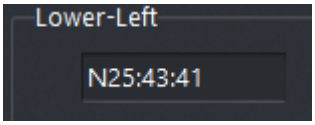

Select the **coordinate units** for the displayed values of the corners.

## 3 Upper-Right



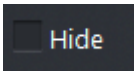

Coordinates of the **top right** corner of the navchart database or only the selected layer (1).

## 4 Lower-Left



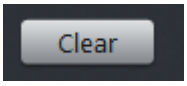

Coordinates of the **bottom left** corner of the navchart database or only the selected layer (1).

## 5 Hide



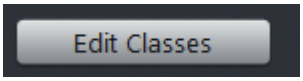

If checked, the navchart database will **not be displayed** on the terrain map.

## 6 Clear



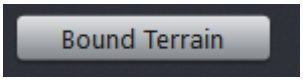

**Remove** the navchart from the memory and scenario.

## 7 Edit



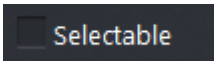

Popup the local editor.
See here.

## 8 Bounding



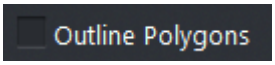

Recalculate the **terrain settings** to match the navchart (database) area (utter most corners).

## 9 Select



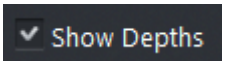

If checked, use the mouse to **select** any **layer** of the navchart database.
Useful to identify some elements for edition or removal.
Selected elements are displayed in bold with their names.

## 10 Outlined



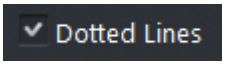

Display contours only for filled areas. Outlining filled regions requires less CPU than filling them. Use that when the map slows too much the refresh rate.

## 11 Depths



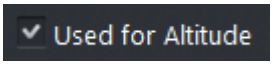

When selected, depths values are shown at a high zoom level. This might be useful but can be disturb the map with too much information.
Unselect this option to prevent depth drawing.

## 12 Dots




Some classes are drawing lines using dots or dashes. It can be nice on the display but CPU intensive on some graphic cards.
Unselect this option to force using plain lines (also thinner) and release the CPU.

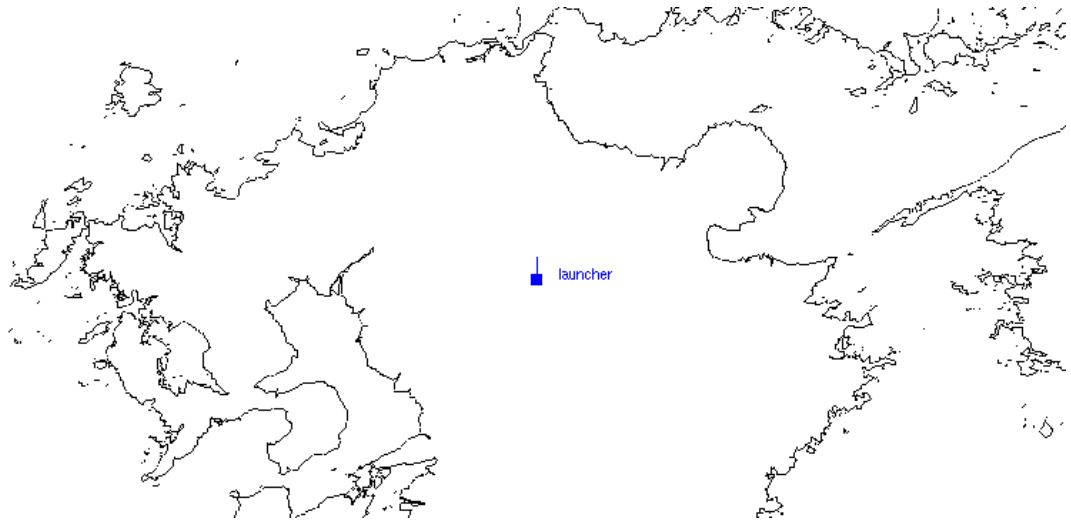
## 13 Used for Altitude



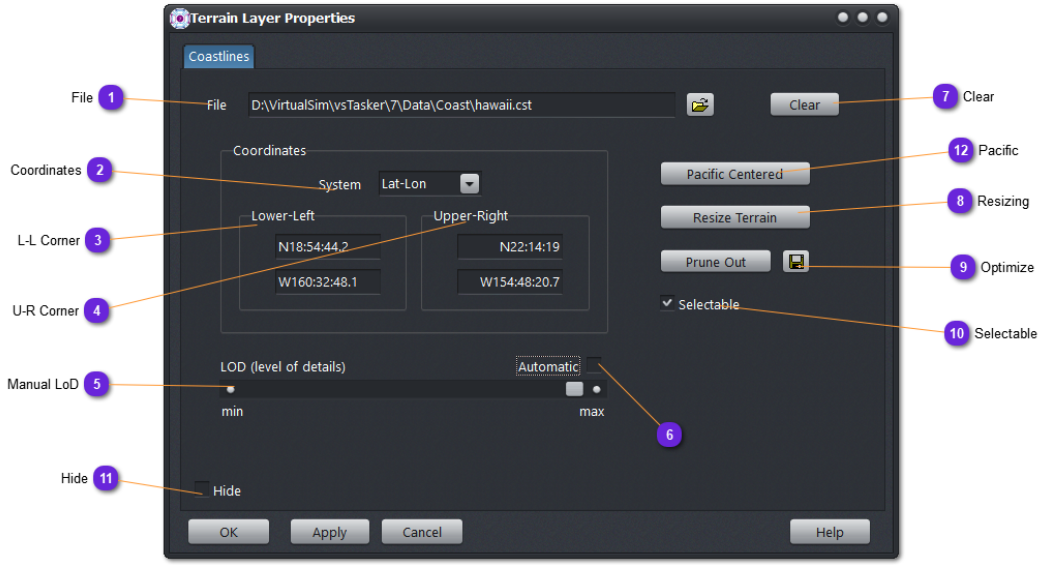

When this option is set (default), the Terrain will use this layer to get the altitude at a given position: function `getAlt(WCoord);`
If disabled, the altitude will be taken from another layer (if any) or return the global Z/Alt value set in Terrain Settings window.

# Coast Lines

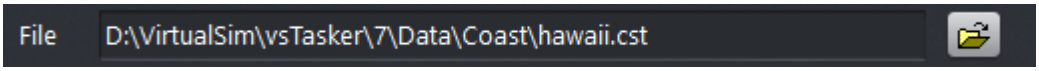
Coastlines are proprietary data coming from conversion using TerrainBuilder tool of shorelines in shapefile format or other ASCII format.
NGDC NOAA resources provides a good online extractor.



# Properties



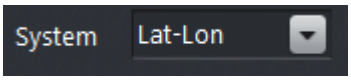

## 1 File




Select here the converted **coastline database**. Supported formats:

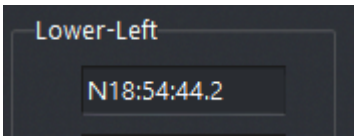
- cst: proprietary format of TerrainBuilder tool.

## 2 Coordinates



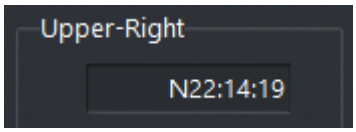

Select the **coordinate units** for the displayed values of the corners.

## 3 L-L Corner



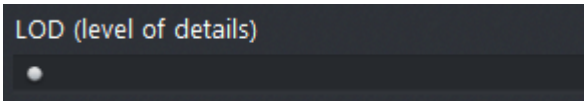

Coordinates of the **bottom left** corner of the coastline database or only the selected layer (1).

## 4 Upper-Right



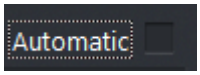

Coordinates of the **top right** corner of the coastline database or only the selected layer (1).

## 5 Manual LoD



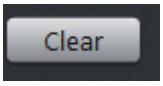

When Automatic LoD is not checked, use this slider to **manually** set the drawing quality that will be kept whatever the zoom level.

## 6 Automatic LoD



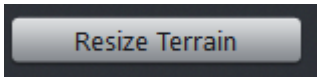

If clicked, the number of segments displayed will be **computed** according to the zoom level. This is only for speeding up the drawing process. Internally, the number of segments is maximum and depends on the database definition.

## 7 Clear



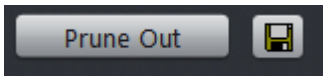

**Remove** the coastline from the memory and scenario.

## 8 Bound Terrain



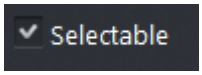

Recalculate the **terrain settings** to match the coastline (database) area (utter most corners).

## 9 Optimize



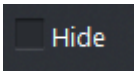

Removes all segments (if any) that fall entirely outside the terrain area.
Use the button to overwrite the coastline file.

## 10 Selectable



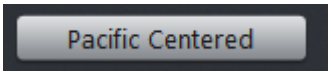

If checked, use the mouse to **select** any segment portion of the coastline database.

## 11 Hide




If checked, the coastline database will **not be displayed** on the terrain map.

## 12 Pacific

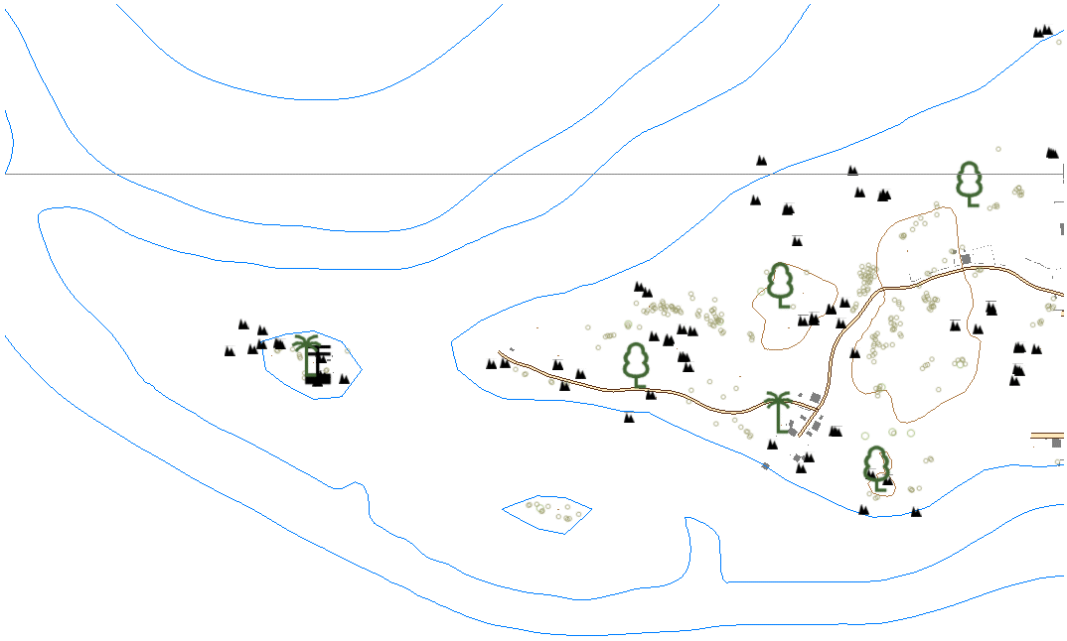



Select this option when the center of the terrain is located close to the International Date Line (between New Zealand and Hawaii) so that the lines are be drawn properly on each side of the IDL.

# Vector Graphics

Vector Graphics are mainly coming for SVG format files (see Scalable Vector Graphics).
vsTASKER does not support all functionalities and has reduced the scope to exported databases from Bohemia Simulation VBS tool.

Importing an SVG file is straightforward.
It can be done directly from vsTASKER GUI or from the TerrainBuilder tool (recommended option).

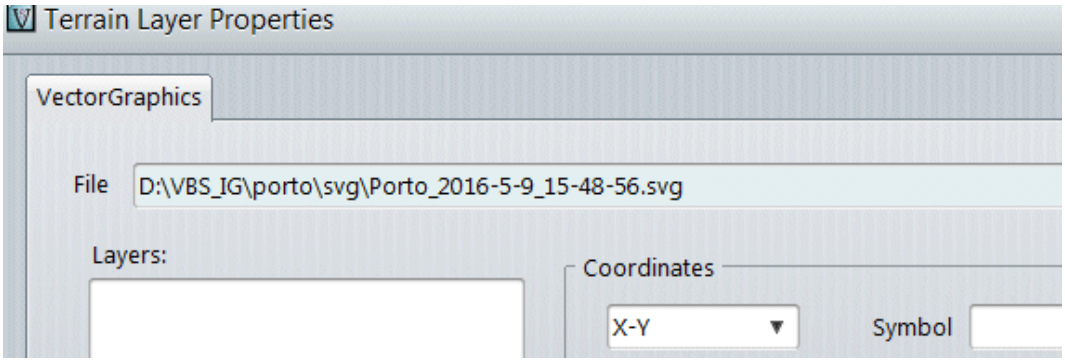


## • How to Use

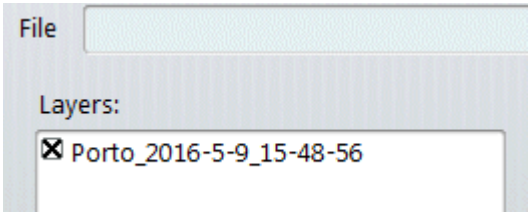
First, open the SVG property window:


Then import your raw SVG file:

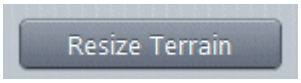
### Vector Graphics



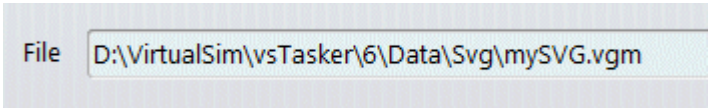

and use the  button to add it as a new layer.
Once the import is successful, you will see the layer in the list (and the file name is removed from the text field.
You can repeat this process with any SVG raw file you may combine.




Now, reposition and resize the terrain to the imported SVG coordinates:

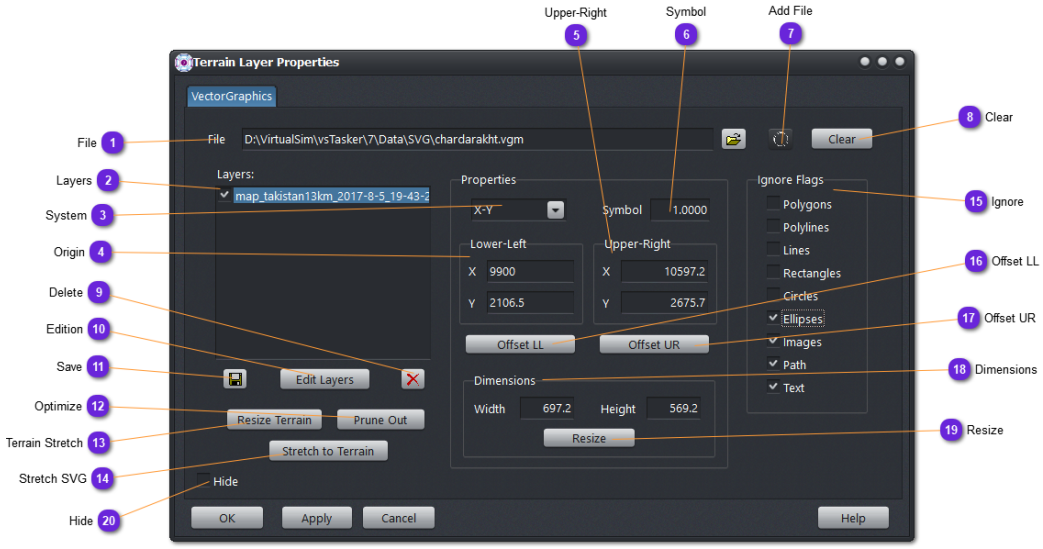



To save the SVG database (all layers) into a vgm proprietary format, open the file selector (or type the full path and name with the .vgm extension) and use the save button: 

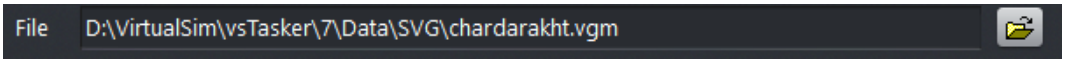



To offset and reposition the SVG database or some layers, refer to the Property Window.

# Properties



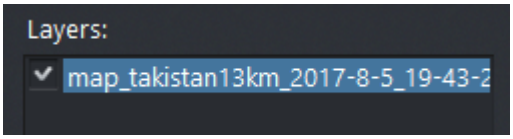

## 1 File




Select here the converted **SVG database**. Supported formats:

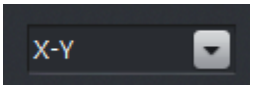
- vgm: proprietary format of TerrainBuilder tool.
- svg: raw format that can be imported into the database.

## 2 Layers




List of all layers belonging to the loaded SVG database or the imported raw file.
To unselect the layers, just click outside the list.

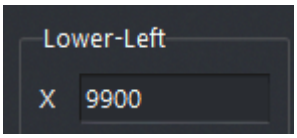
## 3 System




Select in the dropdown list the coordinate formats to be used to display the lower-left and upper-right corners.

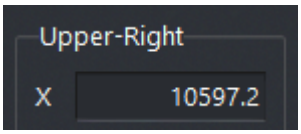
- XY: flat earth, no projection.
- Lat/Lon: latitude/longitude literal coordinates.
- LL Decimal: latitude/longitudes in degrees.

## 4 Origin



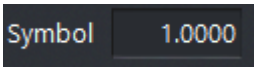

Lower-left corner of the database (if no layer is selected) or the selected layer lower-left corner.
These values can be modified by the user for repositioning the layer.

## 5 Upper-Right




Upper-right corner of the database (if no layer is selected) or the selected layer upper-right corner.
These values can be modified by the user for rescaling the layer.

## 6 Symbol

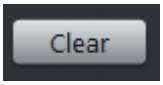



For the selected layer, set the scale of the symbols (bitmaps).

## 7 Add File

Use this button to add the selected SVG file in (1).

## 8 Clear

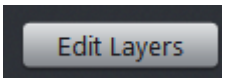



Free the SVG database from all loaded layers.

## 9 Delete

Remove from the SVG database the selected layer of the list (4).

## 10 Edition

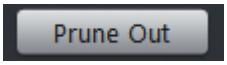



Call the edition property window.

## 11 Save

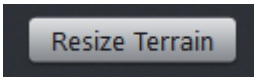
Rewrite the VGM database with the modified layers.

## 12 Optimize



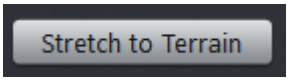

Use this button to prune out all drawings outside the terrain area. This is useful to reduce the size of the database when only a part of the terrain is used.
Do not forget to save the SVG terrain (see Save button above). Use another file name if you want to keep the previous SVG file.

## 13 Terrain Stretch



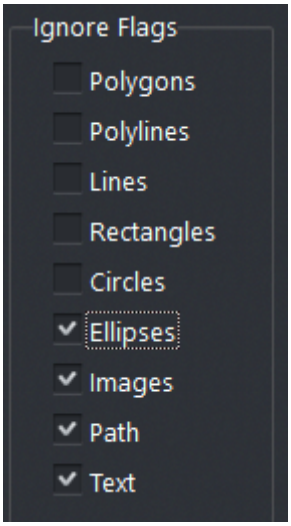

Bounds the terrain min and max corners to the actual SVG database corners.

## 14 Stretch SVG



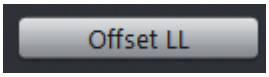

Set the Lower-left and Upper-right corners of all layers to the actual min and max corners of the terrain.
If the SVG database contains one layer or if all layers are natively of the same size and position, use this button. If not, the resulting database will be mixed up.

## 15 Ignore



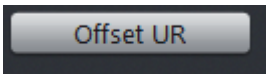

Select here the type of shapes that must be ignored (skipped) in the drawing process.

## 16 Offset LL



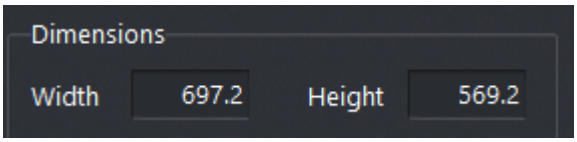

Enabled only when a layer is selected.
Reposition the Lower-left corner of the selected layer to the modified Origin position.

## 17 Offset UR



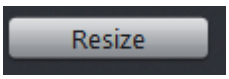

Enabled only when a layer is selected.
Reposition the Upper-right corner of the selected layer to the modified position.
Enable a rescale as the origin (Lower-left) corner will remain unchanged.

## 18 Dimensions



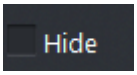

Dimensions in meters of the selected layer (or the full SVG database if no layer selected).
The computation is according to the coordinate system selected.

## 19 Resize




Enabled only when a layer is selected.
Recompute the Upper-right corner according to the new width and height entered by the user.

## 20 Hide




If checked, the SVG database will **not be displayed** on the terrain map.

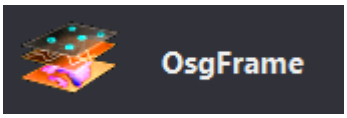
# OsgFrame

OsgFrame display wireframe coming for OSG format files.
vsTASKER does not display textures and only load vertex arrays. This layer is often used as a map support for scenario generation. It can be seen as a handful 2D representation of the 3D editor when OpenSceneGraph database are used.

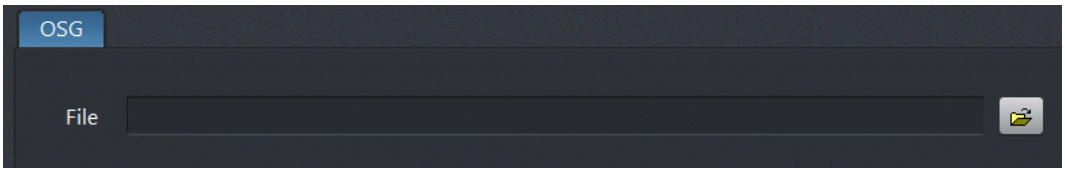
Importing an OsgFrame file is straightforward.
It can be done directly from vsTASKER GUI or from the TerrainBuilder tool (recommended option).

## • How to Use

First, open the OsgFrame property window:

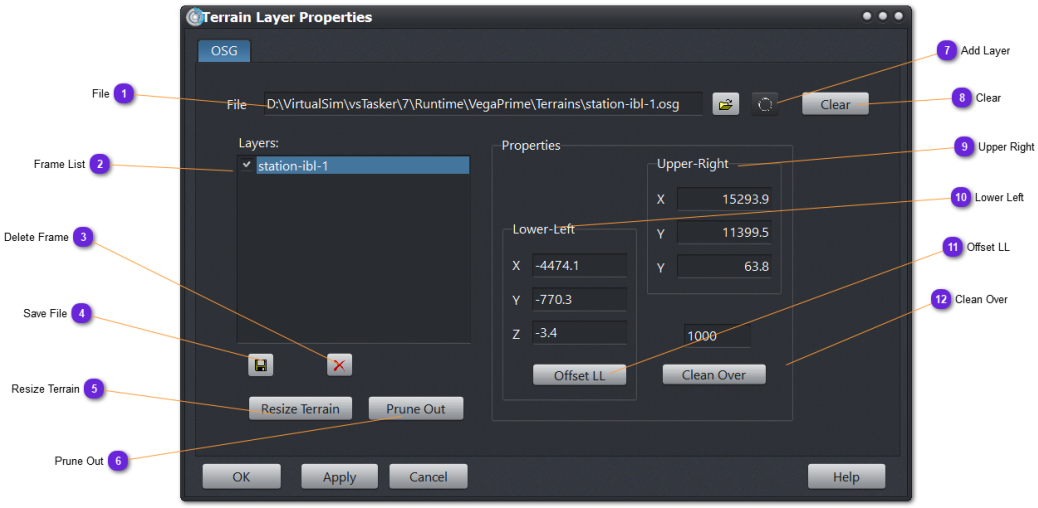



Then select any .osgf file (proprietary format) or .osg (OpenSceneGraph uncompressed format - ive shall be converted using the osgconv.exe tool before import to vsTASKER).

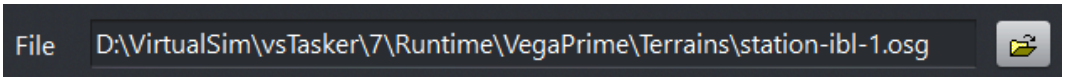



Once loaded, all the vertex arrays are displayed in orange (default color).
The **Resize Terrain** button may be needed to relocate the terrain origin according to the imported area.

# Properties

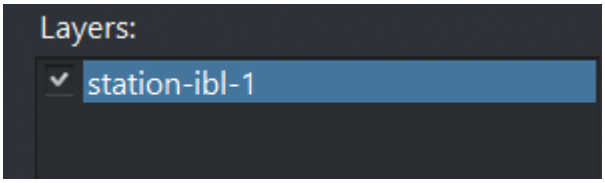



## 1 File




Loaded OsgFrame file. Click on the field or use the open button on the right to select a file to load.
If an osg file is selected, it will be added to the current list, as a new frame.
If an osgf file is selected, the current layer will be first deleted.

## 2 Frame List

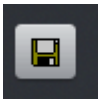



List of all frame layers loaded from the OsgFrame file.
Select any of them for deletion or to display it's properties.
If a layer is selected, Offset LL and Clean Over functions will apply only to it.
Uncheck the layer to ignore it (no more displayed and processed)

## 3 Delete Frame

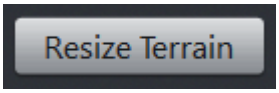
Delete any selected frame above.

## 4 Save File



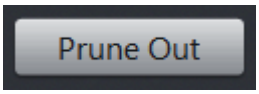
Save the current OsgFrame db into a new file (or overwrite the current one). This is mandatory after an offset or a cleaning (**Clean Over** or **Prune Out**) as the db file is separate from the scenario database.

## 5 Resize Terrain



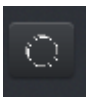

Use this button to move the terrain orgin to the center of the loaded OpenSceneGraph db, and dimension it accordingly.

## 6 Prune Out



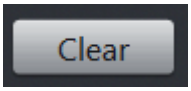

In case the terrain is manually resized to a portion of the OpenSceneGraph db, the OsgFrame db can be reduced by removing all vertex outside the terrain area.
Use Save button to overwrite the file and keep the change.

## 7 Add Layer

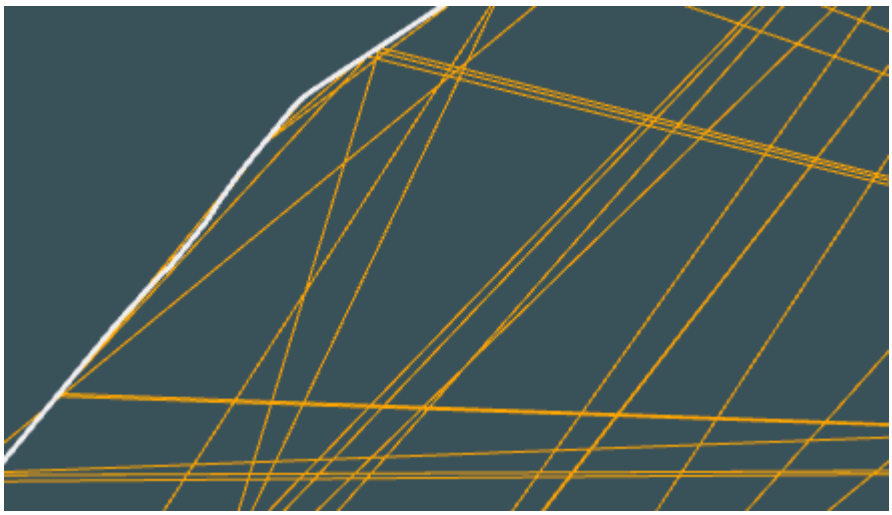


Use this button to add a new .osg file into the current OsgFrame, as a new layer.

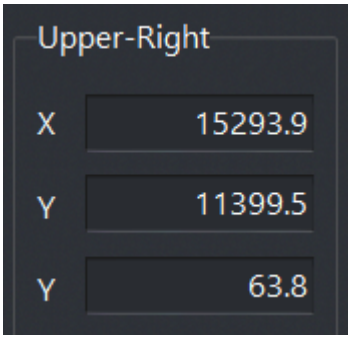
## 8 Clear




Free the OsgFrame memory and remove all the layers.
This does not erase the file unless **Save** is called just after.

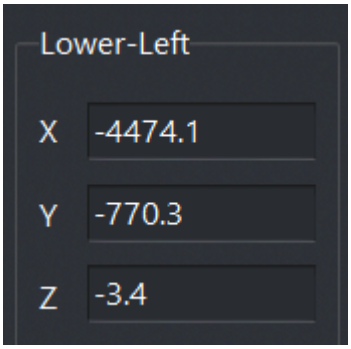


## 9 Upper Right



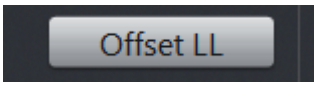

Coordinates of the upper-right corner of the full OsgFrame db or the selected frame if any selected in the list.

## 10 Lower Left



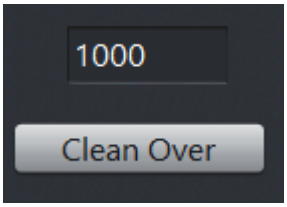

Coordinates of the lower-left corner of the full OsgFrame db or the selected frame if any selected in the list.

## 11 Offset LL




Use this button to offset the lower-left corner of the OsgFrame db or the selected layer (if any) using the value in Lower-Left fields (10).

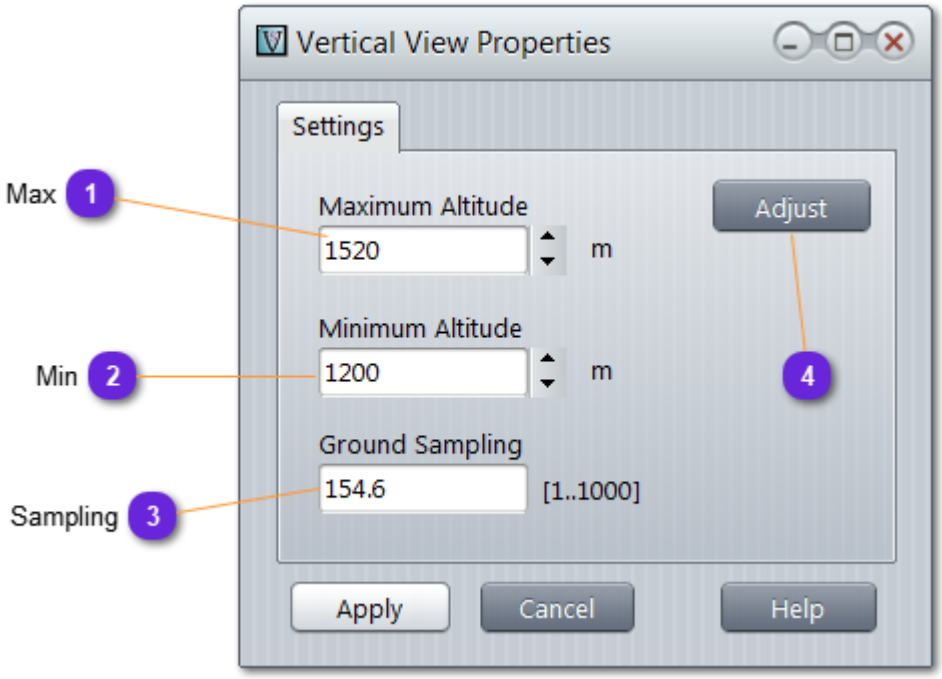
## 12 Clean Over



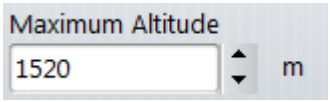

Remove all vertex lines greater than the value (in m) expressed in the above field.
Units are the same as the database one.
Save the db after cleaning if need to keep the changes.

# Vertical View

The vertical view displays a transparent panel on the terrain map to display the vertical terrain below a path or trajectory (when selected) or below entities. Waypoint or Entity altitude are modifiable using the mouse, by selecting it and sliding the position up or down with the mouse button depressed.

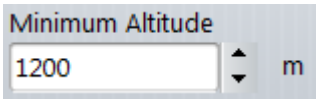




## 1 Max



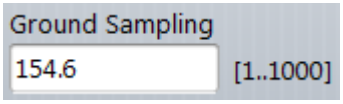

**Maximum** altitude to be displayed on the vertical view window (top).

## 2 Min



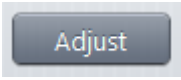

**Minimum** altitude to be displayed on the vertical view window (bottom).

## 3 Sampling




Number of **divisions** applied on the vertical slice to query the ground underneath.
The higher the value, the higher the accuracy of the terrain profile.

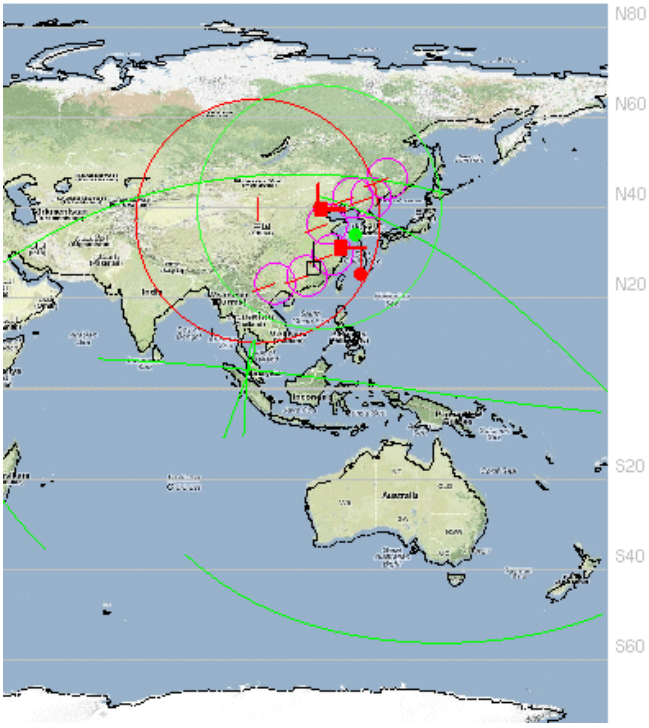
## 4 Adjust




**Automatically** compute the best values for Max, Min and Sampling, according the terrain size and height.

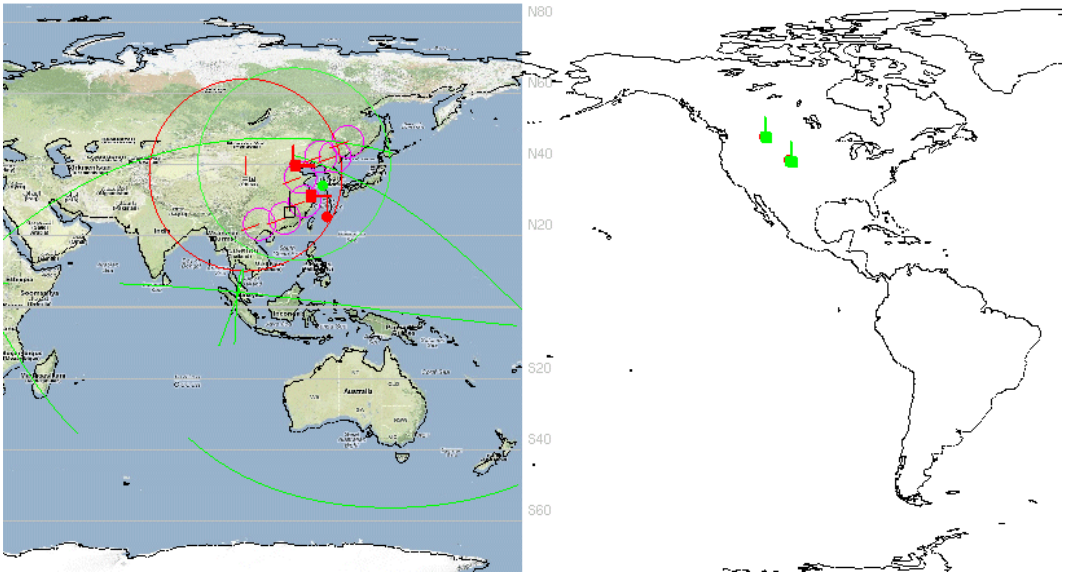
# Pacific Date Line

vsTASKER converts Lat/Lon coordinates into XY according to a flat earth projection. It is very convenient and efficient for small to medium areas but might lead to some problems when dealing with large map areas, mostly when these one are centered around the pacific date change line, when longitudes change from -179 degrees to 180 degrees.

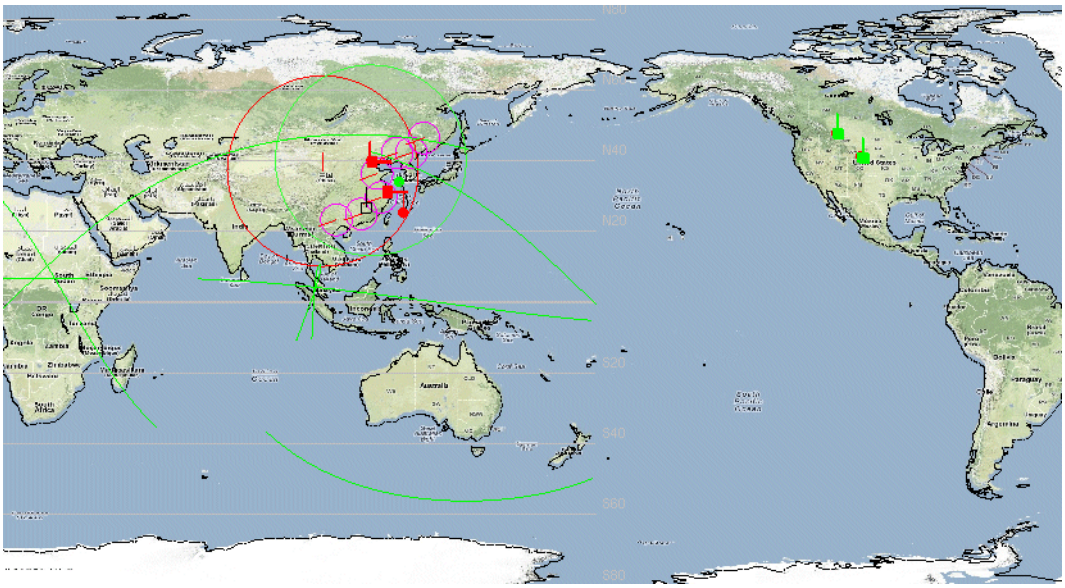



To correct such defect, the solution is to use a proper terrain database: Pacific Coastline (/Data/Coastlines/pacific.cst) database:

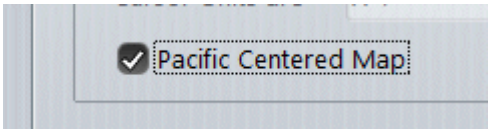
### Pacific Date Line

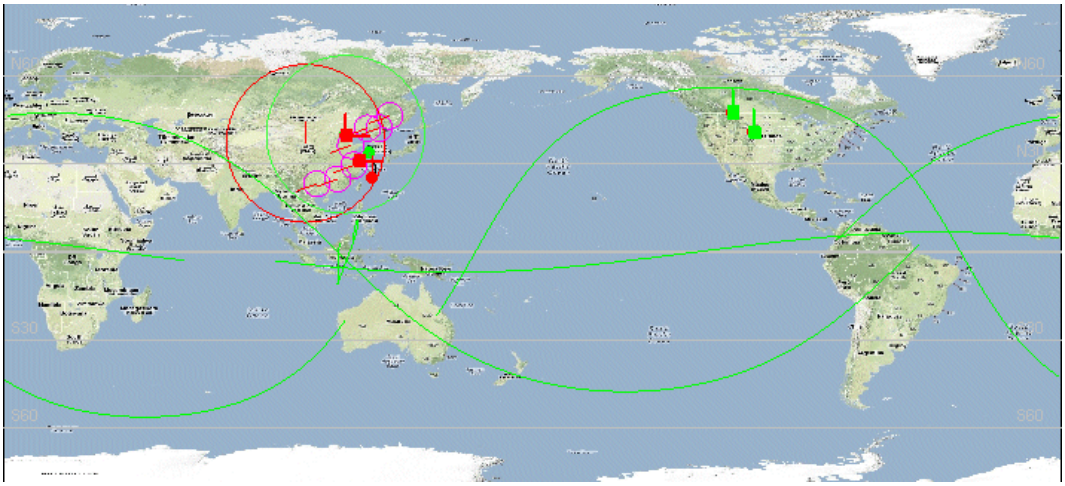



and/or Pacific Map (/Data/Maps/Terrains/pacific.map):



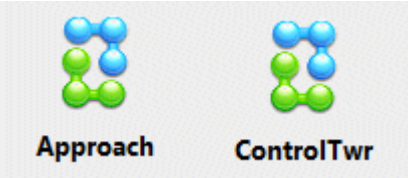
But this is not enough as grid, trajectories or orbits (...) can still remain drawn on the left side of the terrain map.
In Database::Settings::Map, use the Pacific Centered Map option to correct the display.





# Logics

Logics are one of the most useful and basic concept of vsTASKER. A logic is a grafcet that connects various objects to represent a flow of sequences.
Each of the objects allowed in a logic have a particular purpose. Combining and linking these objects together can create simple obvious behavior up to complex and unpredictable ones.



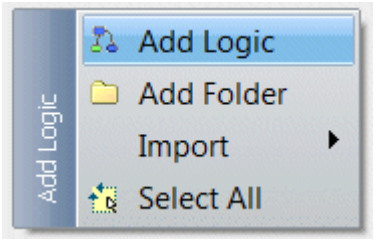


*Refer to the Tutorial to learn how to create your own.*

## • Environment


## • Popup Menu

On the **background**:



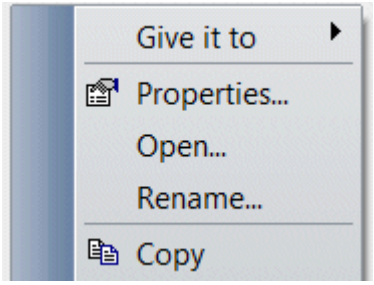

**Add Logic**: create a new Logic in the database. Can also use ✚ from the toolbar.

**Add Folder**: see here

**Import**: list all exported Logics available in /Data/Shared directory

**Select All**: select all Logics of the database (useful for moving symbols or deleting them).

**When** a Logic is **selected**:




**Give it to**: give the current Logic to the Entity listed in the sub list. Faster than to go through the Behavior panel.

**Properties**: open the Logic property window.

**Open**: See the content. Same as double clicking.

**Rename**: Give a new name to the Logic.

**Copy**: Copy the Logic into the clipboard.

**Relink**: Force all dangled arrows to attach to the closest anchor. This is useful after some copy/paste or undelete.

# • Creating Objects

To **add objects** into a logic, you must first **select** which object to add using the **vertical Toolbar** shown here. Then, clicking anywhere on the **Drawing Area** (Diagram) creates the designated object displaying its own symbol.

**Entry Point** (EPoint, green head) and **Exit Point** (XPoint, red head) cannot be added alone. They must be attached to another object. If another object is already selected, clicking the EPoint or XPoint button will immediately create and attach the point to it. Otherwise, cursor will change until an object is designated into the diagram for dropping the point.
For EPoint or XPoint property windows, see here.

Add a Task.

### Logics

 Add a Transition.

 Add a State.

 Add an Action.

 Add a Choice.

 Add a Synchro.

 Add a Splitter.

 Add a Delay.

 Add a Watcher.

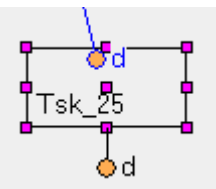
 Add a Text for comment.

*Logic objects does not inherit from the logic. They are embed into the logic (or another group).*

## • Connecting Objects

All objects have one or two **connectors** (orange head below them).
The connector can have several modes:

- Done: orange head, default. Object processing is finished.
- Quit: red head. Object processing is finished and the level (Group, Context, Knowledge or Logic) must be quit.
- Connect: arrow head. When object processing is finished, another object must be instanciated.
- Immediate: allowed for Action only, force the next object to be processed inside the same cycle and not the following one.
- Nothing: connector is not visible. *Nothing* is like *Done* but without graphical representation.




To connect one object to one another, click the connector head with the mouse while maintaining down the left button then drag it toward the other object. When close enough, the target object shows magenta anchors. Release the mouse while close to one of these anchors.

To disconnect one object, select the arrow head and move it away from any anchor, then release.
The arrow changes to orange head (Done mode, default).

*A connector cannot be connected to its parent, except for Task and Group objects. Use the entry point to do so.*

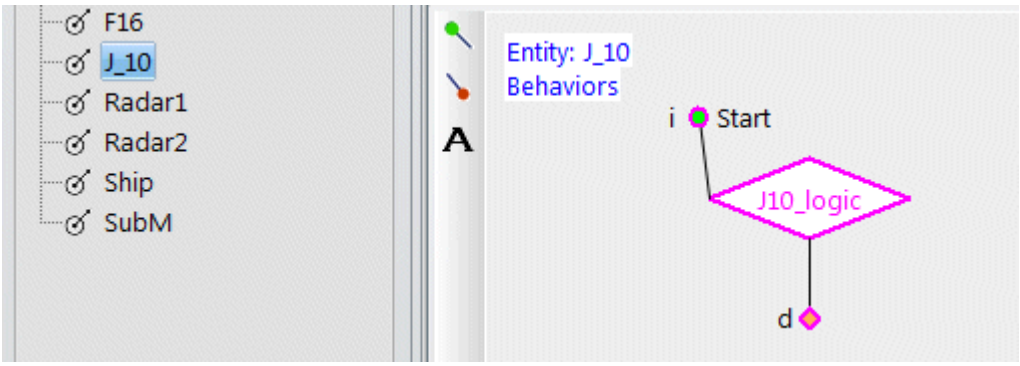
## • Groups

In order to create a group that will gather sub-logics, do the following:

Select one or more objects of a current logic.
Right click and select the **Make Group** option.
A group is then created. It contains all selected objects.
A group can be seen as a logic part. It has an entry point and an exit point.
A group can also embed other groups.
Several groups can run concurrently inside the same logic (or group).
Like a logic, a group has a specific property window. See here.

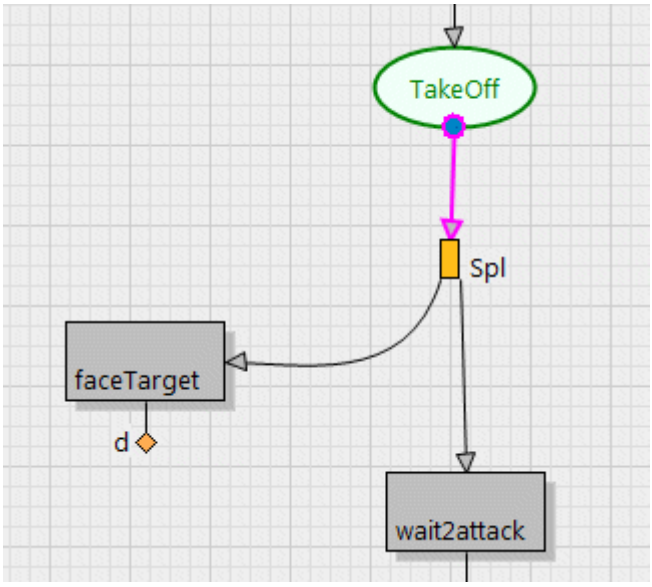
## • Runtime monitoring

During runtime, when an entity is selected, the behavior panel shows all logics currently running (magenta).

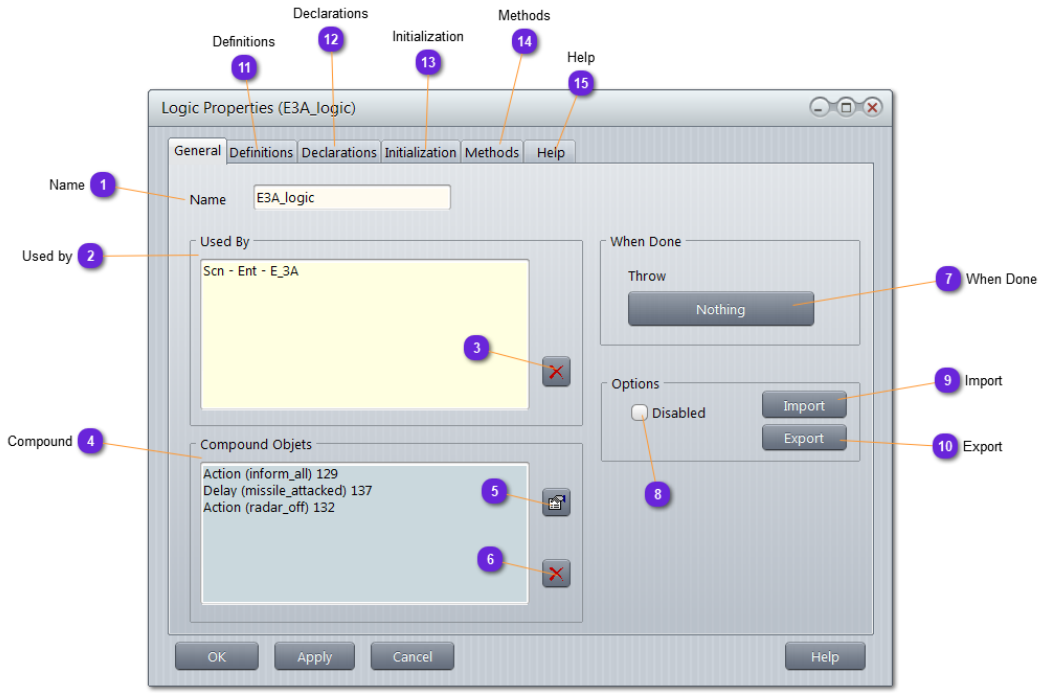



Double click any of these symbol (J10_logic for the example above) to open the definition.
The magenta outlines highlight in real time running objects or transiting connections.
The runtime monitoring sequence visually the logical flow.

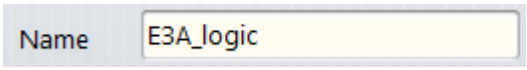
### Logics



# Properties

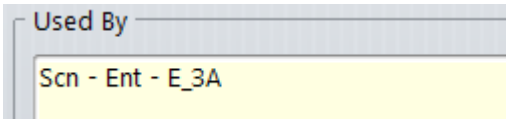



## 1 Name




Name of the Logic. Must be unique.
The class generated will be `E3A_logic_Lgk`.

## 2 Used by



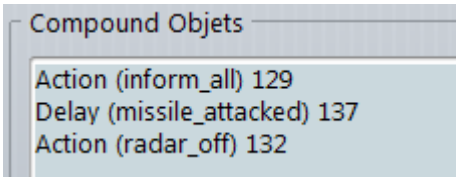

List all the entities that use this Logic in their behavior.
For each line (A - B - C):
A: name of the scenario (*above: Scn*)
B: type of the user: Ent for Entity, Cat for Catalog and Ply for the Player.
C: name of the entity (*above: E_3A*)


*There will be one instance of a Logic (including compound objects) for each entity that uses it in its behavior.*

## 3 Unlink

**Suppress** the **reference** to this Logic for the selected entity of the list (2).

## 4 Compound




List all the objects contained in the Logic.

## 5 Properties

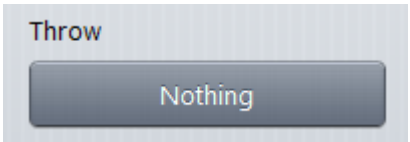
Call the property window of the selected object of the list (4).

## 6 Delete

Remove the selected object of the list (4).

*When removing an object, all connectors to this object will remain dangled.*

## 7 When Done

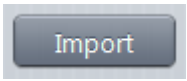



Specify here the Event or Fact that will be triggered when the Logic is exited (optional).

## 8 Disabled


When checked, the Logic will not generate any code and cannot be enabled at runtime.

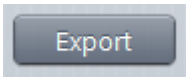
## 9 Import




Use this button to import a previously exported Logic in /Data/Shared.

*An exported Logic has extension .lgk*

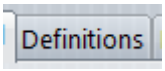
## 10 Export




Use this button to export a Logic for reuse or share between databases or users.
The Logic is exported by default in /Data/Shared but can be saved in any directory or support.

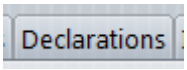
*An exported Logic has extension .lgk*

## 11 Definitions



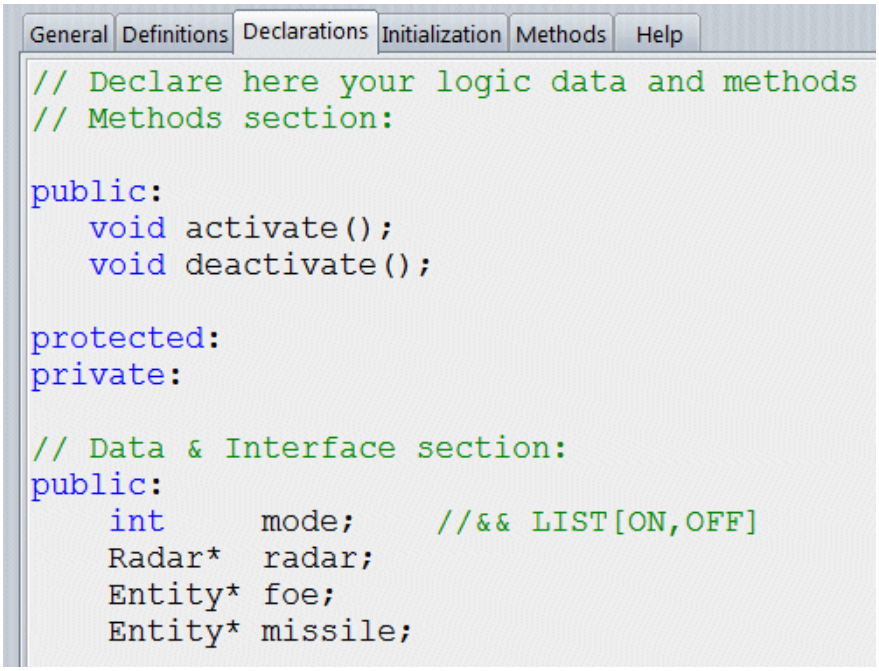

Put here all the includes and structures that might be needed by the code put inside the compound objects.
Definition part is put in the generated header file.

# 12 Declarations




A Logic is a class so, user can add special methods that will be used by compound objects.
Variables can also be declared here.
From a compound object code, using the `activate()` method (see picture below) would be done this way: `L:activate() // L for Logic, only one :`
Any public variable is accessible from the Logic objects (ie: `L:mode`)
Private data/method shall be reserved for Logic storage/process only.

User parameters `//&&` can be defined in a Logic and will generate an interface in the corresponding behavior symbol or the entity. This can be very useful when defining a general purpose Logic that will rely on parameters that will change from one entity to one another.
For example, a Logic **Escape** will be using some parameters like speed, way, duration, etc. and each entity using this Escape Logic will locally overwrite default values of these parameters.



## 13 Initialization

Initialization

Put here the code that is needed at every phase of the Logic. Used mainly to initialize the internal data and parameters used by the compound objects. The initialization function is called for each instance of the Logic.

`INIT`: called once at creation time, for each entity that uses this Logic.
`RESET`: called several times, each time the Logic is activated.

General | Definitions | Declarations | Initialization | Methods | Help

```
// Initialization phase
// function udata() (returning void*) can be used.

switch (_phase) {

  case INIT: {
  } // no break, RESET will also be processed

  case RESET: {
    foe = NULL;
    radar = (Radar*) E:getComponent("Radar");
  } break;
```

*Refer to the Developer Guide for more details on system phases/events.*

## 14 Methods

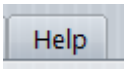
Methods

Put here the methods defined in the Declaration part (if any). Leave it empty otherwise.

General | Definitions | Declarations | Initialization | Methods | Help

```
// Declare here all your logic methods
// Do not forget to write Logic|E3A_logic_Lgk::method(...

void E3A_logic_Lgk::activate() { mode = ON; }
void E3A_logic_Lgk::deactivate() { mode = OFF; }
```

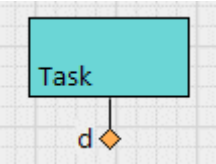
## 15 Help




Any description of the Logic can be put here and will be used for the automatic document generation (see Make Documents)

# Task

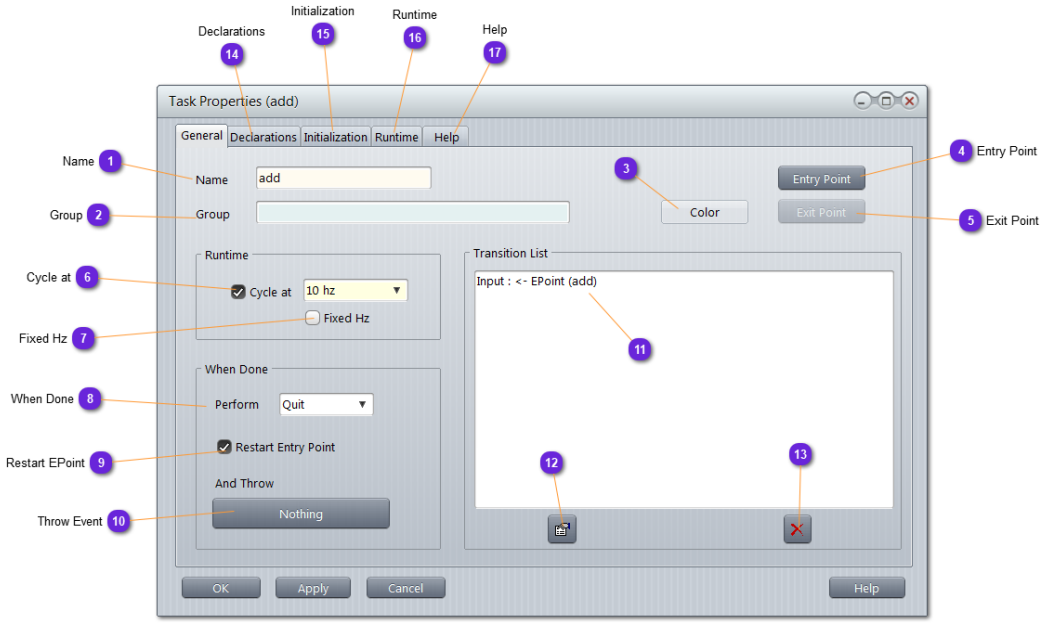
A **Task** is a Logic object whose user code is processed at a given frequency. A **Task** is a runtime object that, once activated, stops the logical flow until terminated.
The user can control the termination condition from either the code itself, from a Transition or using an Exit Point.



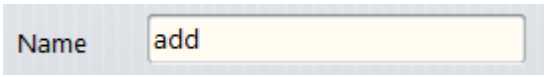

## • Code Hints

`scen()` or `S:` is a macro that returns a pointer to the Scenario instance that holds the Entity instance that holds the Logic instance that holds the current Task.
`ent()` or `E:` is a macro that returns a pointer to the Entity instance that holds the Logic instance that holds the current Task.
`logk()` or `L:` is a macro that returns a pointer to the Logic instance that holds the current Task.
`group()` or `G:` is a macro that returns a pointer to the current Group, if any, that holds the current Task. NULL if none.

# Properties




## 1 Name

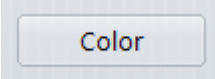



Name of the Task inside the Logic.
This name can be used to retrieve the object (instance) in case user wants to stop it or modify some parameters from source code.

## 2 Group

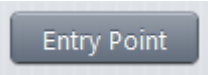

If the Task is part of a Group, it will be named here.
A Group can also be part of another Group, thus, a Group name will be the concatenation of all nested Group names: group1::group2::..::group3 (here, group3 being the deepest one).

## 3 Color



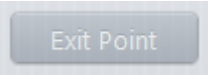

Set here the background color of the Task symbol.
You might also personalize the default color to highlight some specific tasks or associate color with functionality or priority.

## 4 Entry Point




Use this button to access the Entry Point window (can also be opened from the diagram using double click on the Entry Point symbol).

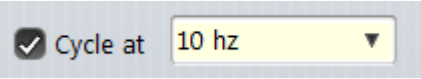
## 5 Exit Point




Use this button to access the Exit Point window (can also be opened from the diagram using double click on the Exit Point symbol).

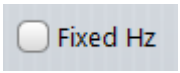

*If grayed out, that means the Point is not defined.*

## 6 Cycle at



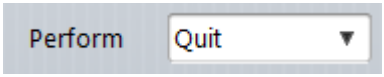

Check this to enable the runtime cycling of the Task (meaning, the runtime part).
The frequency list provided is according to the RTC base definition (an object cannot cycle faster than the RTC).
If you need a user defined frequency not listed, take the highest frequency and add your own frequency divider code.

## 7 Fixed Hz




If you check this option, the frequency selected will be kept whatever the simulation speed (acceleration or deceleration).
The frequency will be against the wall clock and not the RTC clock.

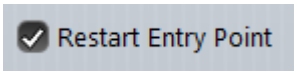
## 8 When Done




Inform about the kind of connection set for the Task after completion (meaning, when `return DONE;` is encountered) in the runtime part:

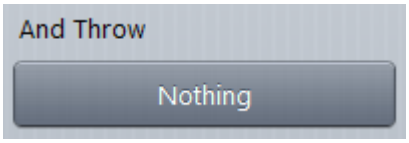
- Connect: the Task is connected to another object of the Logic
- Done: once the Task is finished, the logical flux is done for this one. The Task will not trigger any other object
- Quit: the Logic is left (and returns to the Behavior level) or the Group is left (if the Task is embedded into a Group)
- Nothing: Similar as Done but no event will be triggered

## 9 Restart EPoint



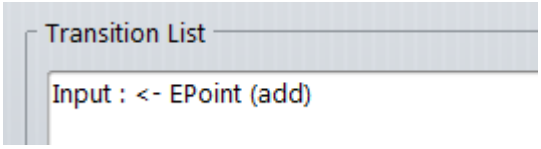

When this is checked, the Entry Point will be reactivated (if not persistent, which should not be with tasks while these one running to prevent it to be reinitialized).
This only happens if Perform mode is not Nothing (see 8).

## 10 Throw Event




When the Task is over (and if Perform mode is not Nothing), an event can be raised.
Click here to set (or unset) the event.

## 11 Transition List




List here all the objects connected to the current Task (in and out).

## 12 Properties

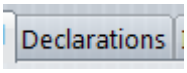
Display the property window of the selected object in the list (11).

## 13 Delete

Delete the selected object of the list (11)

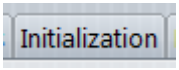
*Cannot undo.*

## 14 Declarations




Add here all the variables needed into the Task.
They will not generate interface so, needless to use `//&&` here.

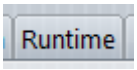
## 15 Initialization




This part will be used at different phases of the simulation.
Use the RESET part to initialize your variables.

*Refer to the Developer Guide for more details on system phases/events.*

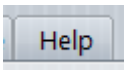
## 16 Runtime




This part is the most important of the task as this part of the code is called at the task selected frequency.
Put here whatever code you like.
The code must return something:

AGAIN: will be called again according to the cycle preference
DONE: finish the cycling, connect to the next object if any and send event if any defined
ABORT: like DONE but do not connect nor send event
QUIT: like DONE but leave the logic or the group if embedded.

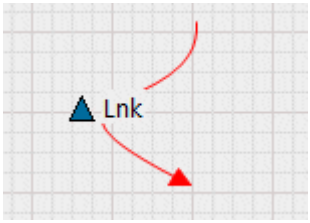
## 17 Help




Put here whatever description of information useful to the designer to understand what the Task is about.
The help part is also used in the hint and the automatic document generation (see Make Documents).

# Transition

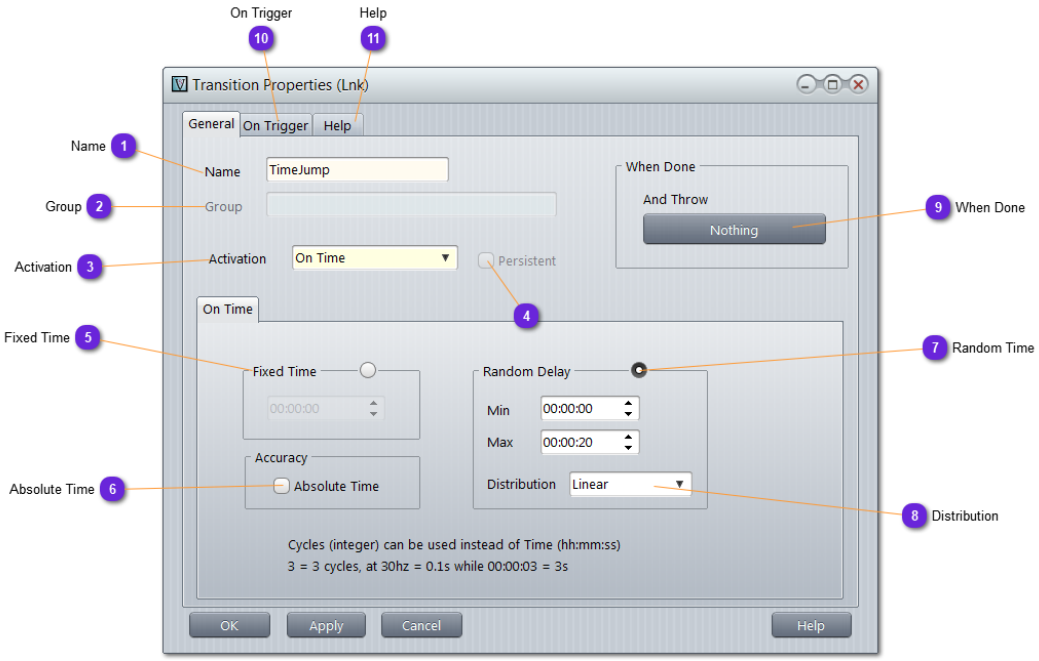
A **Transition** is a logic object used to exit a Task or a Group. A **Transition** contains conditions that are checked constantly as soon as the from object is triggered. The from object is the one the transition is connected from. The to object is the one the transition is connected to. From and to can be the same.



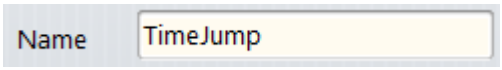

## • Code Hints

`scen()` or `S:` is a macro that returns a pointer to the Scenario instance that holds the Entity instance that holds the Logic instance that holds the current Transition.
`ent()` or `E:` is a macro that returns a pointer to the Entity instance that holds the Logic instance that holds the current Transition.
`logk()` or `L:` is a macro that returns a pointer to the Logic instance that holds the current Transition.
`group()` or `G:` is a macro that returns a pointer to the current Group, if any, that holds the current Transition. NULL if none.
`from()` is a macro that returns a pointer to the from object of the Transition.
`to()` is a macro the returns a pointer to the to object of the Transition.

# On Time




## 1 Name

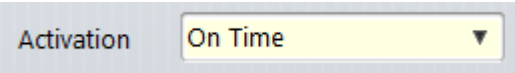



Name of the Transition inside the logic.
This name can be used to retrieve the object (instance) in case user wants to activate it from source code.

## 2 Group


If a Transition is part of a Group, it will be named here.
A Group can also be part of another Group, thus, a Group name will be the concatenation of all nested Group names: group1::group2::..::group3 (here, group3 being the deepest one).

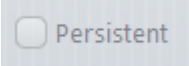
## 3 Activation




Select on which condition the Transition must activate or deactivate the object it is linked to:

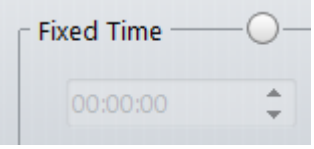
- On Time: See below
- On Reaction: See here
- On Condition: See here

## 4 Persistent



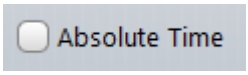

By default, when a Transition is triggered, it will not be reactivated even if the same condition occurs.
If Persistent mode is selected, the Transition reactivates itself automatically after start and is ready at next cycle.
This might lead to unwanted behavior.

## 5 Fixed Time



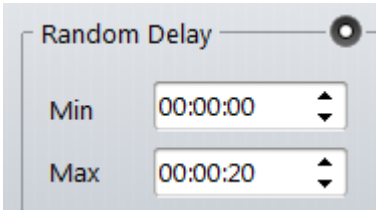

When Fixed Time is selected, as soon as the specified time is elapsed, the Transition triggers.
For i.e: if the Transition is connecting Task A with Task B with a Fixed Time set to10 seconds. If the Logic is triggered 1 minute after the simulation start, then the Transition will be triggered 1 minute 10 seconds after the simulation start and Task A will be left for Task B.

## 6 Absolute Time



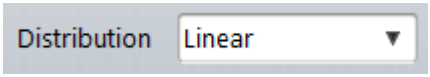

If this option is checked (for Fixed or Random Time), the specified Time is according to the simulation time.
Transition triggers when the simulation time is greater than the given fixed or random time

## 7 Random Time



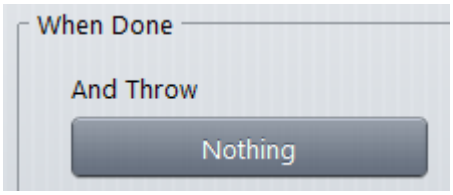

A random value (according to the Distribution) will be taken at Transition initialization. This random value between Min and Max values will act as Fixed Time.

## 8 Distribution



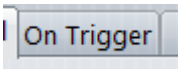

Linear: Random values are distributed equally between Min and Max
Gaussian: Random values are distribution according to a Gaussian curve between Min and Max. Mid-term has the highest probability of occurrence.

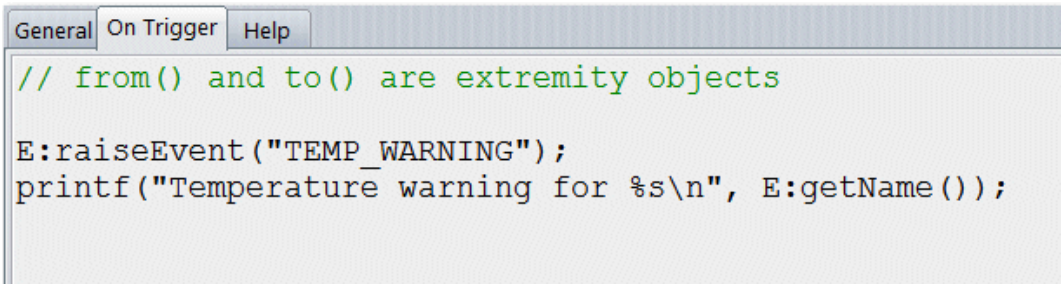
## 9 When Done




Specify here the Event or Fact that will be triggered after activation of the Transition (optional).

## 10 On Trigger

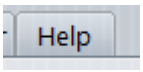



Write here the code you like to be run once after the activation of the Transition.

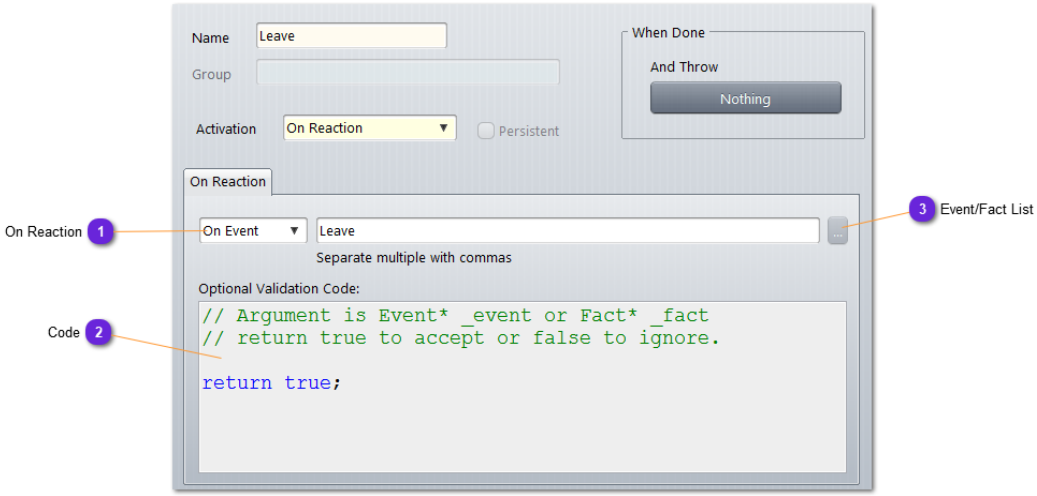

*The user can put whatever C/C++ code he wants there. This code can access all vsTASKER API, all included third-party APIs and all user designed scenario, entity, logic, knowledge and model data of the current database.*

## 11 Help




Put here whatever description of information useful to the designer to understand what the Transition is about.
The help part is also used in the hint and the automatic document generation (see Make Documents).

# On Reaction

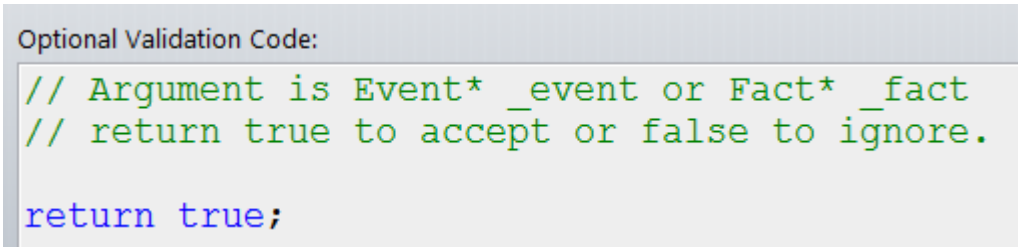



## 1 On Reaction


Specify here the Event (or the Fact existence) that will initiate the triggering of the Point.
In case of an Event, the Transition will register as an observer for events and for each one raised into the Transition scope, triggering will happen.
If case of a Fact, the Transition will continuously check the Logic fact database for existence of any Fact mentioned.

On the text field, write the Event or Fact (if many, separate them with commas) that will trigger the Transition.

## 2 Code

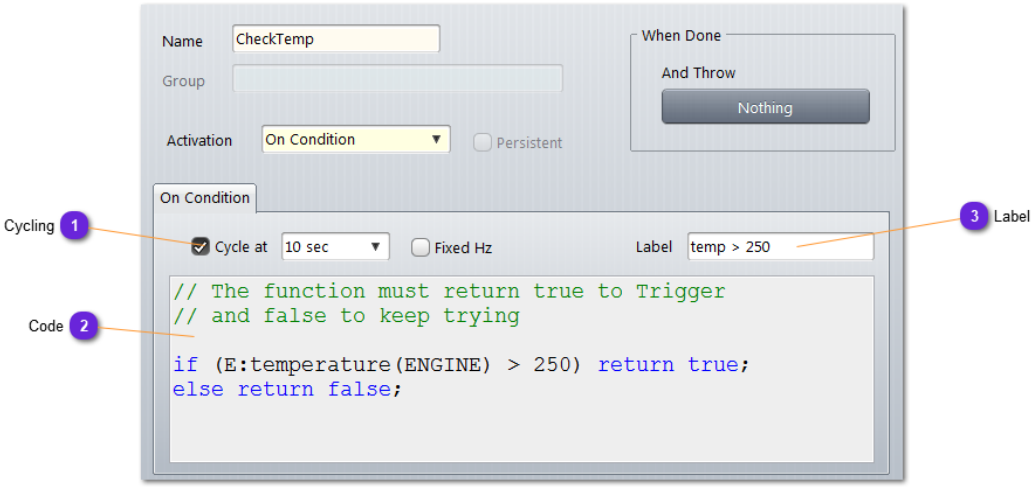



Write here any code that will accept (return `true`) or not (return `false`) the catch of the Event or the existence of a Fact.
This can be any condition the user might decide, for ie.: *Cannot accept the take-off event before engines have run for minimum 7 minutes.*

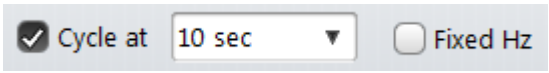
## 3 Event/Fact List

*Not available yet*

# On Condition




## 1 Cycling



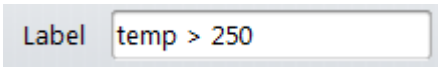

A condition is a piece of code that will be evaluated (called) at a given frequency.
If Fixed Hz is selected, the frequency selected is the wall clock frequency and not the simulation engine clock.
If Cycle at is not selected, the Condition code will be evaluated only once.

## 2 Code

```
// The function must return true to Trigger
// and false to keep trying

if (E:temperature(ENGINE) > 250) return true;
else return false;
```

Put here the condition code that will return either `true` (condition passes, triggering of the Transition) or `false` (condition fails, no trigger)

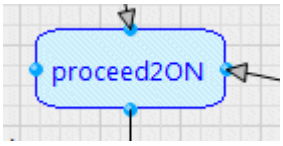
## 3 Label




Optional label that summarize the condition and will be used on the diagram panel.

# State

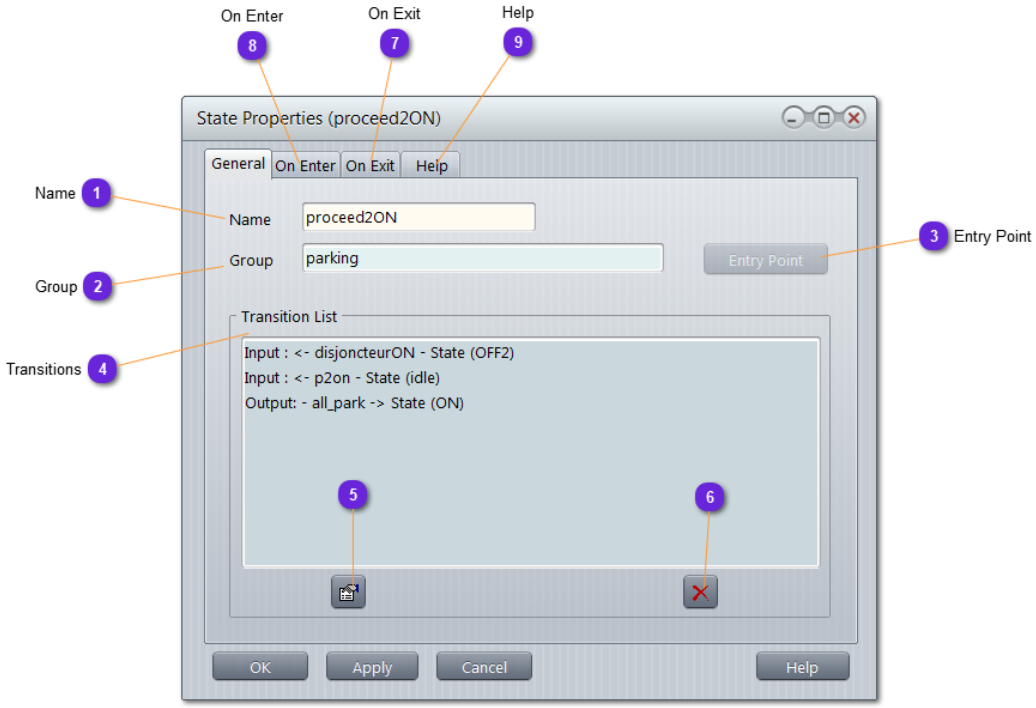
A **State** is a Logic object that contains a user code processed at a given frequency. Once a State is activated, the OnEnter part is executed once and the logical flow stops.
Several Transitions can raise from a State towards other logic objects. The first to be activated will make the State exits and the OnExit part be executed once.



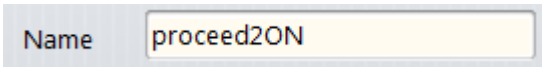

## • Code Hints

`scen()` or `S:` is a macro that returns a pointer to the Scenario instance that holds the Entity instance that holds the Logic instance that holds the current State.
`ent()` or `E:` is a macro that returns a pointer to the Entity instance that holds the Logic instance that holds the current State.
`logk()` or `L:` is a macro that returns a pointer to the Logic instance that holds the current State.
`group()` or `G:` is a macro that returns a pointer to the current Group, if any, that holds the current State. NULL if none.

# Properties




## 1 Name

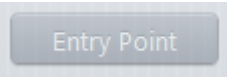



Name of the State inside the logic.
This name can be used to retrieve the object (instance) in case user wants to activate it from source code.

## 2 Group

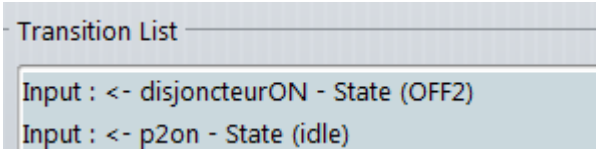

If the State is part of a Group, it will be named here.
A Group can also be part of another Group, thus, a Group name will be the concatenation of all nested Group names: group1::group2::..::group3 (here, group3 being the deepest one).

## 3 Entry Point




Use this button to access the Entry Point window (can also be opened from the diagram using double click on the Entry Point symbol).

## 4 Transitions




List here all the objects connected to the current State (in and out).

## 5 Properties

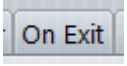
Display the property window of the selected object in the list (4).

## 6 Delete

Delete the selected object of the list.

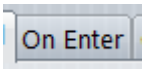

*Cannot undo.*

## 7 On Exit



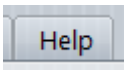

Put here the code that is executed as soon as the State object is left (from a Transition).

## 8 On Enter




Put here the code that is executed as soon as the State object is activated.

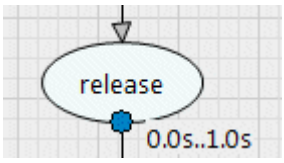
## 9 Help




Put here whatever description of information useful to the designer to understand what the State is about.
The help part is also used in the hint and the automatic document generation (see Make Documents).

# Action

An **Action** is a Logic object that is called once and performs a specific action defined in a user code. An Action is a **one-shot execution**. It is triggered from another object connector or Transition. The difference with a State is that an action always exit and activate its connector. The exit process can be delayed using the embedded Delay (see Properties).



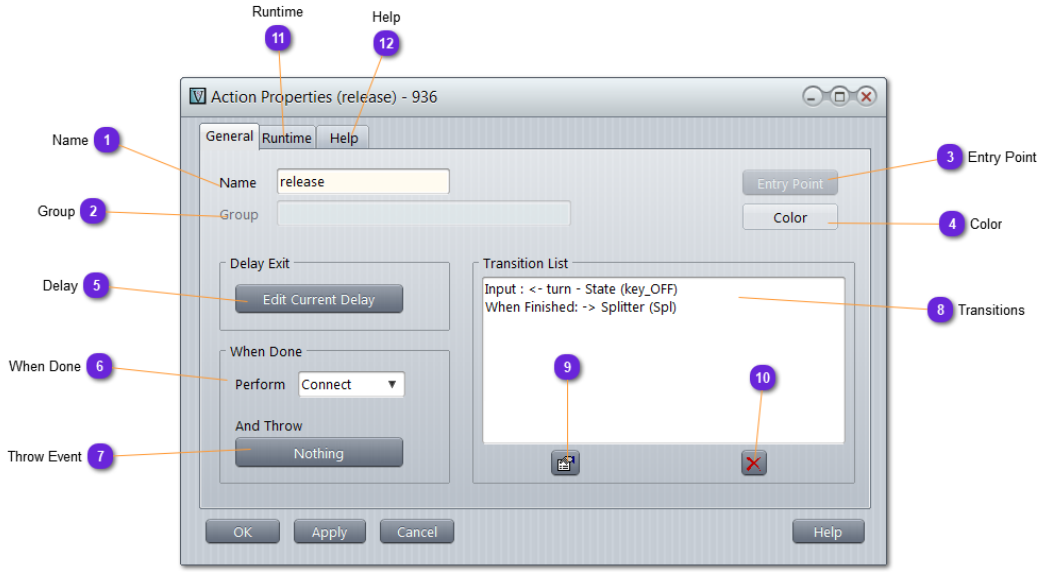


*When Actions are sequenced, each of them occupies one cycle, unless the Immediate option is checked.*

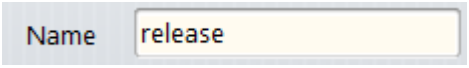
## • Code Hints

`scen()` or `S:` is a macro that returns a pointer to the Scenario instance that holds the Entity instance that holds the Logic instance that holds the current Action.
`ent()` or `E:` is a macro that returns a pointer to the Entity instance that holds the Logic instance that holds the current Action.
`logk()` or `L:` is a macro that returns a pointer to the Logic instance that holds the current Action.
`group()` or `G:` is a macro that returns a pointer to the current Group, if any, that holds the current Action. NULL if none.

# Properties




## 1 Name

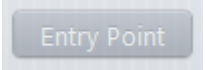



Name of the Action inside the logic.
This name can be used to retrieve the object (instance) in case user wants to activate it from source code.

## 2 Group

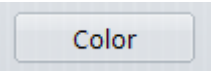

If the Action is part of a Group, it will be named here.
A Group can also be part of another Group, thus, a Group name will be the concatenation of all nested Group names: group1::group2::..::group3 (here, group3 being the deepest one).

## 3 Entry Point



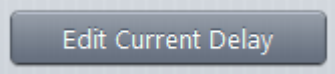

Use this button to access the Entry Point window (can also be opened from the diagram using double click on the Entry Point symbol).

## 4 Color




Set here the background color of the Action symbol.
You might also personalize the default color to highlight some specific actions or associate color with functionality or priority.

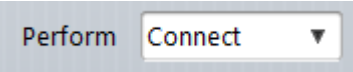
## 5 Delay




Set here the delay between execution of the runtime part and the activation of the connection link.
Normally, without a delay, the connected object will be activated at the next cycle (if Action is not immediate).
With a Delay, it can be postponed.
See here to learn how to set a Delay.


*It has the same behavior as if a Delay object is connected to the Action. Can be seen as a shortcut.*

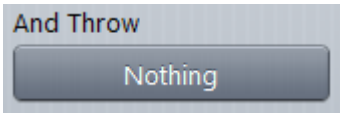
## 6 When Done




Inform about the kind of connection set for the Action after completion (meaning, when `return DONE;` is encountered) in the runtime part:

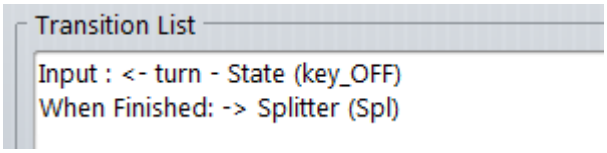
- Connect: the Action is connected to another object of the Logic
- Done: once the Action is finished, the logical flux is done for this one. The Action will not trigger any other object
- Quit: the Logic is left (and returns to the Behavior level) or the group is left (if the Action is embedded into a group)
- Nothing: Similar as Done but no event will be triggered

## 7 Throw Event




When the Action is over (and if Perform mode is not Nothing), an event can be raised.
Click here to set (or unset) the event.

## 8 Transitions




List here all the objects connected to the current Action (in and out).

## 9 Properties

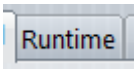
Display the property window of the selected object in the list (8).

## 10 Delete

Delete the selected object of the list.

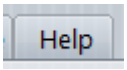

*Cannot undo.*

## 11 Runtime




The runtime part of an Action is only executed once, when the Action triggers.
The user can put whatever C/C++ code he wants there.
This code can access all vsTASKER API, all included third-party APIs and all user designed Scenario, Entity, Logic, Knowledge and Model data of the current database.

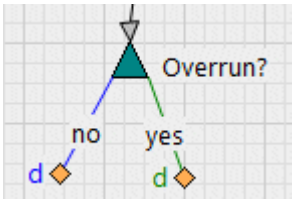
## 12 Help




Put here whatever description of information useful to the designer to understand what the Action is about.
The help part is also used in the hint and the automatic document generation (see Make Documents).

# Choice

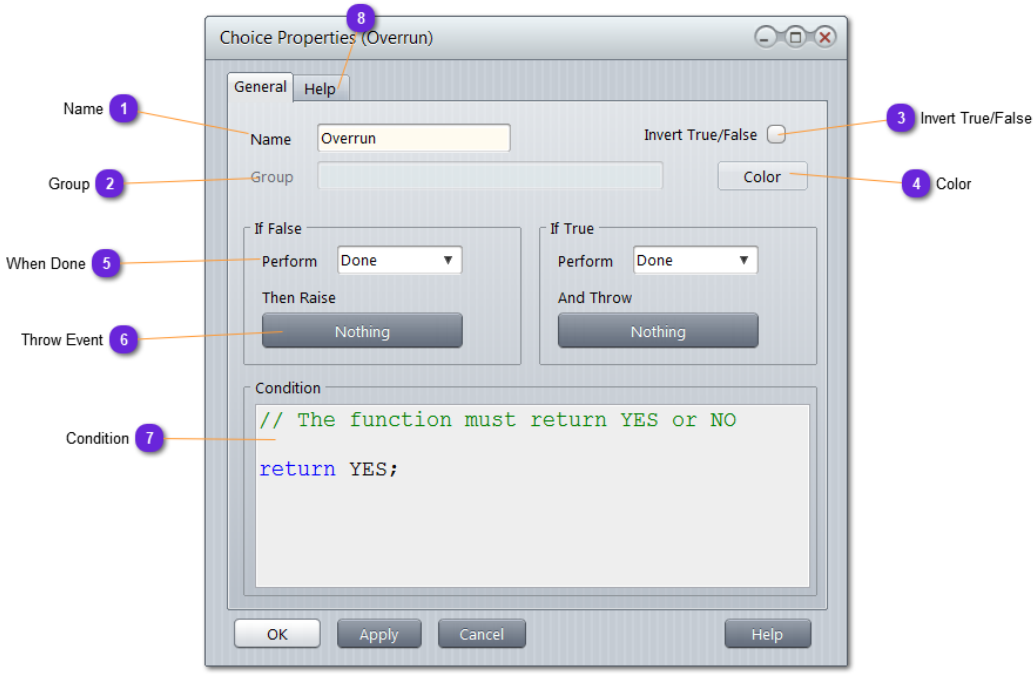
A **Choice** is a Logic object that selects one of the two connectors according to a user defined (C/C++) condition. A Choice is triggered by another object connector or a Transition.
It is used to direct the logical flow.
By default the C/C++ condition returns YES, activating the YES connector.



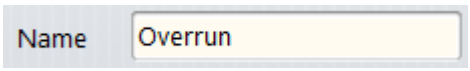

## • Code Hints

`scen()` or `S:` is a macro that returns a pointer to the Scenario instance that holds the Entity instance that holds the Logic instance that holds the current Choice.
`ent()` or `E:` is a macro that returns a pointer to the Entity instance that holds the Logic instance that holds the current Choice.
`logk()` or `L:` is a macro that returns a pointer to the Logic instance that holds the current Choice.
`group()` or `G:` is a macro that returns a pointer to the current Group, if any, that holds the current Choice. NULL if none.

# Properties




## 1 Name

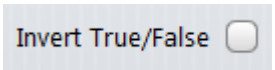



Name of the Choice inside the logic.
This name can be used to retrieve the object (instance) in case user wants to activate it from source code.

## 2 Group

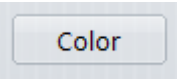

If the Choice is part of a Group, it will be named here.
A Group can also be part of another Group, thus, a Group name will be the concatenation of all nested Group names: group1::group2::..::group3 (here, group3 being the deepest one).

## 3 Invert True/False



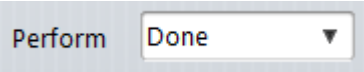

Inverses graphically the two connectors. Can be useful to avoid crossing lines

## 4 Color




Set here the background color of the Choice symbol.
You might also personalize the default color to highlight some specific choices or associate color with functionality or priority.

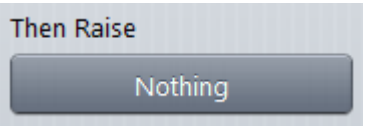
## 5 When Done




Inform about the kind of connection set for the Choice after one leg (YES or NO) is triggered (meaning, when `return YES` or `NO` is encountered) in the runtime part:

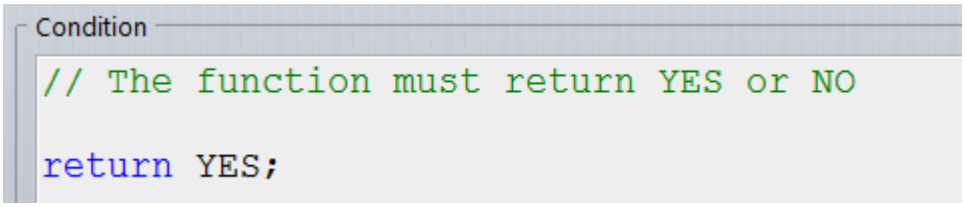
- Connect: the Choice is connected to another object of the Logic
- Done: once the Choice is finished, the logical flux is done for this one. The Choice will not trigger any other object
- Quit: the Logic is left (and returns to the Behavior level) or the group is left (if the Choice is embedded into a group)
- Nothing: Similar as Done but no event will be triggered

## 6 Throw Event



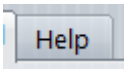

When the Choice is over (and if Perform mode is not Nothing), an event can be raised.
Click here to set (or unset) the event.

## 7 Condition




Put here the condition code that will return either `YES` (condition passes, triggering the **Yes** leg) or `NO` (condition fails, triggering the **No** leg).
The code can contain more than just a `if (..) then .. else ..;` structure. Variables can be defined here (local one) and computation can be performed before returning the final value. The Condition code can access all vsTASKER API, all included third-party APIs and all user designed Scenario, Entity, Logic, Knowledge and Model data of the current database.

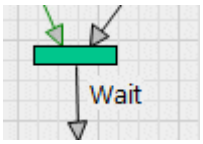
## 8 Help




Put here whatever description of information useful to the designer to understand what the Choice is about.
The help part is also used in the hint and the automatic document generation (see Make Documents).

# Synchro

A **Synchro** is a Logic object that stops the logical flow until **all** input connectors have been triggered. A Synchro is handful when several flows must meet somewhere in time before continuing the process. A Synchro can be seen as a Logic meeting point (synchronization point).



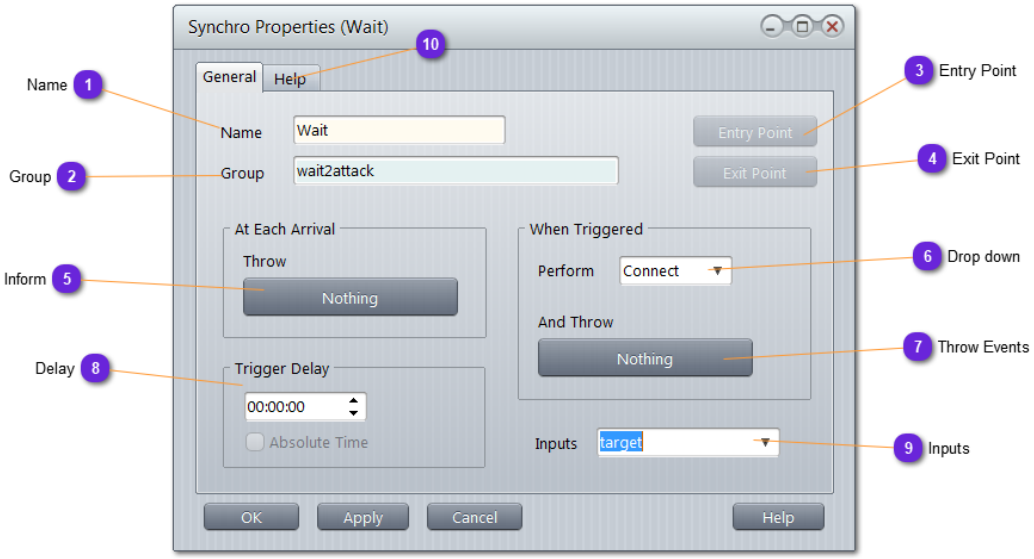


*A Synchro can have an Exit Point. In such case, if the Synchro is active and waiting for all input connectors to be triggered, as soon as the XPoint is activated, the Synchro will trigger regardless of the number of valid inputs. This is handful to automatically resolve dead locks with a kind-of timeout.*

## • Code Hints

`scen()` or `S:` is a macro that returns a pointer to the Scenario instance that holds the Entity instance that holds the Logic instance that holds the current Synchro.
`ent()` or `E:` is a macro that returns a pointer to the Entity instance that holds the Logic instance that holds the current Synchro.
`logk()` or `L:` is a macro that returns a pointer to the Logic instance that holds the current Synchro.
`group()` or `G:` is a macro that returns a pointer to the current Group, if any, that holds the current Synchro. NULL if none.

# Properties



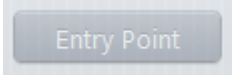

## 1 Name


Name of the Synchro inside the logic.
This name can be used to retrieve the object (instance) in case user wants to activate it from source code.

## 2 Group

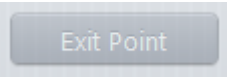

If the Synchro is part of a Group, it will be named here.
A Group can also be part of another Group, thus, a Group name will be the concatenation of all nested Group names: group1::group2::..::group3 (here, group3 being the deepest one).

## 3 Entry Point




Use this button to access the Entry Point window (can also be opened from the diagram using double click on the Entry Point symbol).

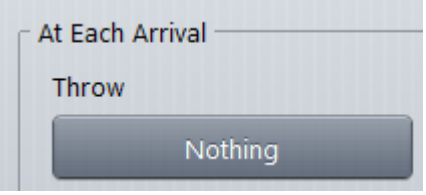
## 4 Exit Point




Use this button to access the Exit Point window (can also be opened from the diagram using double click on the Exit Point symbol).

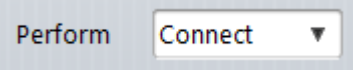
*If grayed out, that means the Point is not defined.*

## 5 Inform




Each time one input connection is triggered, the optional event is raised. Click here to set (or unset) the event.

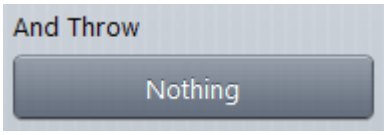
## 6 Drop down




Inform about the kind of connection set for the Synchro when all inputs have triggered.

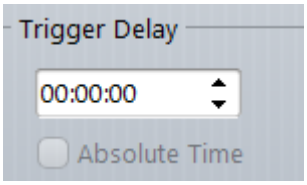
- Connect: the Synchro is connected to another object of the Logic
- Done: once the Synchro is finished, the logical flux is done for this one. The Synchro will not trigger any other object
- Quit: the Logic is left (and returns to the Behavior level) or the group is left (if the Synchro is embedded into a group)
- Nothing: Similar as Done but no event will be triggered

## 7 Throw Events




When the Synchro is over (and if Perform mode is not Nothing), an event can be raised.
Click here to set (or unset) the event.

## 8 Delay



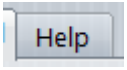

When all inputs have triggered, the Synchro connector is activated after the delay time is elapsed.
When Absolute Time option is checked, the entered time is compared with the simulation time. When simulation time greater, connector triggers.

## 9 Inputs


List all input connections (including Entry/Exit Points).


*Read only drop down.*

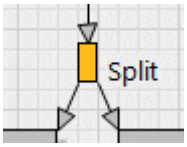
## 10 Synchro




Put here whatever description of information useful to the designer to understand what the Synchro is about.
The help part is also used in the hint and the automatic document generation (see Make Documents).

# Splitter

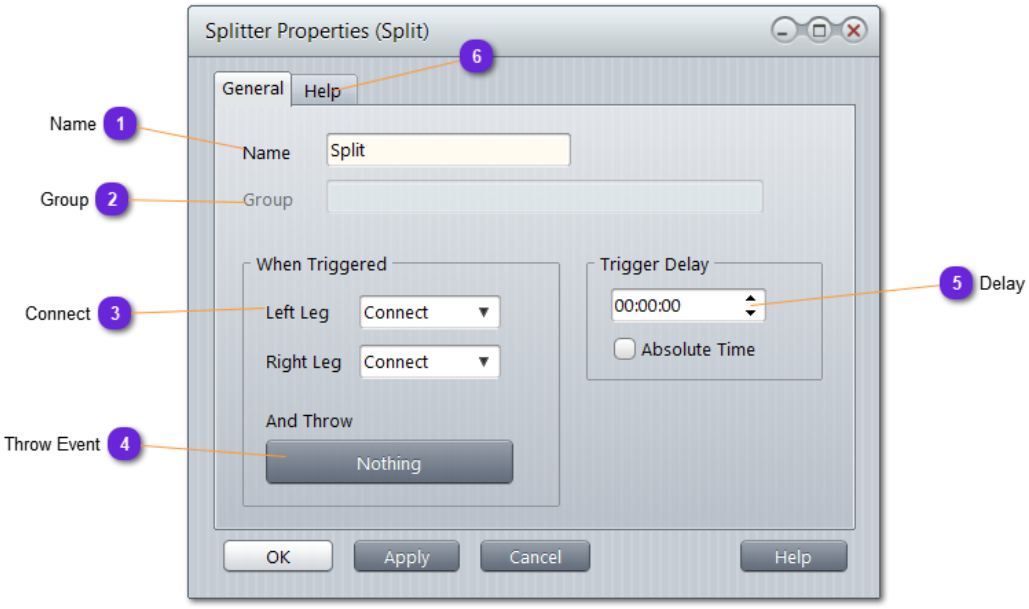
A **Splitter** is a Logic object that splits (parallelize) the logical flow into two flows. Both branches are practically processed at the same time. The Splitter object does not create two threads so there is no way to know which branch is processed first, but both connected objects are processed during the same cycle.
Once the two branches have been activated, they will continue flowing until they end or meet again on a Synchro.




## • Code Hints

`scen()` or `S:` is a macro that returns a pointer to the Scenario instance that holds the Entity instance that holds the Logic instance that holds the current Splitter.
`ent()` or `E:` is a macro that returns a pointer to the Entity instance that holds the Logic instance that holds the current Splitter.
`logk()` or `L:` is a macro that returns a pointer to the Logic instance that holds the current Splitter.
`group()` or `G:` is a macro that returns a pointer to the current Group, if any, that holds the current Splitter. NULL if none.

# Properties



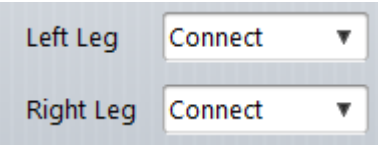

## 1 Name


Name of the Splitter inside the logic.
This name can be used to retrieve the object (instance) in case user wants to activate it from source code.

## 2 Group


If the Splitter is part of a Group, it will be named here.
A Group can also be part of another Group, thus, a Group name will be the concatenation of all nested Group names: group1::group2::..::group3 (here, group3 being the deepest one).

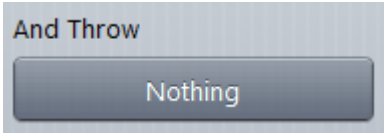
## 3 Connect




Inform about the kind of connection set for the Splitter both legs.

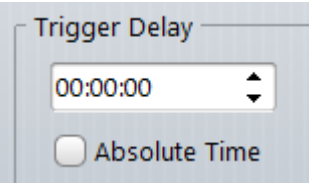
- Connect: the Splitter leg is connected to another object of the Logic
- Done: once the Splitter leg is finished, the logical flux is done for this one. The Splitter leg will not trigger any other object
- Quit: the Logic is left (and returns to the Behavior level) or the group is left (if the Splitter leg is embedded into a group)
- Nothing: Similar as Done but no event will be triggered

## 4 Throw Event



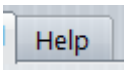

Each time one input connection is triggered, the optional event is raised. Click here to set (or unset) the event.

## 5 Delay




The Splitter leg connectors are activated after the delay time is elapsed. When Absolute Time option is checked, the entered time is compared with the simulation time. When simulation time greater, connection legs trigger.

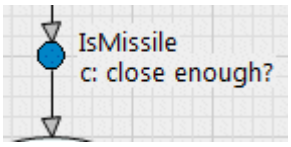
# 6 Help




Put here whatever description of information useful to the designer to understand what the Splitter is about.
The help part is also used in the hint and the automatic document generation (see Make Documents).

# Delay

A **Delay** is a Logic object that stops the logical flow until the embedded condition occurs. The condition can be a random time delay, a specific time delay, an event or a fact or a user-condition. The first condition to occur triggers the Delay.



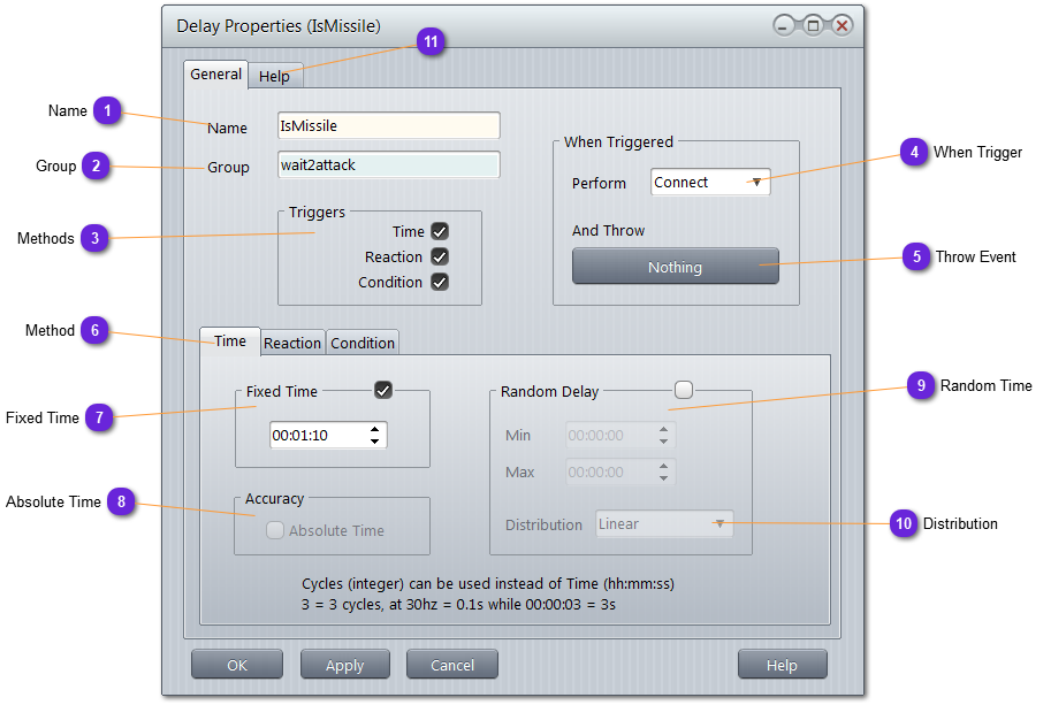


*If a Delay has no triggering condition, it will act as a blocking node. Be careful to always define a trigger/condition that will eventually pass.*

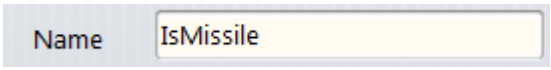
## • Code Hints

`scen()` or `S:` is a macro that returns a pointer to the Scenario instance that holds the Entity instance that holds the Logic instance that holds the current Delay.
`ent()` or `E:` is a macro that returns a pointer to the Entity instance that holds the Logic instance that holds the current Delay.
`logk()` or `L:` is a macro that returns a pointer to the Logic instance that holds the current Delay.
`group()` or `G:` is a macro that returns a pointer to the current Group, if any, that holds the current Delay. NULL if none.

# On Time

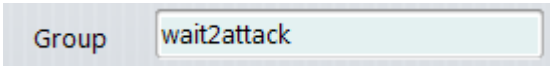



## 1 Name



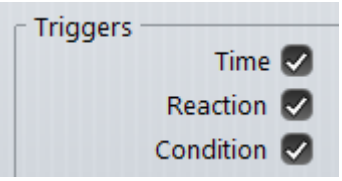

Name of the Delay inside the logic.
This name can be used to retrieve the object (instance) in case user wants to activate it from source code.

## 2 Group




If the Delay is part of a Group, it will be named here.
A Group can also be part of another Group, thus, a Group name will be the concatenation of all nested Group names: group1::group2::..::group3 (here, group3 being the deepest one).

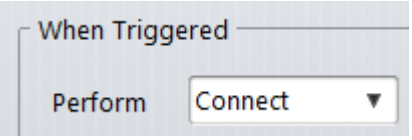
## 3 Methods




List the type of delay methods that has been setup.

*These check boxes are read only. Use the Time/Reaction/Condition panes and activate the method whenever needed.*

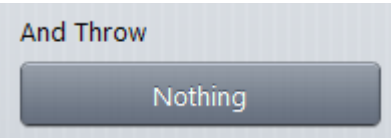
## 4 When Trigger




Inform about the kind of connection set for the Delay connector.

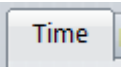
- Connect: the Delay is connected to another object of the Logic
- Done: once the Delay is finished, the logical flux is done for this one. The Delay will not trigger any other object
- Quit: the Logic is left (and returns to the Behavior level) or the group is left (if the Delay is embedded into a group)
- Nothing: Similar as Done but no event will be triggered

## 5 Throw Event



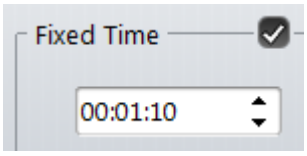

Each time one input connection is triggered, the optional event is raised. Click here to set (or unset) the event.

## 6 Method



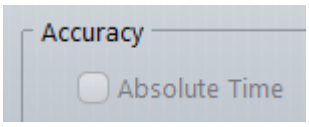

Select this tab to setup a time based delay. See below.

## 7 Fixed Time



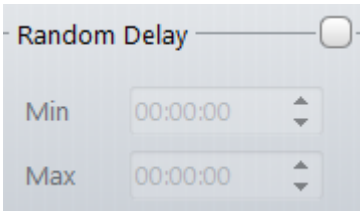

When Fixed Time is selected, as soon as the specified time is elapsed, the Delay triggers.
For i.e: if the Delay is inside a Logic with a Fixed Time set to 10 seconds. If the Logic is triggered 1 minute after the simulation start, then the Delay will be triggered 1 minute 10 seconds after the simulation start.

## 8 Absolute Time



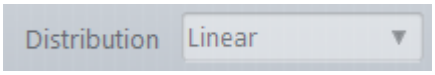

If this option is checked (for Fixed or Random Time), the specified Time is according to the simulation time.
Delay triggers when the simulation time is greater than the given fixed or random time.

## 9 Random Time



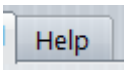

A random value (according to the Distribution) will be taken at Delay initialization. This random value between Min and Max values will act as Fixed Time.

## 10 Distribution



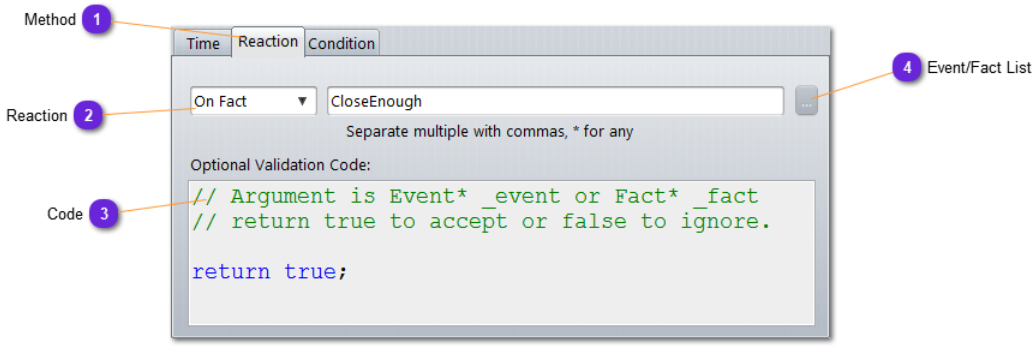

Linear: Random values are distributed equally between Min and Max
Gaussian: Random values are distribution according to a Gaussian curve between Min and Max. Mid-term has the highest probability of occurrence.

## 11 Help

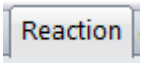



Put here whatever description of information useful to the designer to understand what the Delay is about.
The help part is also used in the hint and the automatic document generation (see Make Documents).

# On Reaction

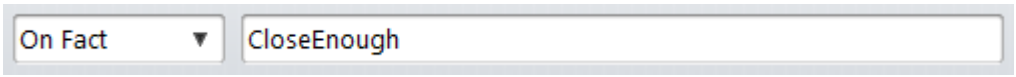



## 1 Method




Select this tab to setup an Event or Fact based delay.

## 2 Reaction




Specify here the Event (or the Fact existence) that will initiate the triggering of the Delay.
In case of an Event, the Delay will register as an observer for events and for each one raised into the Delay scope, triggering will happen.
If case of a Fact, the Delay will continuously check the Logic fact database for existence of any Fact mentioned.

On the text field, write the Event or Fact (if many, separate them with commas) that will trigger the Delay.

## 3 Code

Optional Validation Code:

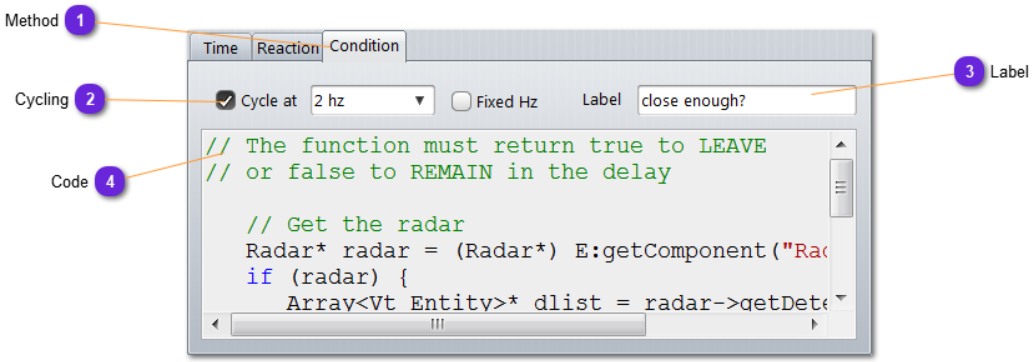
```
// Argument is Event* _event or Fact* _fact
// return true to accept or false to ignore.

return true;
```

Write here any code that will accept (return `true`) or not (return `false`) the catch of the Event or the existence of a Fact.
This can be any condition the user might decide, for ie.: *Cannot accept the take-off event before engines have run for minimum 7 minutes.*

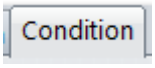
## 4 Event/Fact List

*Not available yet.*

# On Condition




## 1 Method

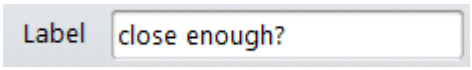



Select this tab to setup code condition based delay.

## 2 Cycling


A condition is a piece of code that will be evaluated (called) at a given frequency.
If Fixed Hz is selected, the frequency selected is the wall clock frequency and not the simulation engine clock.
If Cycle at is not selected, the Condition code will be evaluated only once.

## 3 Label




Optional label that summarize the condition and will be used on the diagram panel.

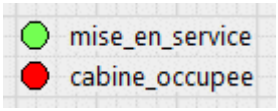
## 4 Code

```
// The function must return true to LEAVE
// or false to REMAIN in the delay
```

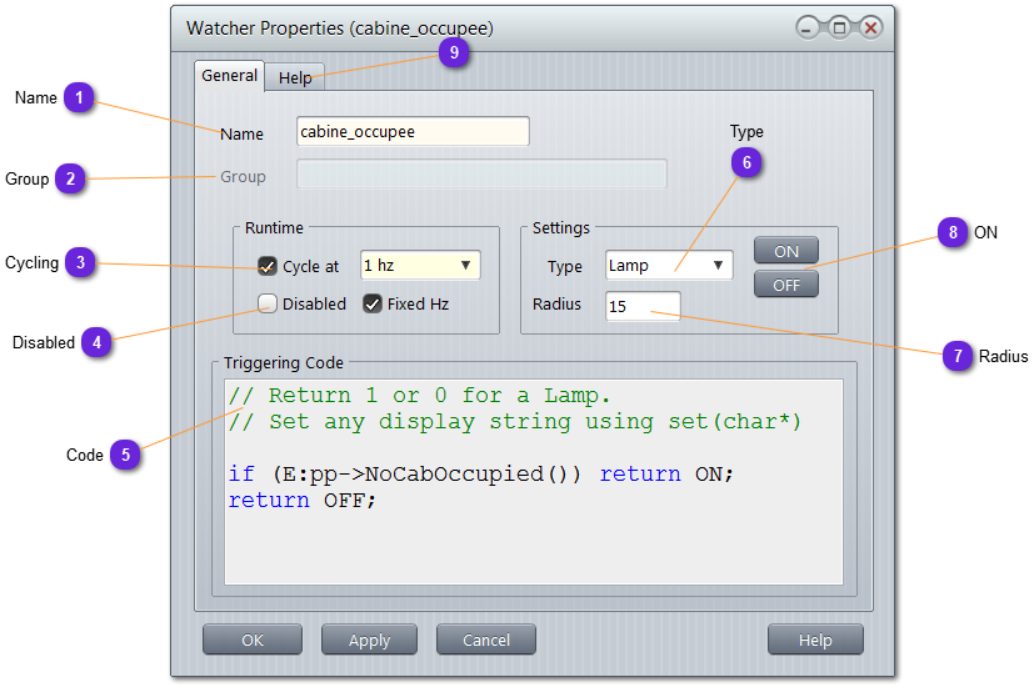
Put here the condition code that will return either `true` (condition passes, triggering of the Delay) or `false` (condition fails, no trigger)

# Watcher

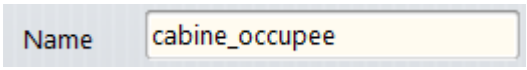
A watcher is a convenient object to monitor some values or complex parameters of the Logic (although it can be any value of the simulation).
It is a GUI object that is automatically fed by the SIM engine during runtime.



# Properties




## 1 Name

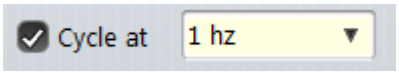



Name of the Watcher inside the logic.
This name can be used to retrieve the object (instance) in case user wants to activate it from source code.

## 2 Group

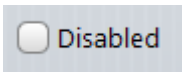

If the Watcher is part of a Group, it will be named here.
A Group can also be part of another Group, thus, a Group name will be the concatenation of all nested Group names: group1::group2::..::group3 (here, group3 being the deepest one).

## 3 Cycling



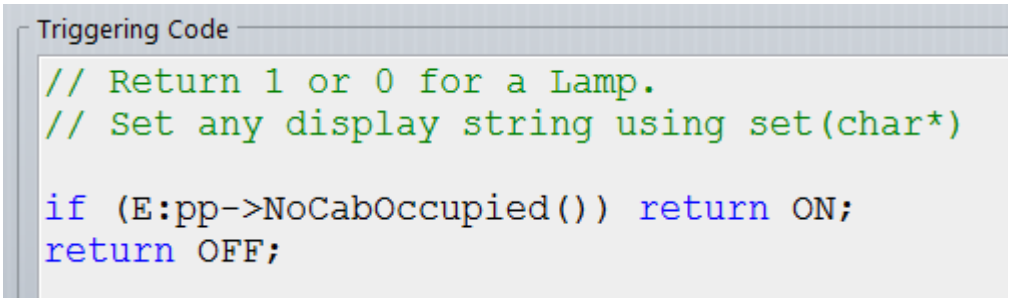

The triggering code (6) will be evaluated (called) at a given frequency.
If Fixed Hz is selected, the frequency selected is the wall clock frequency and not the simulation engine clock.
If Cycle at is not selected, the Condition code will be evaluated only once.

## 4 Disabled



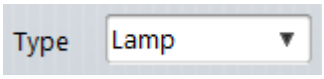

When checked, the Watcher will be disabled. No code will be generated.
No need to remove the object from a Logic definition. Disabling it is enough because it is simply skipped by the code generator.

## 5 Code




Put here the code processed by the SIM and returning either ON or OFF, to switch ON or OFF the Watcher lamp.

## 6 Type



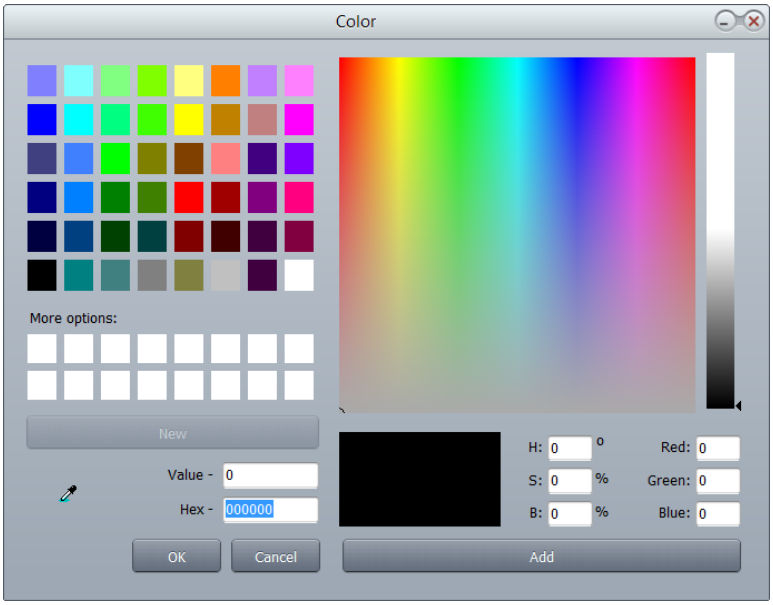

Select the type of the watcher:
Lamp: circular object that has two color states: ON and OFF

## 7 Radius

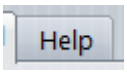

Size of the graphical object on the GUI.

## 8 ON


Select for each state (ON and OFF) the color that will be used at runtime to color the object.



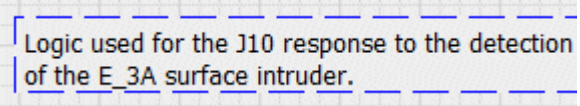

## 9 Help



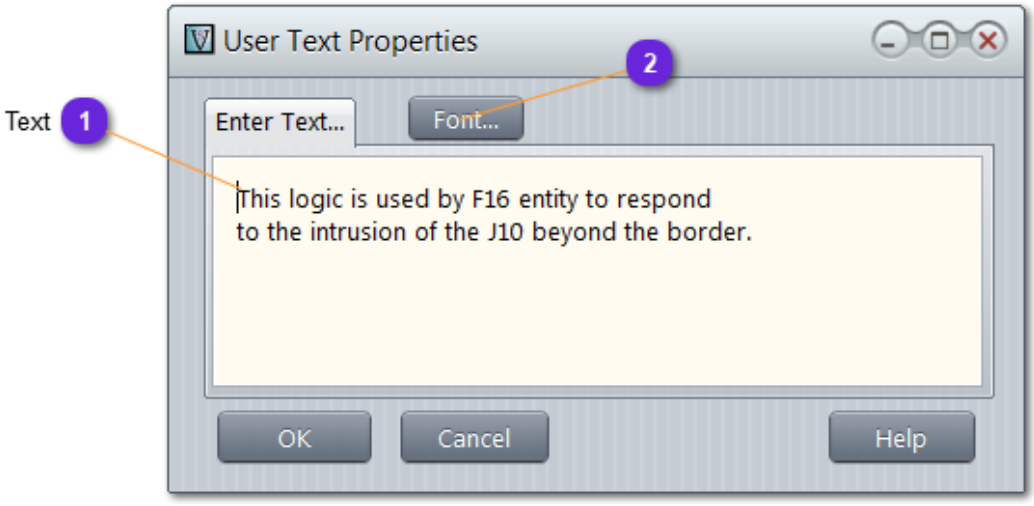

Put here whatever description of information useful to the designer to understand what the Watcher is about.
The help part is also used in the hint and the automatic document generation (see Make Documents).

# User Text

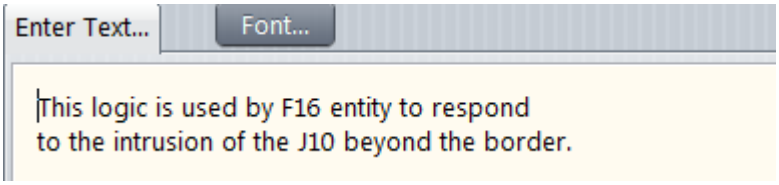
Use this object to enter any text you might find useful to annotate the Logic or to document parts of any diagrams.
This object is for the design only; it will not code generate anything.



# Properties

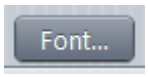



## 1 Text

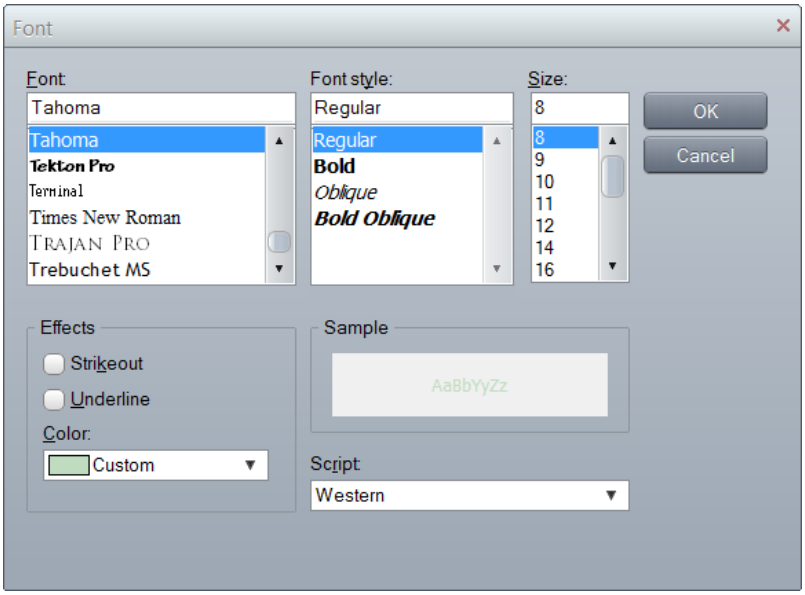



Enter here the text that will be displayed on the Diagram background.
The text area does not do word wrap so it is necessary to use carriage return to size the text box.

## 2 Font

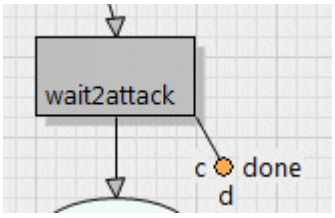



Use this button to set the font and the color of the text bloc.
The setting is for the whole text only, it does not apply to the selected part.



# Group

A **Group** is a Logic object that gathers and hide a group of Logic objects defining something specific. A Group is a kind of encapsulation of a specific process or complex logic. A Group is a kind of abstraction that can be manipulated as a new named object that contains own variables and methods.




A Group can be seen as a sub Logic in a sense that, once activated, its content is running independently. It simplifies the drawing and its comprehension.
Several groups can run concurrently inside a same logic.


*A Group is represented like a Task with a shadow. The Task is cycling a runtime code and a Group is cycling a bench of objects.*

- To **create** a Group, select all objects that must belong to the group and click the  button. All selected object should disappear from the Diagram and be replaced by the Group symbol (see above). If the button seems not to work, it is because conditions for making a Group out of the selected objects are not met. Selected objects must not have Connectors or Transitions going outside of the selected set.

- To **see** the **content** of a Group, double click the Group. To go back one level up, double click the diagram background (if inside a Group) of use the  button.

- To **ungroup** a Group and release all its content, select the Group symbol and use the  button. All objects of the Group will be dropped down to the upper group or the base.

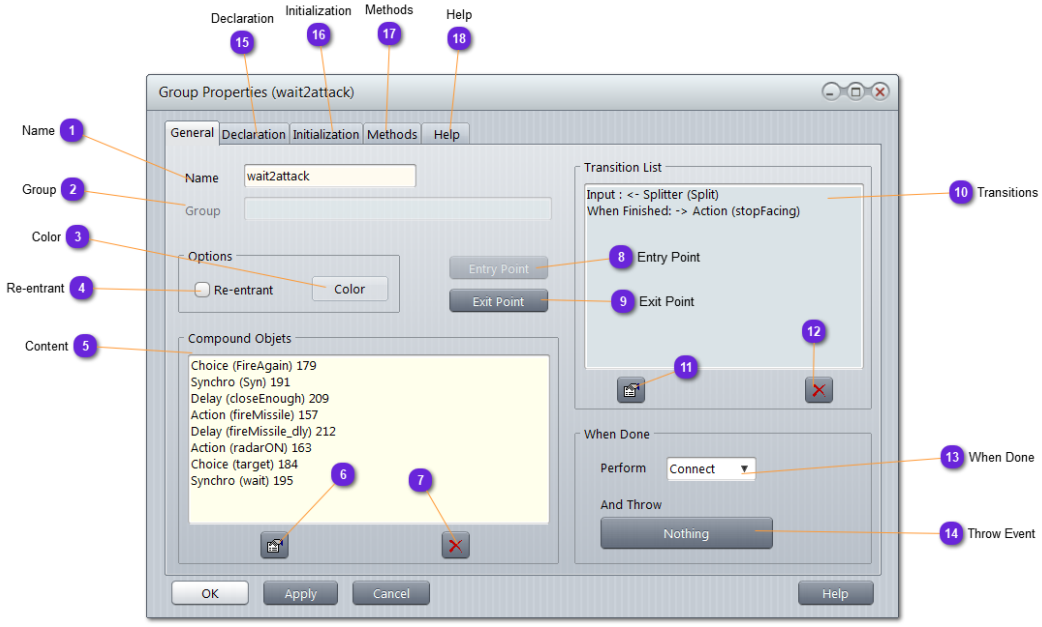
## • Code Hints

`scen()` or `S:` is a macro that returns a pointer to the Scenario instance that holds the Entity instance that holds the Logic instance that holds the current Group.
`ent()` or `E:` is a macro that returns a pointer to the Entity instance that holds the Logic instance that holds the current Group.

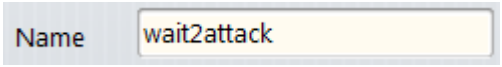
### Group

`logk()` or `L:` is a macro that returns a pointer to the Logic instance that holds the current Group.
`group()` or `G:` is a macro that returns a pointer to the current Group, if any, that holds the current Group. NULL if none.

# Properties




## 1 Name

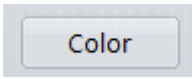



Name of the Group inside the logic.
This name can be used to retrieve the object (instance) in case user wants to activate it from source code.

## 2 Group

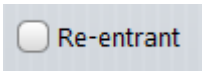

If the Group is part of another Group, the parent will be named here.
A Group name is the concatenation of all nested Group names: group1::group2::..::group3 (here, group3 being the deepest one).

## 3 Color

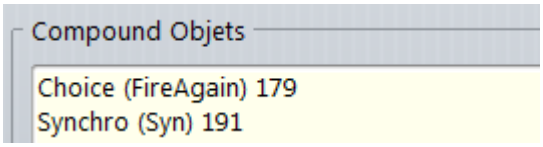



Set here the background color of the Group symbol.
You might also personalize the default color to highlight some specific Group or associate color with functionality or priority.

## 4 Re-entrant




## 5 Content




List all objects included into this Group.

## 6 Properties

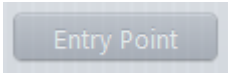
Display the property window of the selected object in the list (5).

## 7 Delete

Delete the selected object of the list (5)

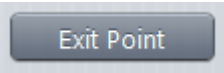

*Cannot undo*.

## 8 Entry Point




Use this button to access the Entry Point window (can also be opened from the diagram using double click on the Entry Point symbol).

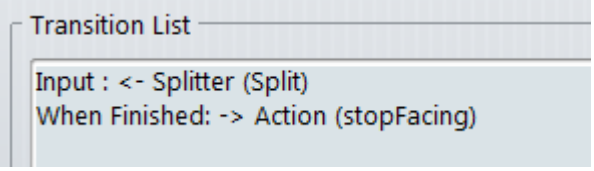
## 9 Exit Point




Use this button to access the Exit Point window (can also be opened from the diagram using double click on the Exit Point symbol).

*If grayed out, that means the Point is not defined.*

## 10 Transitions




List here all the objects connected to the current Group (in and out).

## 11 Properties

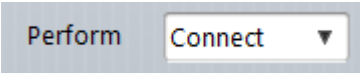
Display the property window of the selected object in the list (10).

## 12 Delete

Delete the selected object of the list (10)

*Cannot undo.*

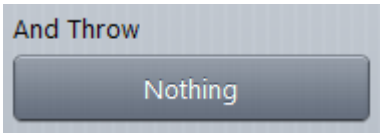
## 13 When Done




Inform about the kind of connection set for the Group after completion (meaning, when `return QUIT;` is encountered in one of its objects, or when one connector quits).

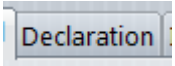
- Connect: the Group is connected to another object of the Logic
- Done: once the Group is finished, the logical flux is done for this one. The Group will not trigger any other object
- Quit: the Logic is left (and returns to the Behavior level) or the upper Group is left (if the Group is embedded into another Group)
- Nothing: Similar as Done but no event will be triggered

## 14 Throw Event



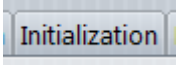

When the Group is over (and if Perform mode is not Nothing), an event can be raised.
Click here to set (or unset) the event.

## 15 Declaration




Add here all the variables and methods needed into the Group.
They will not generate interface so, needless to use `//&&` here.

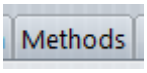
## 16 Initialization




This part will be used at different phases of the simulation.
Use the `RESET` part to initialize your variables.

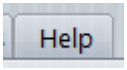

*Refer to the Developer Guide for more details on system phases/events.*

## 17 Methods




Put here the methods defined in the Declaration part (if any). Leave it empty otherwise.
Methods are accessible using `G:` macro from the code of any Group object.

## 18 Help



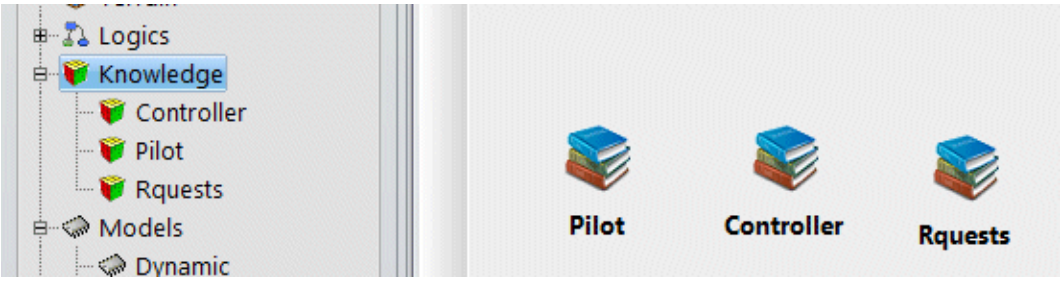

Put here whatever description of information useful to the designer to understand what the Group is about.
The help part is also used in the hint and the automatic document generation (see Make Documents).

# Knowledge

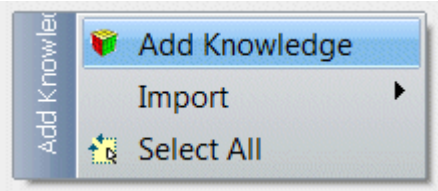
Knowledge is a set of Contexts processing a base of Facts and Rules (proposals) like a simple expert system would do with an inference engine from zero order.
The difference between a Logic and a Knowledge is that the first one is procedural while the second is declarative.
Knowledge might not be appropriate for all situations. It is best when decisions must be made according to well known rules or conditions which can occur at any time outside of a predefined sequence of actions or states.
knowledge can be used to trigger some Logics and stop some others.
It is a good approach to let Knowledge plays with Facts and Logic uses Events.


*Refer to the Tutorial to learn how to create your own.*

## • Environment




## • Popup Menu




**Add Knowledge**: create a new Knowledge in the database. Can also use ✚ from the toolbar.

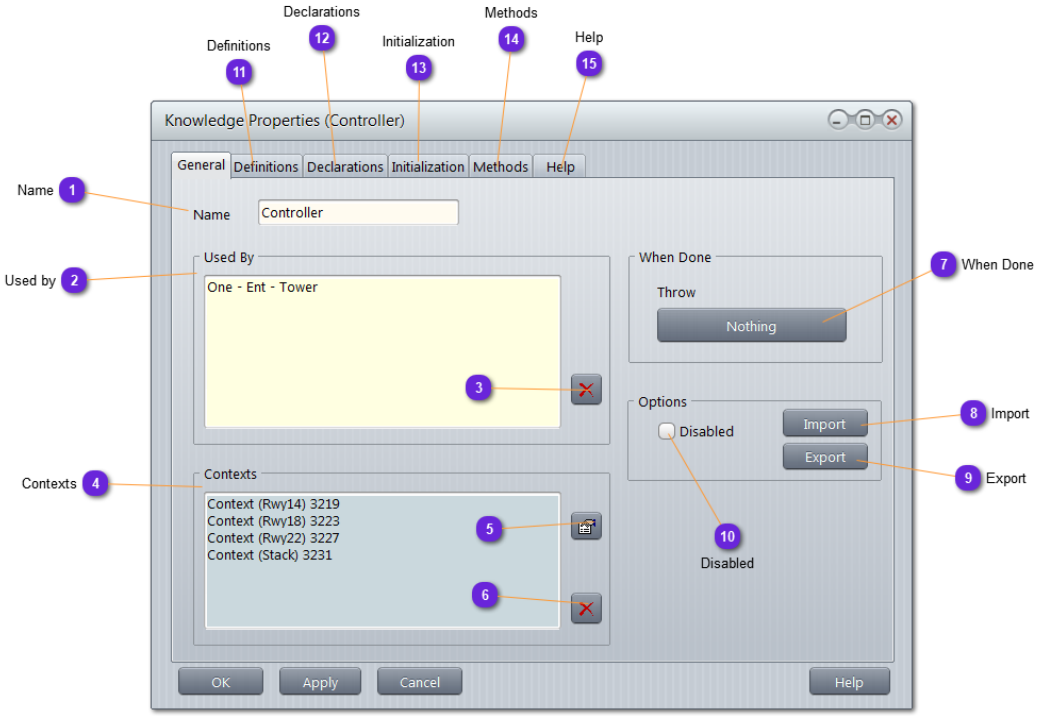
**Import**: list all exported Knowledge available in /Data/Shared directory

**Select All**: select all Logics of the database (useful for moving symbols or deleting them).

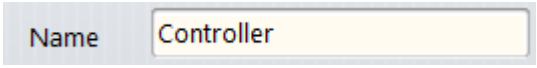
## • Code Hints

`scen()` or `S:` is a macro that returns a pointer to the Scenario instance that holds the Entity instance that holds the Logic instance that holds the current Choice.
`ent()` or `E:` is a macro that returns a pointer to the Entity instance that holds the Logic instance that holds the current Choice.
`know()` or `K:` is a macro that returns a pointer to the current Knowledge instance.

# Properties

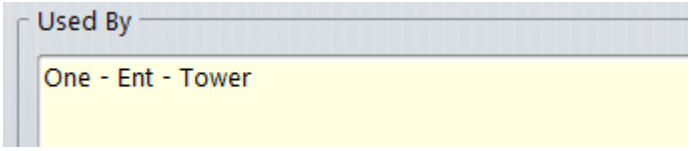



## 1 Name




Name of the Knowledge. Must be unique.
The class generated will be `Controller_Knw`.

## 2 Used by



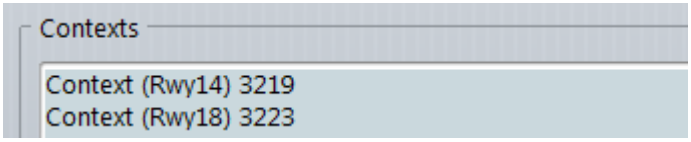

List all the entities that use this Knowledge in their behavior.
For each line (A - B - C):
A: name of the scenario (*above: One*)
B: type of the user: Ent for Entity, Cat for Catalog and Ply for the Player.
C: name of the entity (*above: Tower*)


*There will be one instance of a Knowledge (including compound Contexts) for each entity that uses it in its behavior.*

## 3 Delete

**Suppress** the **reference** to this Knowledge for the selected entity in the list (2).

## 4 Contexts




List all the Contexts contained in the Knowledge.

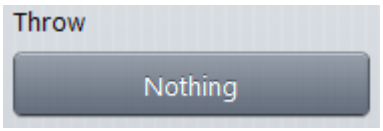
## 5 Properties

Call the property window of the selected Context of the list (4).

## 6 Delete

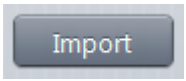
Remove the selected Context of the list (4).

## 7 When Done




Specify here the Event or Fact that will be triggered when the Knowledge is exited (optional).

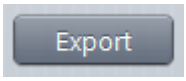
## 8 Import




Use this button to import a previously exported Knowledge in /Data/Shared.


*An exported Knowledge has extension .knw*

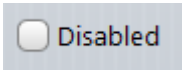
## 9 Export




Use this button to export a Knowledge for reuse or share between databases or users.
The Knowledge is exported by default in /Data/Shared but can be saved in any directory or support.

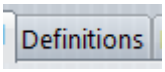

*An exported Knowledge has extension .knw*

## 10 Disabled




When checked, the Knowledge will not generate any code and cannot be enabled at runtime.

## 11 Definitions



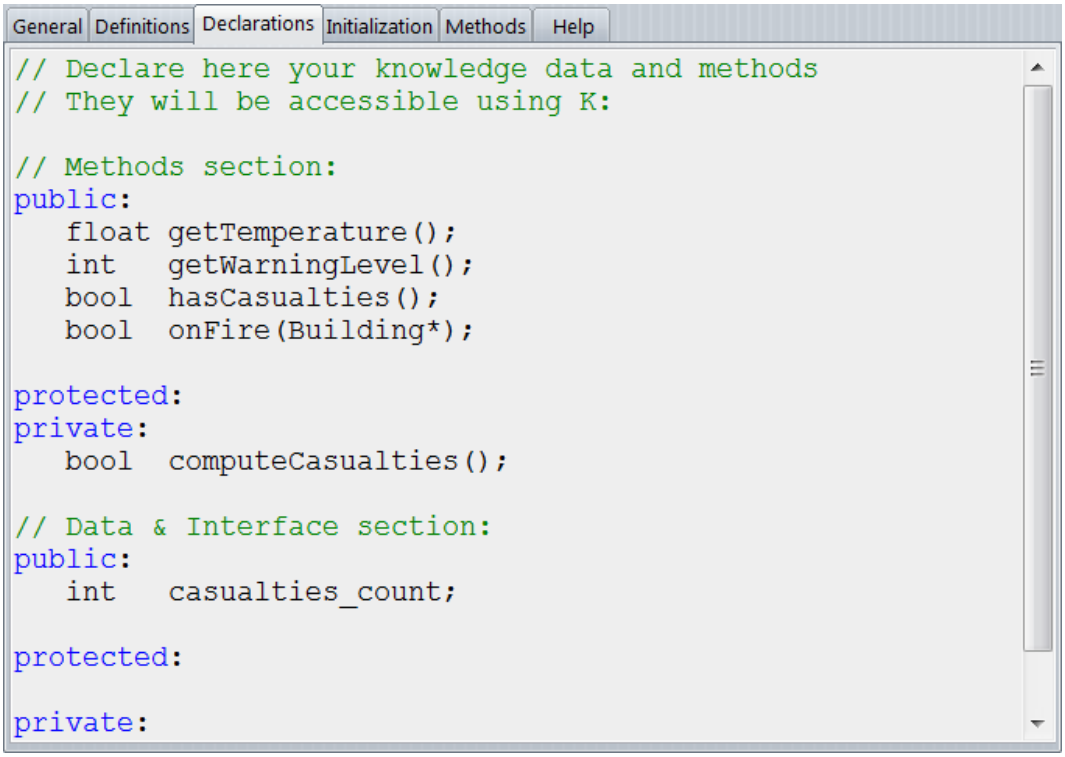

Put here all the includes and structures that might be needed by the code put inside the Context and Rules.
Definition part is inserted into the generated header file.

# 12 Declarations

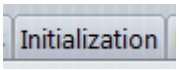
Declarations

A Knowledge is a class so, user can add special methods that will be used by Context and Rules.
Variables can also be declared here.
From a compound object code, using the `getTemperature()` method (see picture below) would be done this way: `K:getTemperature()`
Any public variable is accessible from the logic objects (i.e: `K:casualties_count`)
Private data/methods shall be reserved for Knowledge storage/process only.

User parameters `//&&` can be defined in a Knowledge and will generate an interface in the corresponding behavior symbol or the Entity. This can be very useful when defining a general purpose Knowledge that will rely on parameters that will change from one entity to one another.
For example, a Knowledge **Protect** will be using some parameters like speed, level, duration, etc. and each Entity using this Protect Knowledge will overwrite default values of these parameters.

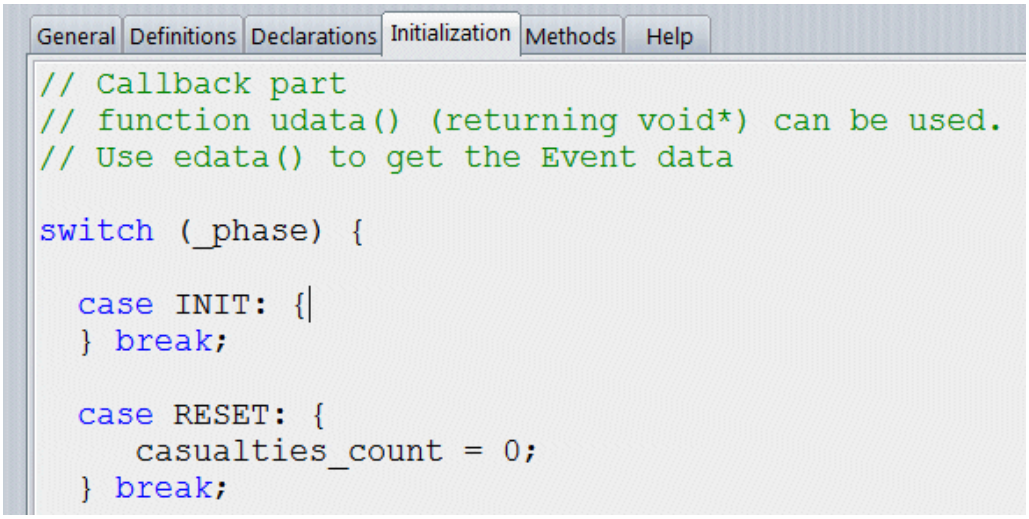


# 13 Initialization




Put here the code that is needed at every phase of the Knowledge. Used mainly to initialize the internal data and parameters used by the compound objects.
The initialization function is called for each instance of the Knowledge.

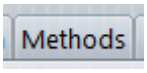
`INIT`: called once at creation time, for each entity that uses this Knowledge.
`RESET`: called several times, each time the Knowledge is activated.



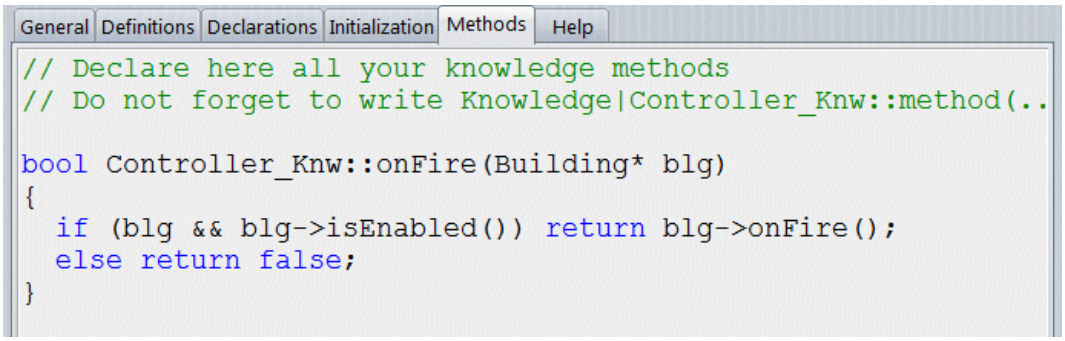


*Refer to the Developer Guide for more details on system phases/events.*

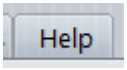
## 14 Methods




Put here the methods defined in the Declaration part (if any). Leave it empty otherwise.



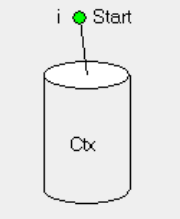

## 15 Help




Any description of the Knowledge can be put here and will be used for the automatic document generation (see Make Documents)

# Context

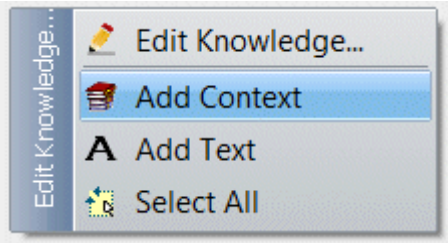
A **Context** is a Knowledge object that contains several user defined Rules.
It processes Rules in sequence and trigger them if their conditions succeed.
A Context expresses one semantic. For example, in a Crossing Knowledge, you might decide to give three Contexts: Pedestrians, Traffic Lights, Stop Sign. Each of these Contexts will harbor Rules relative to their own semantic. Each of them will them populate a Knowledge Fact database that could be used by a Logic to decide whether or not to cross the intersection.




Rules of a Context are processed from first to last.
When a Rule is triggered (`then` part or `else` part if valid), predefined conditions resets the Rule or not.
A Rule can only be triggered (fired) once until it is reset.

When a whole Context is reset, all embedded Rules are also reset and are ready to be fired again.

## • Popup Menu




**Edit Knowledge**: Call the Knowledge property window.
**Add Context**: To **add contexts** into the Knowledge. You can also use  on the toolbar, then, click anywhere on the Drawing Area (Diagram).

**Add Text**: Add a Text for comment.

**Select All**: select all Contexts of the Knowledge (useful for moving symbols or deleting them).

## • ToolBar

**Entry Point** (EPoint, green head) and **Exit Point** (XPoint, red head) cannot be added alone. They must be attached to a Context. If a Context is already selected, clicking the EPoint or XPoint button will immediately create and attach the point to it. Otherwise, cursor will change until a Context is designated into the diagram for dropping the point.

For EPoint or XPoint property windows, see here.

Add a Text on the diagram for comment.

*A Context must have an Entry Point in order to be evaluated. Without an EPoint, the Context stays idle (but can be started from the code by a Logic or a Rule).*

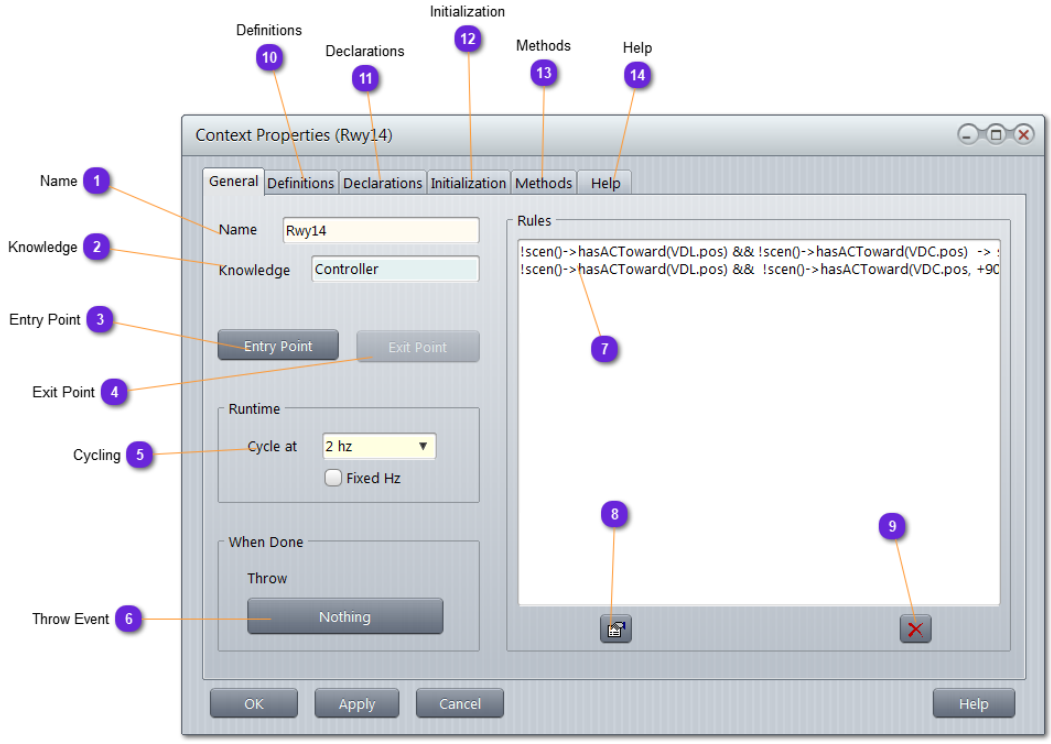
## • Code Hints

**`scen()`** or **`S:`** is a macro that returns a pointer to the Scenario instance that holds the Entity instance that holds the Knowledge instance that holds the current Context.

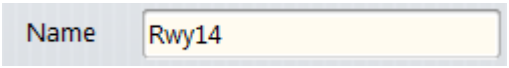
**`ent()`** or **`E:`** is a macro that returns a pointer to the Entity instance that holds the Knowledge instance that holds the current Context.

**`know()`** or **`K:`** is a macro that returns a pointer to the Knowledge instance that holds the current Context.

# Properties

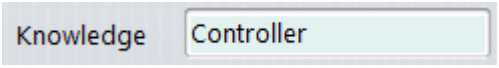



## 1 Name



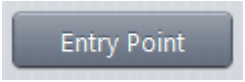

Name of the Context inside the Knowledge.
This name can be used to retrieve the Context (instance) in case user wants to stop the processing or modify some parameters from source code.

## 2 Knowledge



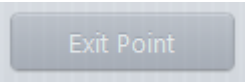

Name of the Knowledge the Context belongs to. Read only value.

## 3 Entry Point




Use this button to access the Entry Point window (can also be opened from the diagram using double click on the Entry Point symbol).

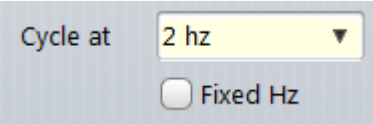
## 4 Exit Point




Use this button to access the Exit Point window (can also be opened from the diagram using double click on the Exit Point symbol).

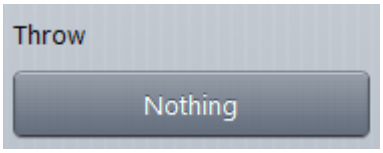
*If grayed out, that means the Point is not defined.*

## 5 Cycling



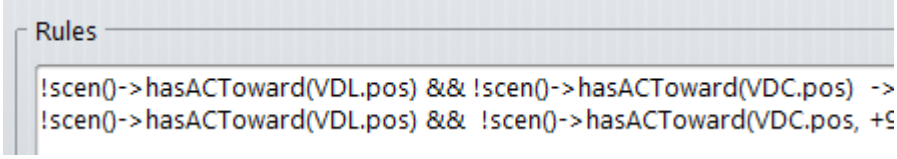

Check this to enable the runtime cycling of the Context (meaning, the Rules part).
The frequency list provided is according to the RTC base definition (a Context cannot cycle faster than the RTC).
If you need a user defined frequency not listed, take the highest frequency and add your own frequency divider code.

## 6 Throw Event




When the Context is over (and if Perform mode is not Nothing), an event can be raised.
Click here to set (or unset) the event.

## 7 Rules




List of all the rules defined in the context.
Click on each of them to edit them.

## 8 Properties

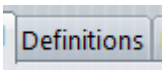
Display the property window of the selected Rule in the list (7).

## 9 Delete

Delete the selected Rule of the list (7).

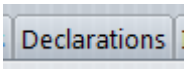
*Cannot undo.*

## 10 Definitions



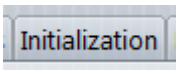

Put here all the includes and structures that might be needed by the code of the Rules.
Definition part is put in the generated header file.

## 11 Declarations




Add here all the variables needed into the Context to be used by the Rules.
They will not generate interface so, needless to use `//&&` here.

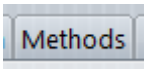
## 12 Initialization




This part will be used at different phases of the simulation.
Use the RESET part to initialize your variables.

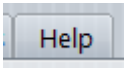
*Refer to the Developer Guide for more details on system phases/events.*

## 13 Methods




Put here the methods defined in the Declaration part (if any). Leave it empty otherwise.

## 14 Help



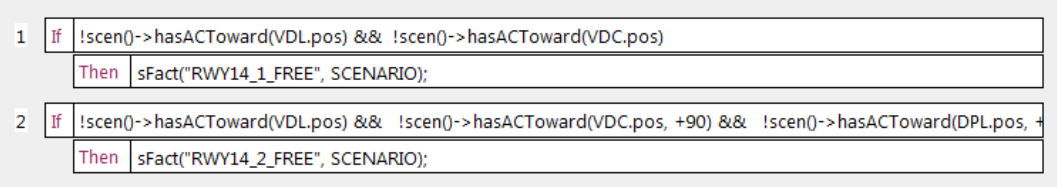

Any description of the Context can be put here and will be used for the automatic document generation (see Make Documents)

# Rule

A **Rule** is a Context piece of code that is called until the rule is fired (triggered).
A Rule is considered fired on two conditions only:

- The *If...* part evaluates as True and the *Then...* part is defined (not empty).
- The *If...* part evaluates as False and the *Else...* part is defined (not empty).

Double click on a Context symbol to open it and display its content (Rules) on the Diagram panel.
Double click on the background to close it and return to the Context view.




A Rule can access directly all data and methods defined in the Context as well as all vsTASKER API and internal data.

## • Popup Menu

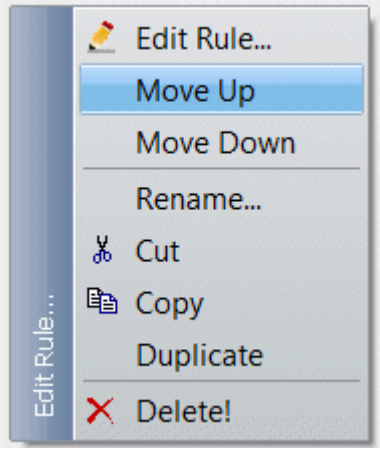
On the **background**:


**Edit Context**: call the Context property window.

**Add Rule**: append a new Rule in the list (at the end).

**Up**: close the View and return to the Context view (same as double clicking on the background)

**When** a Rule is **selected**:




**Edit Rule**: call the Rule property window.

**Move Up**: move up the selected Rule in the list, to be processed earlier.

**Move Down**: move down the selected Rule in the list, to be processed later.

**Rename**: *no effect, Rule has no name.*

**Cut**: allow moving one rule from one Context to one another. Copied and deleted locally.

**Copy**: copy the selected Rule into the clipboard.

**Duplicate**: copy and append the selected Rule.

**Delete**: remove the Rule from the list (cannot undo).

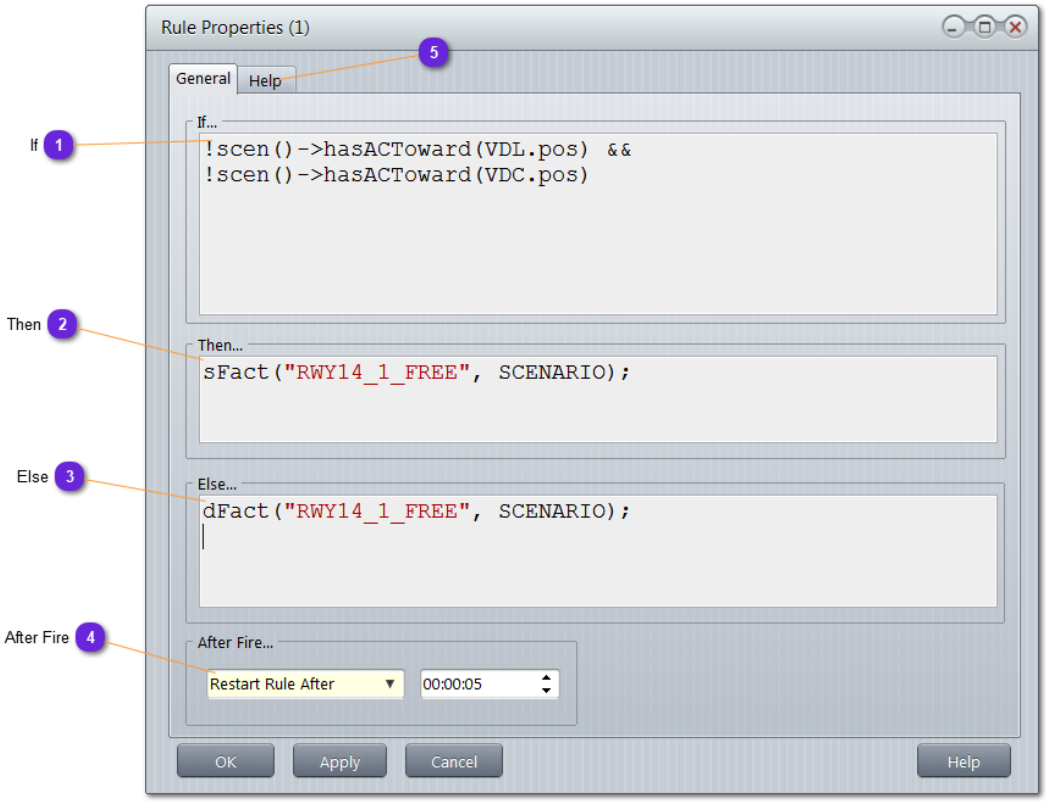
## • ToolBar

 To **add rules** into a Context, you must first open a Context by double clicking its symbol. Any click on this button appends a new Rule into the context.

## • Code Hints

`scen()` or `S:` is a macro that returns a pointer to the Scenario instance that holds the Entity instance that holds the Knowledge instance that holds the current Context.
`ent()` or `E:` is a macro that returns a pointer to the Entity instance that holds the Knowledge instance that holds the current Context.
`know()` or `K:` is a macro that returns a pointer to the Knowledge instance that holds the current Context.

# Properties




## 1 If

```
If...
!scen()->hasACToward(VDL.pos) &&
!scen()->hasACToward(VDC.pos)
```

Contains the condition to trigger the Rule.
This condition is a C/C++ expression that is `true` or `false`. It is not a function, it is an expression.

## 2 Then

```
Then...
sFact("RWY14_1_FREE", SCENARIO);
```

Contains the code that is run if the condition (1) of the rule is `true`.
The user can put whatever C/C++ code he wants there. This code can access all vsTASKER API, all included third-party APIs and all user designed scenario, entity, knowledge and context data of the current database.

## 3 Else

```
Else...
dFact("RWY14_1_FREE", SCENARIO);
```

Contains the code that is run if the condition (1) of the Rule is `false`.
The user can put whatever C/C++ code he wants there. This code can access all vsTASKER API, all included third-party APIs and all user designed scenario, entity, knowledge and context data of the current database.

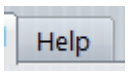
*This part is optional and can be left blank.*

## 4 After Fire


Selection box to decide what to do with the Rule once it triggers (evaluates to `true`):

- Rule is Done: Once triggered, the Rule is not revaluated again until the Context is reset.
- Reactivate Rule: Once triggered, the Rule is evaluated again at next Context cycle.
- Reactivate Rule After: Once triggered, the Rule is evaluated again after a time delay specified in the next field.
- Reactivate Context: Once triggered, the Context is reactivated, meaning, all fired rules are reactivated.
- Reactivate Context After: Once triggered, the Context is reactivated after a time delay specified in the next field.
- Exit Context: Once triggered, the Context is exited. If an Reaction has been set in the Context property window (On Exit), it will be triggered.
- Exit Knowledge: Once triggered, the Knowledge is exited. All Context are terminated. The hand returns to the Behavior.

## 5 Help




Any description of the Rule can be put here and will be used for the automatic document generation (see Make Documents)

# Models

Entities need Models to be able to physically interact, like a car needs an engine, or a bicycle needs a dynamic law not to fall.
All this and so much more can be provided by Models.

vsTASKER provides some predefined Models for general topics like dynamics, sensors and weapons.
User can easily add their own and expand without limit the built in Model list.

Once a Model is defined and included into a database, it can be attached to any entity like an engine can be installed into a car to make it run.
Changing one Model with another one can thus be done with one mouse click. Code will be automatically generated, reducing the programming effort making the process user friendly.

*Refer to the Developer Guide to learn more on built-in models. The Tutorial will also explain how to create your own.*

## • Type of Models

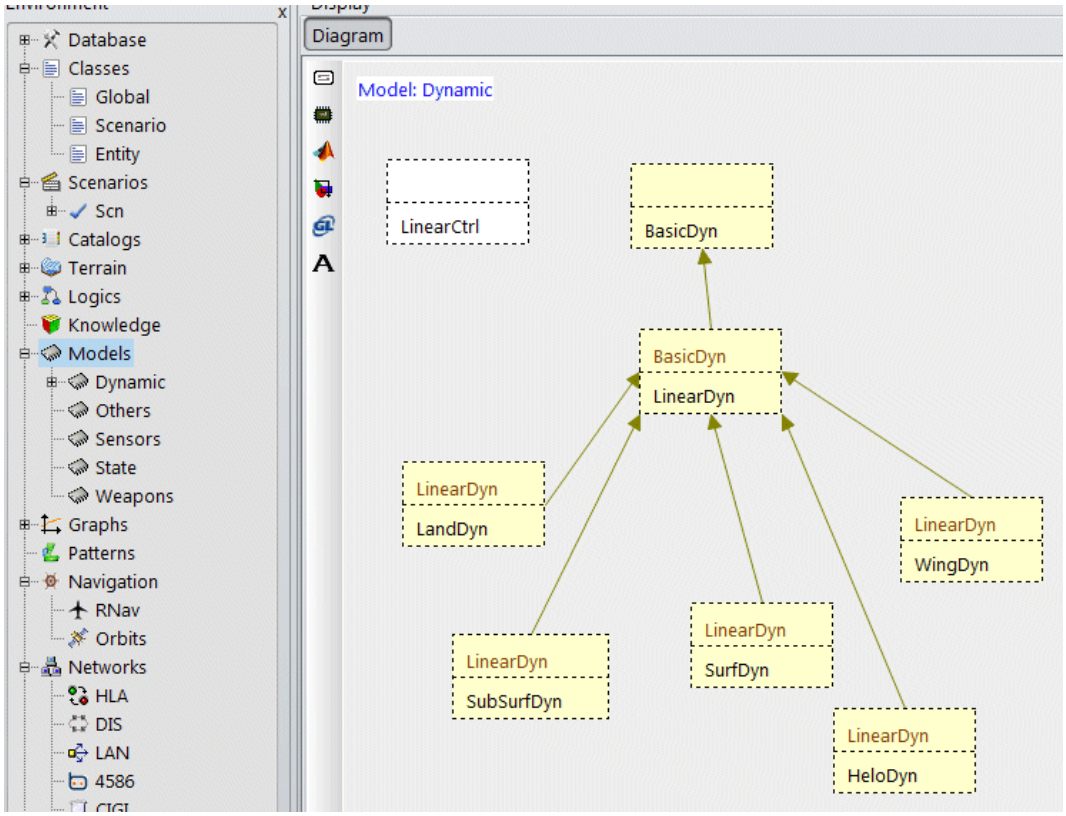
DataModel: simple data structure.
Component: a DataModel with a runtime user code and reactive to events.
Matlab: a special component for handling Matlab™ function.
Simulink: a special component for handling Simulink™ model.
GL-Studio: a special component for handling GL-Studio™ graphic object.

## • Containers

Models are gathered in various Containers.

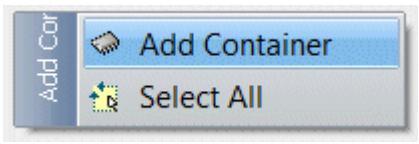

A container is merely a folder that isolate its content from other containers.
Nevertheless, each model generates a class with its name. So, even if inside a container, two models cannot have the same name.
Also, a model name cannot be a C++ keyword.

*For now, a container cannot embed another container.*




## • Popup Menus

From **Model View**

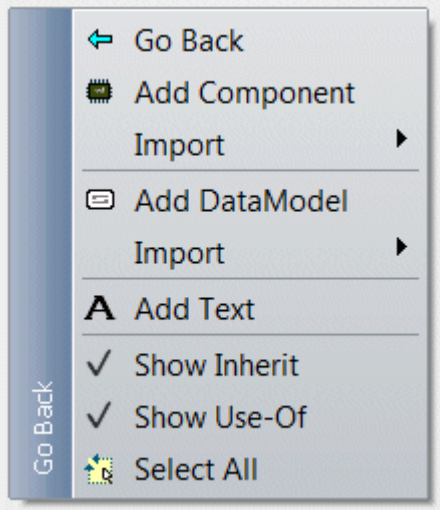



**Add Container**: create a new empty container.

**Select All**: select all containers (for moving or deletion).

From inside a **Container**

### Models




**Go Back**: close the actual container and return to the Model view.

**Add Component**: Import Components from the library. Go here for more info on the importer.

**Import**: list all Components defined in the /shared/models directory.

**Add Data-Model**: create a new empty Data-Model.

**Import**: list all Data-Models defined in the /shared/models directory.

**Add Text**: create a text block for comments.

**Show Inherit**: if checked, all Model inheritance links (to models belonging to the same Container) will be shown with a green arrow.

**Show Use-Of**: if checked, all Model usage links (to models belonging to the same Container) will be shown with a dashed arrow.

**Select All**: select all models of the current Container.

## • ToolBar

create a new empty Data-Model.

create a new empty Component.

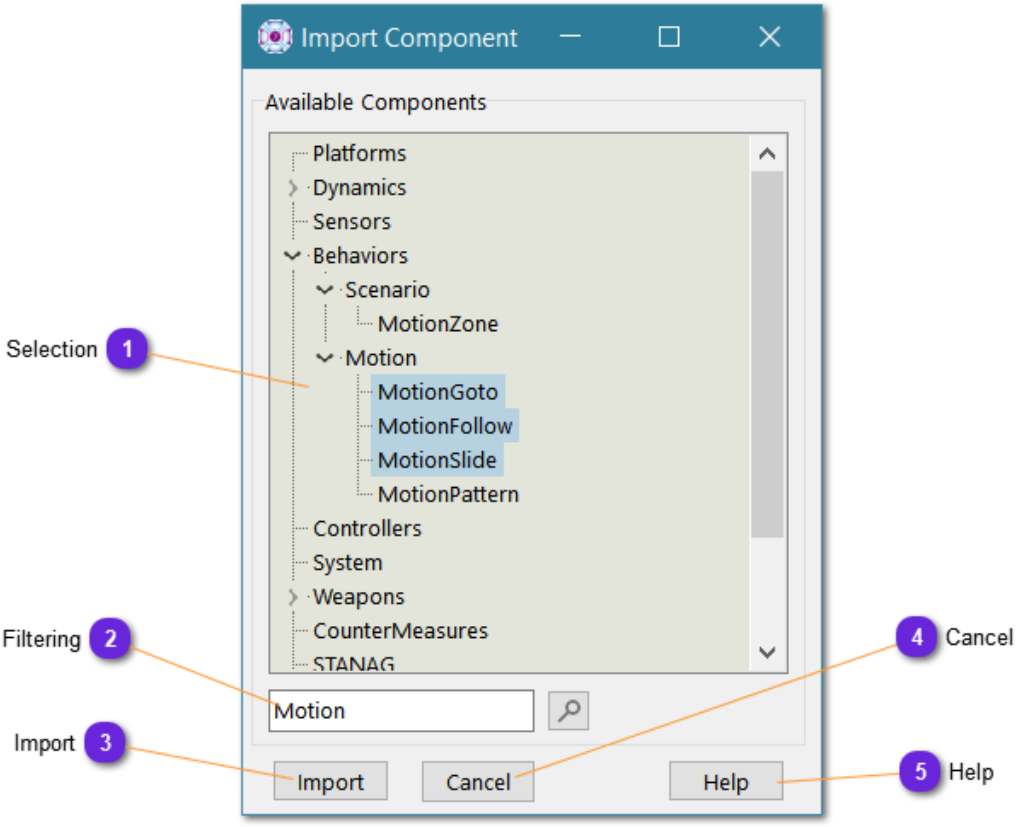
create a Matlab™ function model.

create a Simulink™ handler model.

create a GL-Studio™ wrapper model.

create a text block for comments.

# Model Importer




## 1 Selection

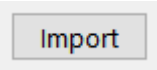

Components or data-models are listed according to their definitions in the corresponding /Model/*.cpt or /Model/*.dml files.
Use the control key to select many.

*The **multiple selection** is a new feature of v7 to simplify a tedious import process of the previous versions.*

## 2 Filtering

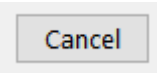

Type here the component or data-model name you are looking for, then press the button. The tree-list will expand the branches containing the searched items.

## 3 Import



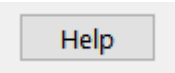

Import into the database the selected components or data-models, only if not already there.

## 4 Cancel




To quit without importing anything.

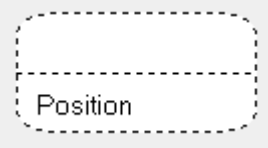
## 5 Help




Display this window

# Data Model

A Data-Model is a **C++ Class** that can be used by any other objects of vsTASKER (Scenario, Entity, Logic, Knowledge, Component...).
Typically, a DataModel is used to store data. It can inherit from another DataModel.

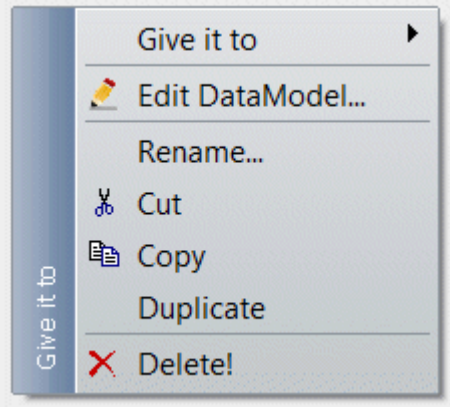



Data-Models are attached to entities on the Model Behavior panel. Several are built-in, like `PtfModel` that contains a list of variables and methods to characterize the entity.

*A DataModel is shown dashed when external (defined in a library) and in plain line when local.*

## • Popup Menu

**When selected:**




**Give it to**: attach the Data Model to the entity being selected in the popup list.

**Edit Data Model**: call the property window.

**Rename**: change the name of the Data Model.

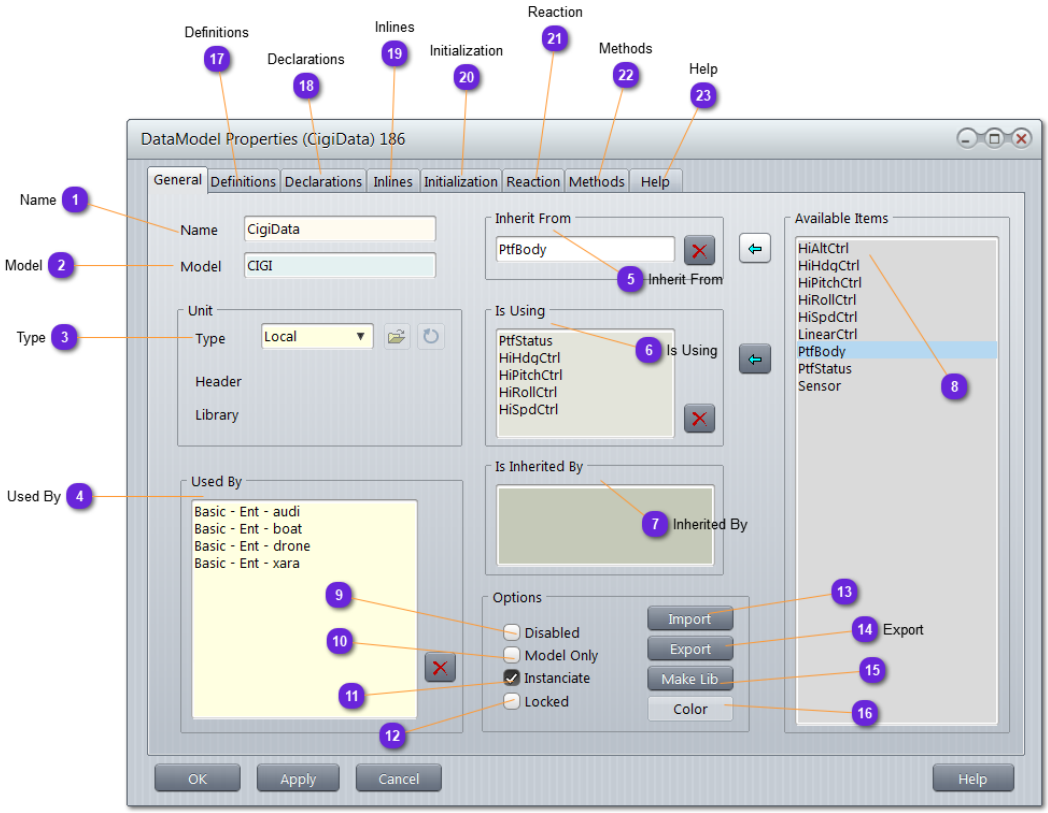
**Cut**: copy it into the clipboard and remove it from the Container.

**Copy**: copy it into the clipboard.

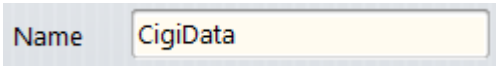
**Duplicate**: create a clone of the Data Model into the same Container.

**Delete**: ask confirmation and delete the Data Model. There is no undo.

# Properties




## 1 Name

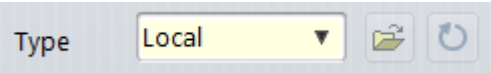



Name of the Data Model. Must be unique.
The class generated will be `CigiData`.

## 2 Model


Name of the Container the Data Model belongs.

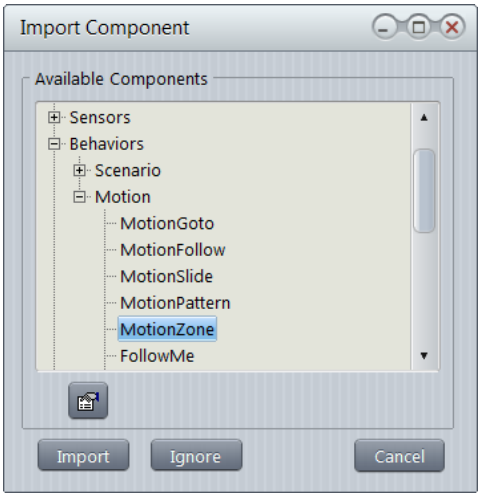
## 3 Type




Choice between:

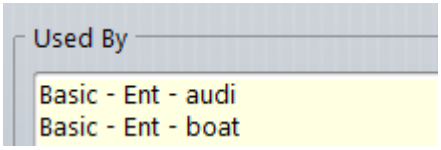
- Local: the Data Model is defined locally in that database only. It can be exported, copied, duplicated and changed. Source code is available.
- External: the Data Model is available in a library. Only the interface (header file) is available in read-only. User can only changes default values.

When External, the object can be specified or changed using button:




From the selection window, header file and library (including the Data Model) will be listed (read only).
These information are part of the Model Definition files. See here for more info.

## 4 Used By




List all the entities in different scenarios, using this Data Model.
Format: Scenario - Type (Entity, Player or Catalog) - Name

## 5 Inherit From

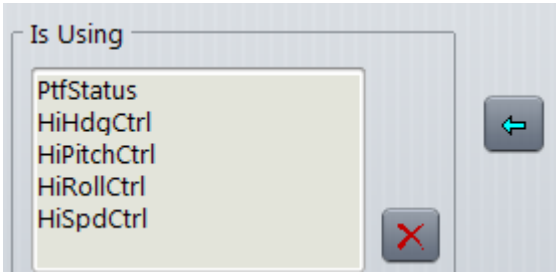

Specify the Data Model (external or local) the current one inherit from, to specialize it.
Plain arrow is use to translate this inheritance.

Select the Data Model in the Available Items list (8) then use the button to set.

Use the button to remove the inheritance.

*Inheritance is translated into C++ inheritance as the Data Model is itself a C++ class. This inheritance feature must be understood this way.*

## 6 Is Using



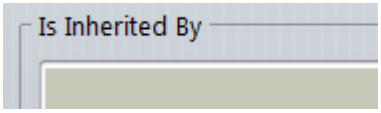

Specify all the Data Models (externals or locals) the current one are using internally as variables.
This setting is used for the code generation to create correct makefile and add necessary `#include` into the header file.
Dashed arrows are use to translate this usage.

Select the Data Model in the Available Items list (8) then use the button to add.

Select the Data Model in the list and use the button to remove.

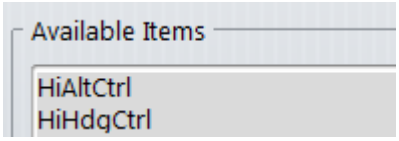
*This is important for declaration resolution conflicts at compile time. vsTASKER automatically lists Data Models in the proper order.*

## 7 Inherited By



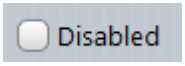

List all the Data Models (or Components) that inherit from the current one.

## 8 Available Items



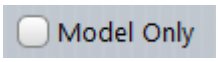

List all the Data Models loaded into the database (externals or locals).

## 9 Disabled



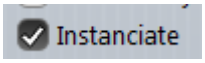

Exclude the Data Model from the code generation.
Disabled Data Model will not be included into the items using it (entities but not only) without the need to remove it from all users.
Disabling is error prone while removing might leave dangled pointers.

## 10 Model Only



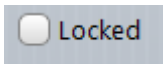

When checked, the Data Model will not be available into the entity model list.

## 11 Instanciate



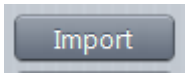

If checked, the Data Model class, when attached to an owner (Entity, Player, etc.) will create an object with `new`.
If unchecked, the attachment will be seen as a reference without object (`NULL` pointer). This can be useful when different Entities (for example) must share a same and unique Data Model. It is then the responsibility of the user code (wherever it is) to create the Data Model, reference it whenever it is needed and delete it at simulation stop.

## 12 Locked



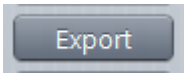

When checked, the Data Model cannot be modified. All changes will be lost at Ok or Apply.

## 13 Import



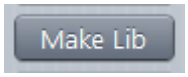

Use this button to replace the current Data Model with a previously exported Data Model in /Data/Shared/Models.
All current content will be lost.

## 14 Export



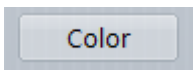

Use this button to save the current Data Model for later import. Default location is /Data/Shared/Models.

## 15 Make Lib



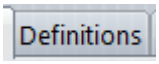

Use this button to automatically translate the current local Data Model into a C++ class with a header and a source file, in order to include it into a dedicated library.
The Data Model will then become External.
See here for more information.

## 16 Color



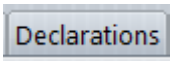

Specify the color to be used for drawing the Data Model symbol into the Diagram panel.
Default is white.

## 17 Definitions



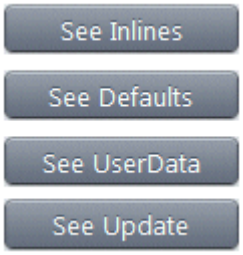

Put in this panel all `#include` (third-party API), `#define` and `typedef` needed for the current Data Model.
This content is put as is, without any preprocessing, at the beginning of the database header file and outside of any class definition.

## 18 Declarations



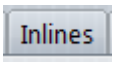

Put in this panel all **methods and data** you want to add in the current Data Model.

**Interface**: User can define here interface variables. See Dyn-UI for all Options.



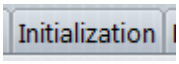

**See Inlines**: show the automatic inlines as they will be generated for the Dyn-UI variables.
**See Defaults**: show the `setDefault()` function as it will be code generated from the Dyn-UI `DEF[]` definitions.
**See User-Data**: show the `setUserData()` function (user defined interface values) as it will be generated, for the Dyn-UI variables.
**See Update**: show the update() function (sim to gui at runtime) as it will be generated, for the Dyn-UI variables.

## 19 Inlines

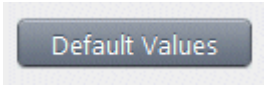



Put in this panel all **inline methods** of the current DataModel.
There is a preprocessing of the Inlines content, so do not forget the keyword `inline` and the word `Dml::` before any method name.

## 20 Initialization



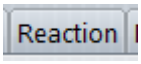

This part is called automatically at instantiation (INIT), when DataModel is created using *new*, or every time the DataModel is *reset* and finally at destruction (CLEAN) using *delete*.




**Default Values**: Pop up the variable Dyn-UI allowing the user to supersede or overwrite the default values (`DEF[]`) of the current Dyn-UI. These values will be set using the `setUserData()` function call with the default Dyn-UI then, the same function will be called with the DataModel owner Dyn-UI setting.


*It is a good practice to put in the `RESET` part whatever must be reinitialized at reset. There is no break after `INIT` to allow the `RESET` part to be processed after `INIT`, at Data Model creation time.*

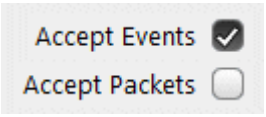
## 21 Reaction




This part is used to process `EVENT` or `PACKET` events when selected.

```
switch (_type) {

  case EVENT: {
     Event* event = (Event*) _mdata;
  } break;

  case PACKET: {
     Packet* packet = (Packet*) _mdata;
  } break;

}
```

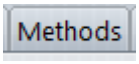
`case Event` received from `raiseEvent(...)` method in the code.
`case Packet` received from `send(...)` method in the code.




If **Accept Events** is checked, the `case..break` code block will be called whenever an Event is raised towards the Entity holding the Data Model.
If **Accept Packets** is checked, the `case..break` code block will be called whenever a Packet is received in a Channel used by the Entity holding the Data Model.

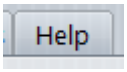

*Refer to the Developer Guide for more details on system phases/events.*

## 22 Methods




Put in this panel all **methods** of the current Data Model declared in the Declaration panel. There is a preprocessing of the Methods content, so do not forget the word `Dml::` before any method name.

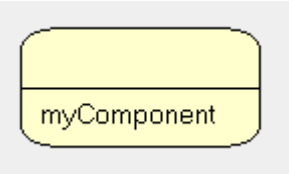
## 23 Help




Any description of the Data Model can be put here and will be used for the automatic document generation (see Make Documents)

# Component

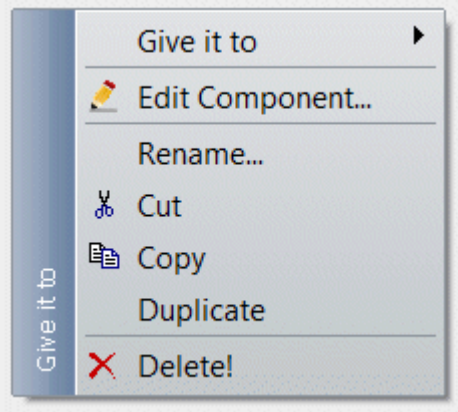
A **Component** (Model) can be seen as a Data-Model with a runtime routine (hearth beat) spinning at his own frequency.




*A Component is shown dashed when external (defined in a library) and in plain line when local.*

## • Popup Menu

**When selected:**




**Give it to**: attach the Component to the entity being selected in the popup list.

**Edit Component**: call the property window.

**Rename**: change the name of the Component.

**Cut**: copy it into the clipboard and remove it from the Container.

**Copy**: copy it into the clipboard.

**Duplicate**: create a clone of the Component into the same Container.

**Delete**: ask confirmation and delete the Component. There is no undo.

## • Property Window

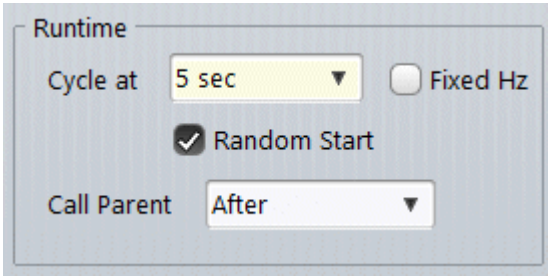
See this property window and replace Data Model with Component.
The only difference between Data Model and Component is the Runtime part.

*If the Component is manual, do not forget to set scenario and entity pointers in the RESET part of the Initialization panel.*

# Runtime Part

A Component is running at a specific frequency.
It must be set in the setting box.

## • Frequency Settings




**Cycle at**: select from the list the appropriate frequency for the Component, according to the RTC runtime frequency.

**Fixed Hz**: when checked, the selected frequency is not according to the RTC but according to the wall clock.
With a fixed frequency, even if the simulation engine is accelerated or slowed down, the number of calls per seconds will remain the same.

**Random Start**: when checked, the first call of the runtime tic() will not be at time 0 but randomized between 0 and the specified frequency cycle. For example, at 5 seconds cycle, a Random Start will make the component start cycling between 0 and 5 seconds after simulation starts or component activated. This will insure that if the same component is given to several entities, they will be distributed over the cycles and not all running under the same cycle.

**Parent**: Select the parent runtime call mode:

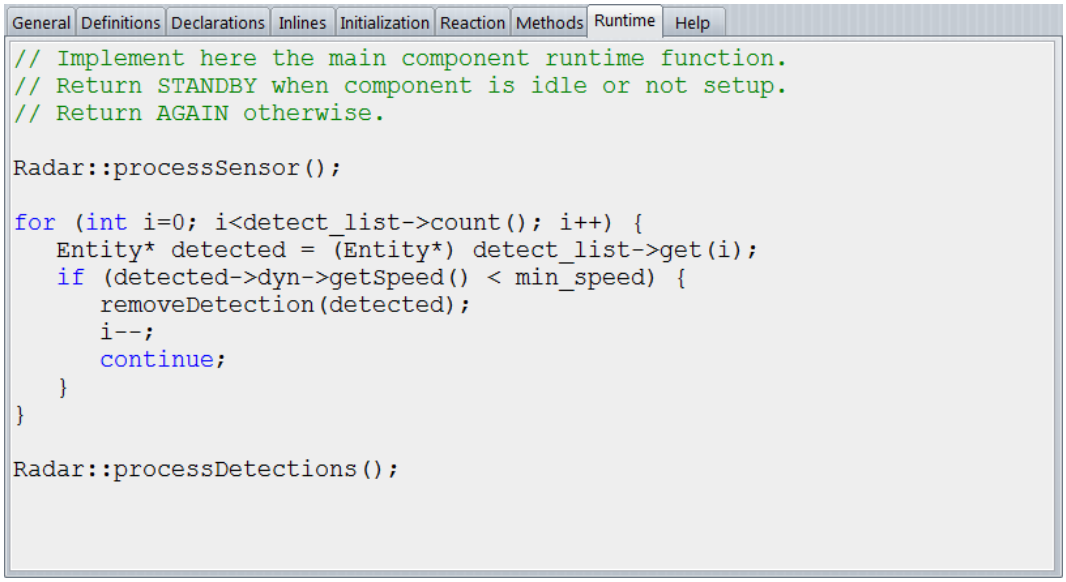
- Before: parent `tic()` called before the current Component one (useful for post processing)
- After: parent `tic()` called after the current Component one (useful for preprocessing)
- Manual: parent `tic()` must be called (or not) from the current Component `tic()` function.

## • Runtime Part

The Component runtime code is the tic() function.
This function is called at the requested frequency (see above).

### Runtime Part

When a Component `A` inherit from a Component `B`, `A::tic()` function might or must contain `B::tic()` call (in Manual mode, see above) or any other public methods of `B` (see below).




## • Code Hints

**`scen()`** or **`S:`** is a macro that returns a pointer to the Scenario instance.
**`ent()`** or **`E:`** is a macro that returns a pointer to the Entity instance that holds the Component.

# DLL

A Component can be local to the database, external to a library or compiled as a DLL and loaded by the simulation engine at runtime.

## • Header

Create the DLL header interface (here accel.h)

```
 typedef int (__stdcall* AccelFunc)(float speed, float acc, float&
result);
```

Specify it into the Definition panel of the DLL component

```
#include "../model/dll/accel.h"
```

## • Declaration

In the Declaration panel, create all the DLL function pointers as defined in the header as a `typedef`, any data that is needed for interfacing the Component with the simulation engine and user.

```
AccelFunc pAccel;
float acc; //&&  DEF[30]
HMODULE hDll;
```

## • Initialization

In the Initialization panel, the compiled DLL file must be loaded and its embedded function must be extracted and allocated the the Component function pointers. Once this is done at simulation load (`INIT`), the DLL functions can be used either in the Component methods or the runtime `tic()`.

```
  case INIT: {    // at first start
    hDll = LoadLibrary(strAdd(vsTaskerDir(),"/bin/accel.dll"));
    pAccel= (AccelFunc) GetProcAddress(hDll, "accel");
 } break;
```

## • Runtime

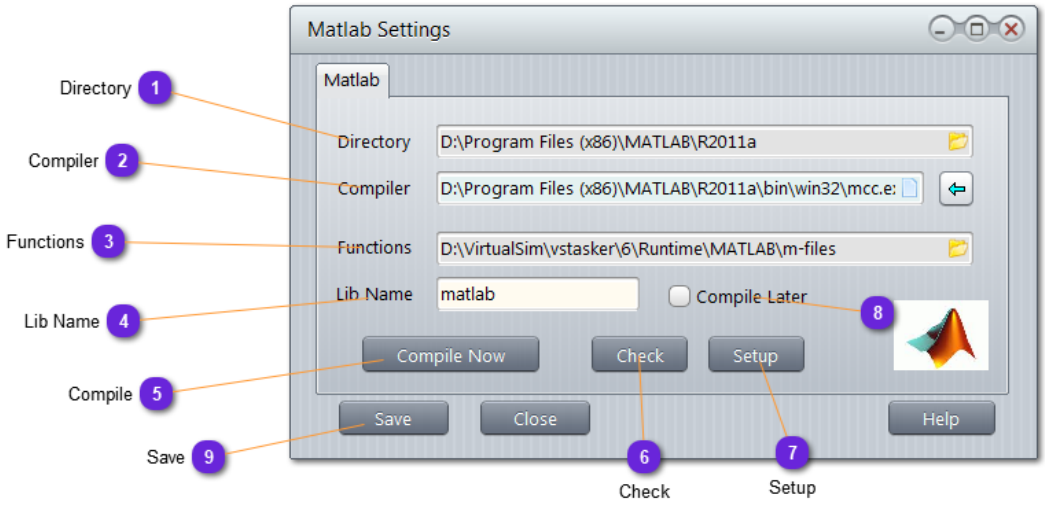
The DLL functions will be used according to their interface defined in the header. For i.e:

```
pAccel(current_spd, acceleration, new_speed);
```

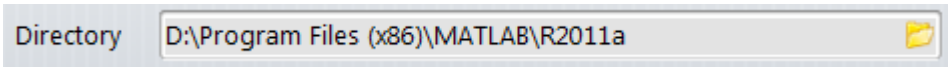
# Matlab Function

A Matlab function is a compiled m-file that can be called from any place in the vsTASKER code. vsTASKER creates a Matlab container called **m** with all m-file embedded functions defined in the database.
For ie: if a Matlab function `foo()` is compiled, then, it can be accessed from the C++ code with: `m.foo()`

# Settings

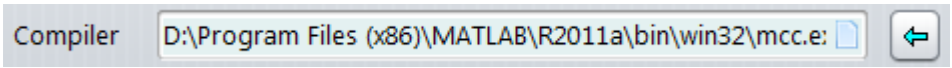



## 1 Directory



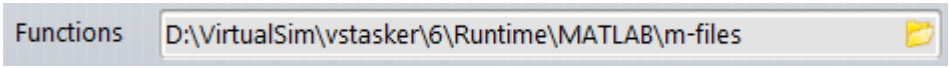

Select here the directory where MATLAB has been installed.
Use the little folder icon to select it.

## 2 Compiler



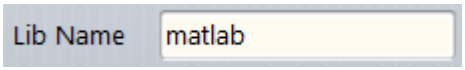

Location of the mcc.exe compiler that will be used to compile the MATLAB functions defined in the database to create the local library.
Use the blue left arrow button to automatically retrieve it once the **Directory** has been set.

## 3 Functions




Directory where all MATLAB function (*.m) will be defined and stored for vsTASKER database.
Use the little folder icon to select it.

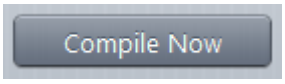
## 4 Lib Name




Name of the library that will be created with all the M function files compiled. This library will be stored in the **Functions** directory (in the snapshot above, it will be matlab.lib)

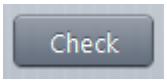

*Because the M files compilation is slow, it is more efficient to build a library and use it with the simulation engine.*

## 5 Compile




Force the build of the library by compiling all M functions of the database. Once the library is built, it will not be necessary to rebuild it every time the database is changed, unless the MATLAB functions are changed.

## 6 Check

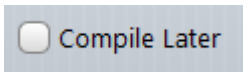



Use this button to check the compiler with a simple default M function file.

## 7 Setup

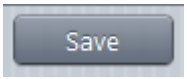

Use this button to select the compiler to use.
If you have several compiler installed, use the one that will be used to also compile the vsTASKER database to avoid mismatch with libraries.

## 8 Compile Later



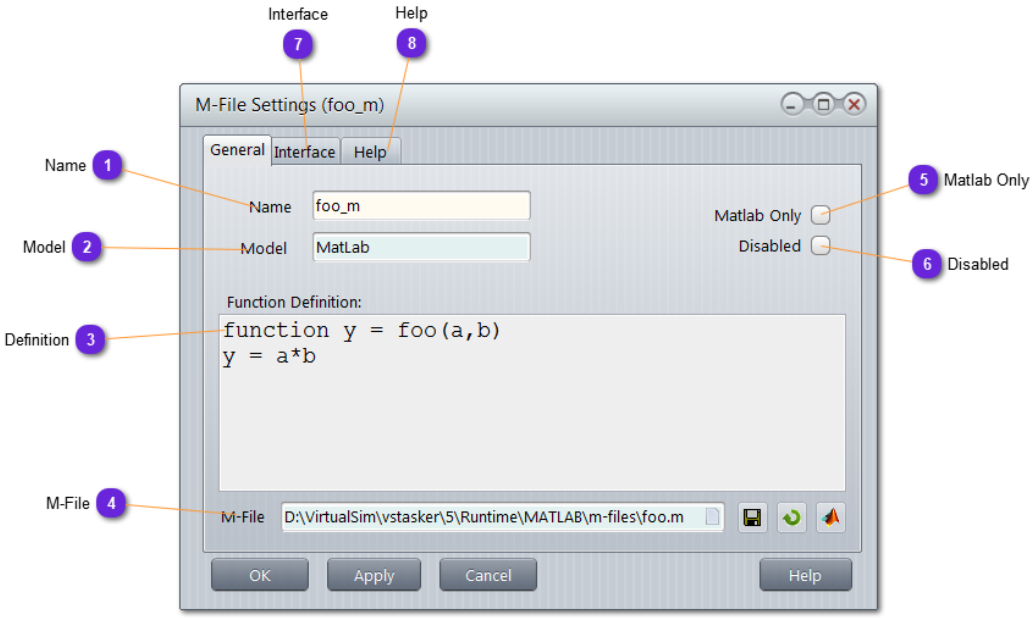

If checked, compilation of the M files will be done at database compilation time.

## 9 Save




Store and memorize the actual settings.

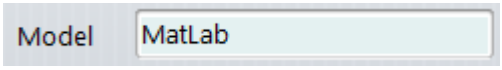
# Properties




## 1 Name

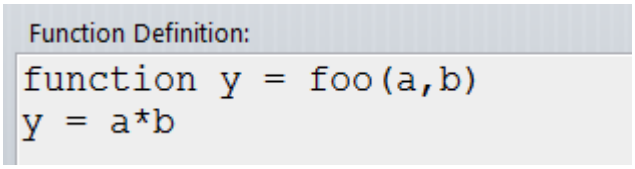

Name of the Matlab function. Must be unique.
The class generated will be `foo_m`.

## 2 Model



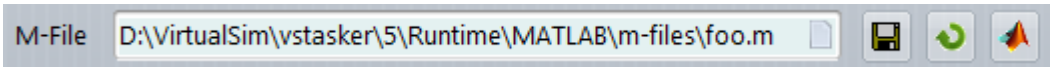

Name of the Container the Matlab function belongs.

## 3 Definition




You can write or modify here the Matlab function the same way as you would do it in Matlab, except that there will be no evaluation nor syntax check.

## 4 M-File



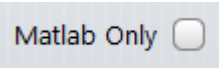

The Matlab function must be written into a file (with .m extension) in order to be compiled with the mcc.exe compiler.
See here for the setting of Matlab in vsTASKER.
Set here the full path and file name to the M-File. If not exist, create one on the appropriate directory (see Concept)

for saving the definition (3) into the M-File

for updating the definition (3) with the content of the M-File

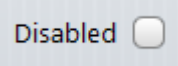
call the Matlab editor (MATLAB environment variable must be set).

## 5 Matlab Only



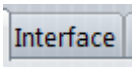

If checked, code is compiled but the function is not included into the `m` container.

## 6 Disabled

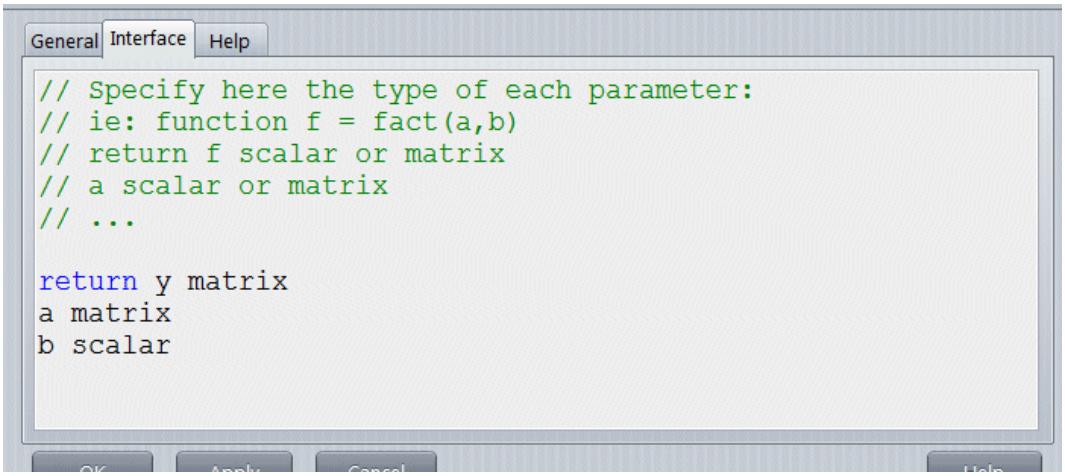



If checked, code is not compiled and function is not included into the `m` container.
Code using this function will no more compile so, use it with care.

## 7 Interface



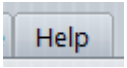

The Interface panel is for defining the parameter types of the function and the return type(s).
In the picture below, **y** variable will be of type **matrix**.
**a** is a parameter of the function and is a **matrix**.
**b** is another parameter of the function is a **scalar**.





*See here for how to use the function into the vsTASKER code.*

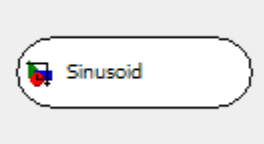
## 8 Help




Any description of the Matlab function can be put here and will be used for the automatic document generation (see Make Documents)

# Simulink Object

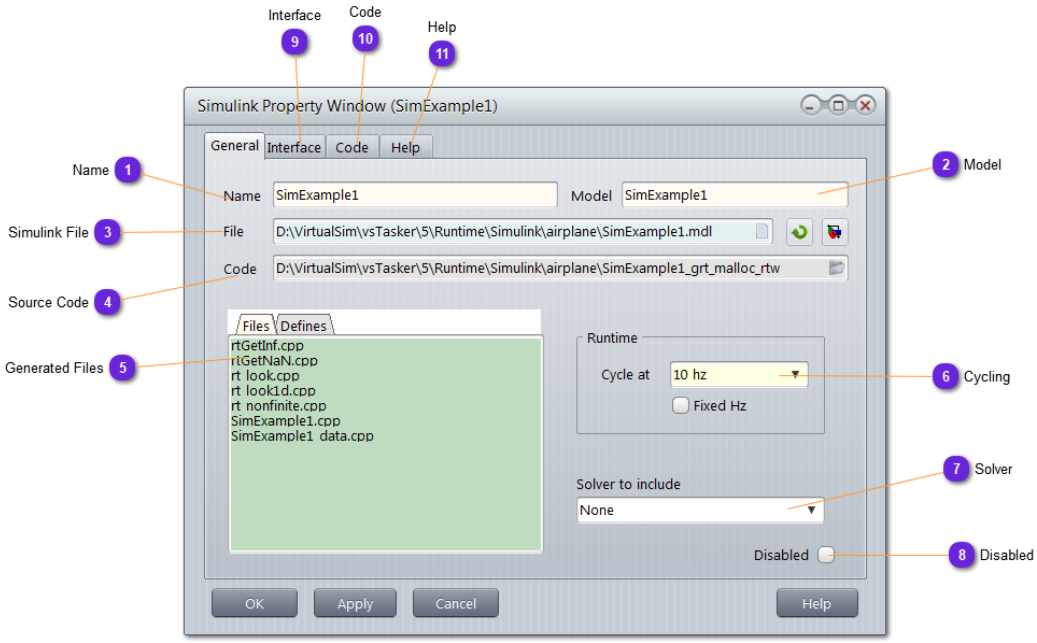
A **Simulink** object embeds a Simulink model defined inside MATLAB Simulink.
It can extract from the model file interfaces, generated files and is capable of updating the makefile in order to compile the generated C code.
The Simulink object is the practical interface between vsTASKER environment (entities, sockets, logics, models, etc.) and the external model.





*For now, only one Simulink model can be included in a database. This is because the generated code of Simulink has static variables that cannot be duplicated. If you need several models, try to combine them inside a same .mdl file.*

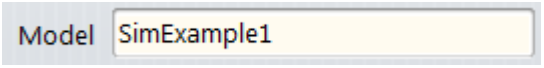
# Properties




## 1 Name

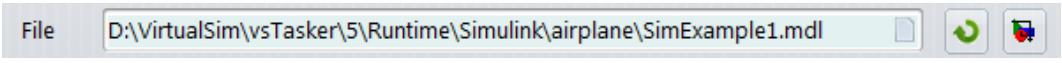

Name of the Simulink object. Must be unique.

## 2 Model




Name of the Model as defined in the .mdl file.

## 3 Simulink File

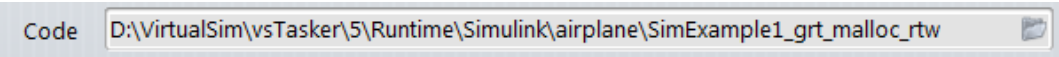



Model file to be parsed and used.

Reload the .mdl file for updating Input/Output.

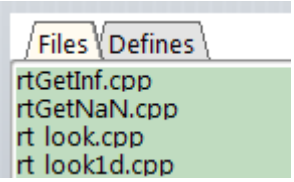
Call the Simulink editor (tries to open/edit the file from a command line. `mdl` extension must be bounded with Simulink)

## 4 Source Code

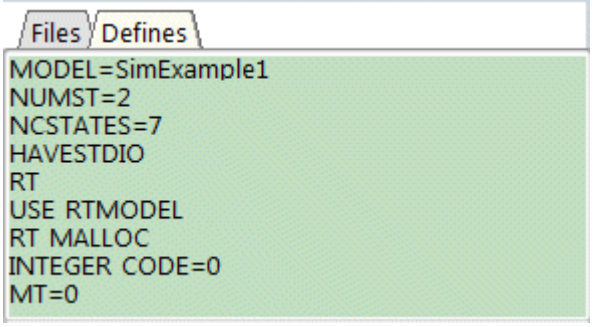



Directory where the Simulink Coder generates all the files for the current model.

## 5 Generated Files

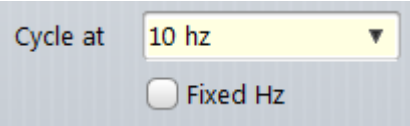



List of all generated code files in (4)




List of all `#define` found in some of the generated header files (4). They will be added in the makefile. They are read-only.

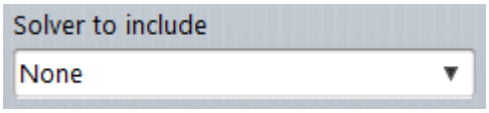
## 6 Cycling




Select from the list the appropriate frequency for the object, according to the Fixed-step defined in the Simulink Coder configuration window.

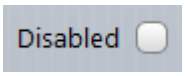
**Fixed Hz**: when checked, the selected frequency is not according to the RTC but according to the wall clock.
With a fixed frequency, even if the simulation engine is accelerated or slowed down, the number of calls per seconds will remain the same.

## 7 Solver




Select here the same solver as the one selected in the Fixed-step defined in the Simulink Coder configuration window.

## 8 Disabled

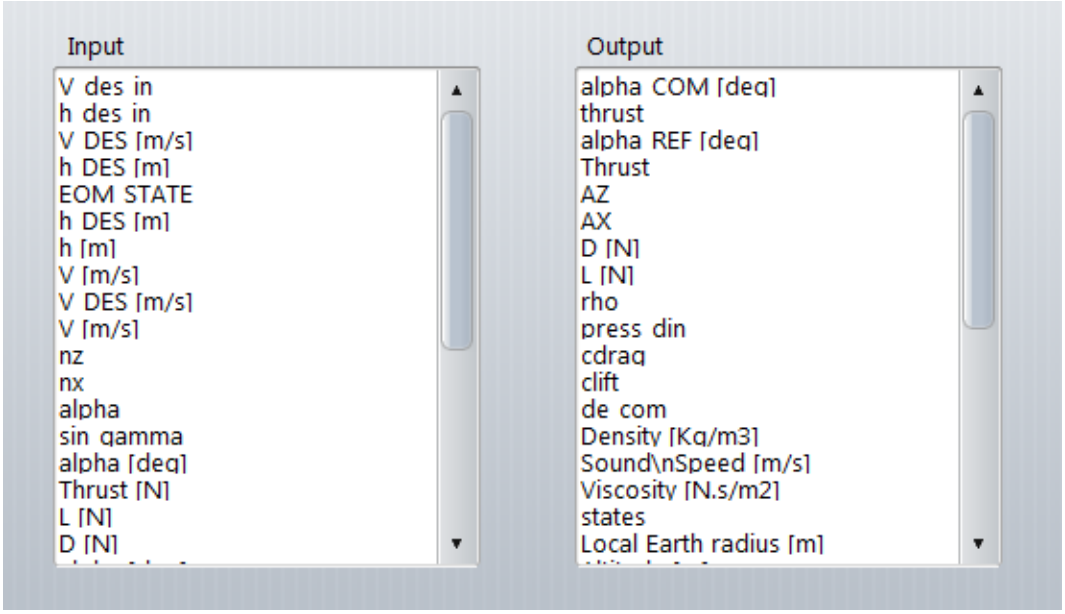



Exclude the Simulink object from the code generation.
Disabled Simulink object will not be included into the items using it (entities but not only) without the need to remove it from all users.
Disabling is error prone while removing might leave dangled pointers.

# 9 Interface

Interface

vsTASKER parses the .mdl file and tries to extract all Inputs and Outputs parameters for corresponding structures.
Not all of them shall be used.



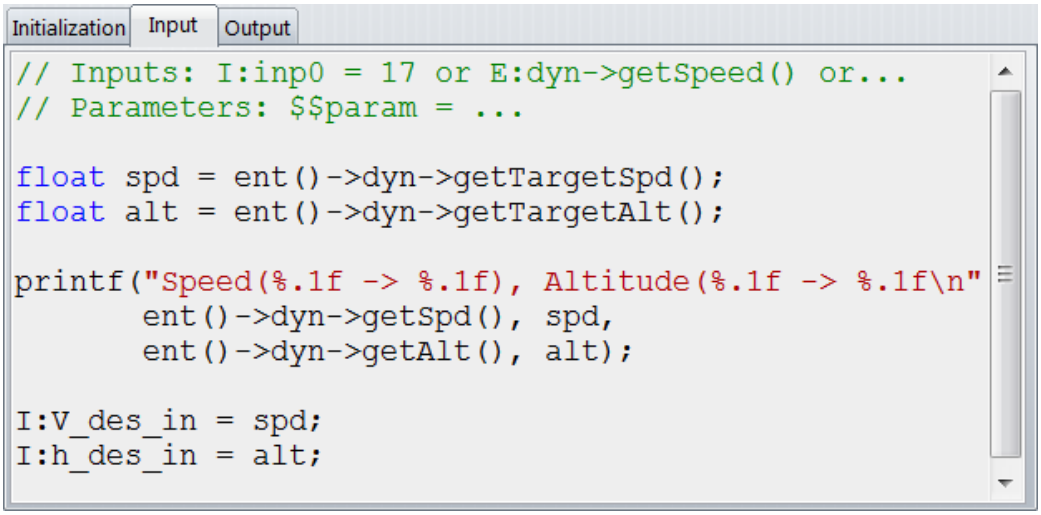

*The parameters names are listed without underscores. Corresponding unit (if any) is expressed in brackets. For i.e:* `V DES [m/s]` *lists the parameter* `V_DES` *whose value is in meter per second.*

## 10 Code

Code

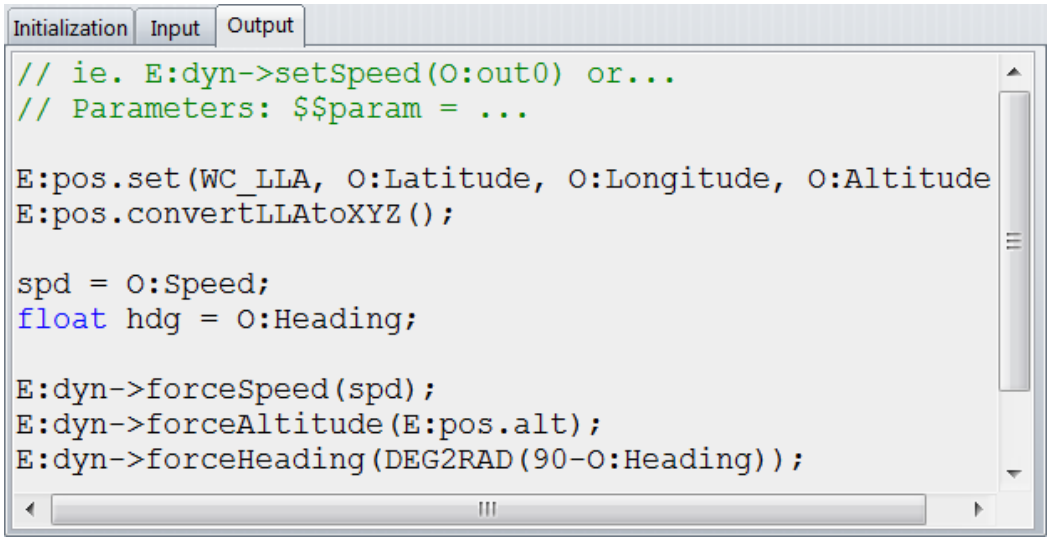
This part is the vsTASKER interface code between the data belonging to the simulation engine and the data pertaining to the Simulink model.
The Initialization part is called at each simulation phase. For i.e, the `INIT/RESET` phase can be used to initialize the Simulink Input parameters.

The Input part is called before the Simulink model `tic()` function. Use this code block to extract all the necessary data from the simulation engine objects to set the Input parameters of the Simulink model.
In the following example, we are extracting the requested speed and altitude of our aircraft to set the corresponding Input parameters.




The Output part is called after the Simulink model `tic()` function. Use this code block to extract all the necessary data from the Output parameters of the Simulink model to set the simulation engine objects.
In the following example, we are extracting the computed speed and heading in our model to update our aircraft.

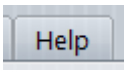
**Properties**





*The Input and Output blocks are called at the frequency set in (6).*

## 11 Help

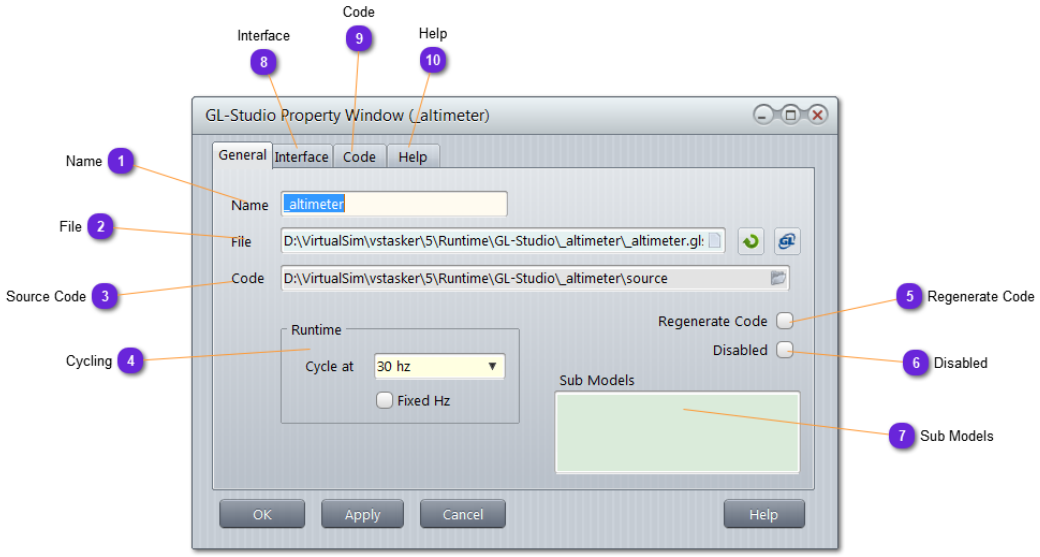



Any description of the Simulink object can be put here and will be used for the automatic document generation (see Make Documents)

# GL-Studio Object

A **GL-Studio** object embeds a GL-Studio model defined inside GL-Studio HMI Builder (from DISTI). The object is relaying on the code generated by the GL-Studio application itself.

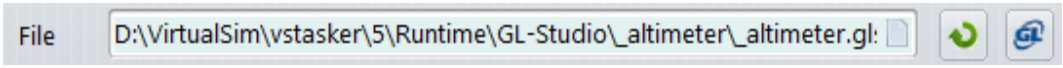
# Properties




## 1 Name


Name of the GL-Studio object. Must be unique.

## 2 File

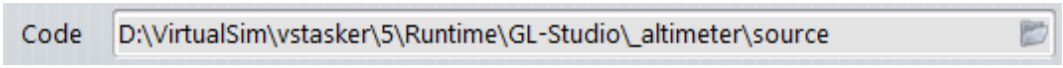



Model file to be parsed and used.

Reload the .gls file for updating the interface.

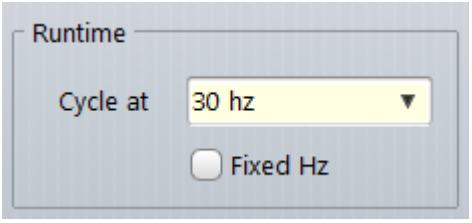
Call the GL-Studio editor (tries to open/edit the file from a command line. gls extension must be bounded with GL-Studio)

## 3 Source Code




Directory where GL-Studio generates all the files for the current model.

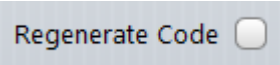
## 4 Cycling




Select from the list the appropriate frequency for the object. As it is a graphic object, the higher the frequency the smoother the result.

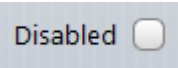
**Fixed Hz**: when checked, the selected frequency is not according to the RTC but according to the wall clock.
With a fixed frequency, even if the simulation engine is accelerated or slowed down, the number of calls per seconds will remain the same.

## 5 Regenerate Code



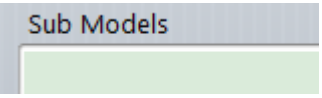

Force GL-Studio to produce a new fresh code for the associated mdl model.

## 6 Disabled



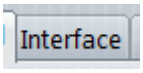

Exclude the GL-Studio object from the code generation.
Disabled GL-Studio object will not be included into the items using it (entities but not only) without the need to remove it from all users.
Disabling is error prone while removing might leave dangled pointers.

## 7 Sub Models

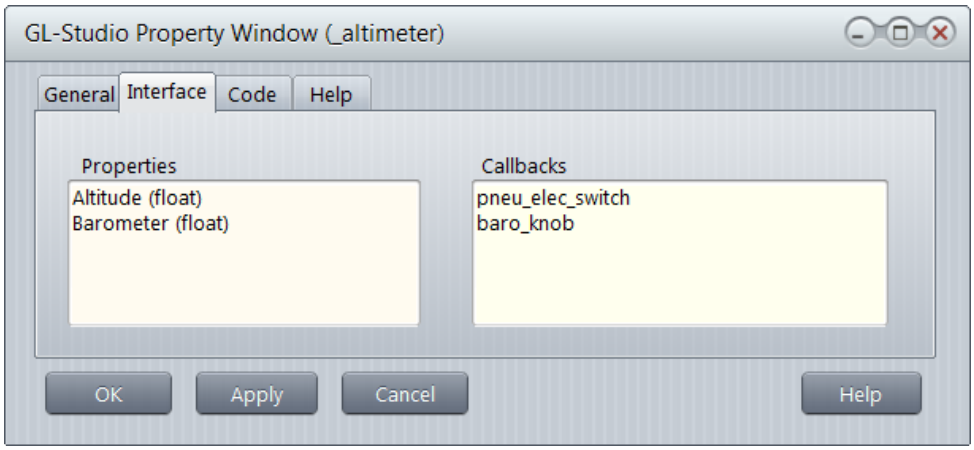



If the current GL-Studio model embed some other models, they will be listed here. Sub models can also be accessed from the source code, so, it is a good hint to get them listed here instead of checking the GL-Studio editor.

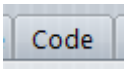
# 8 Interface

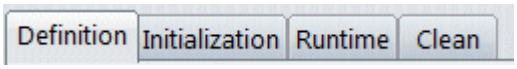



vsTASKER parses the .gls file and tries to extract all Properties and Callbacks.
Not all of them shall be used.



# 9 Code




This part is the vsTASKER interface code between the data belonging to the simulation engine and the objects pertaining to the GL-Studio model.

The Definition panel is used to declare any variable that will be used inside the runtime part (they will persist for all the time of the simulation).

You shall define here user interface variables (`//&&`) for design and runtime setup.

```
// Data & Interface Section:

public:

   float bar;  //&& DEF[28]
   float alt;  //&& DEF[5000]

protected:
```

The Initialization part is called at `RESET` phase can be used to initialize the GL-Studio object as well as local variables.

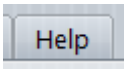
The Runtime part is called **before** each call to the GL-Studio object update function. You can use the local variables to set the object variables.

Use `O:` to access any property or callback defined into the GL-Studio object.

```
// Runtime part called at specified frequency.
// O: gives access to graphical object

 alt = E:getDyn()->getAltitude();

 O:Altitude(alt);
 O:Barometer(bar);
```

The Clean part is called at simulation end (`CLEAN` phase).

## 10 Help




Any description of the GL-Studio object can be put here and will be used for the automatic document generation (see Make Documents)

# Outsourcing a Model

Sometimes, it is a good idea to hide a Model (Data Model or Component) from the user or to avoid having the same Model duplicated into each database that would need it.
Making a Model a library and distributing only its interface will do that.

Built-in Models of vsTASKER have been created this way: designed into vsTASKER, tested, improved until they were complete enough to be extracted from the development database and converted into one header and its associated source file, ready to be compiled and added into the simulation engine library.

The user can do the same with its own models and gather them into specific libraries to be delivered or deployed along with the simulation engine, keeping the code proprietary.

Although the process of generating a library is simple, the inside code may have to be modified because from the library standpoint, their will be no visibility of database specific variables, models or entity parameters. The library component will need to rely on external declaration dependencies and virtual functions.

*Sometimes, it is not possible to outsource a model because of its entanglement with other models or database specific objects. If your model is using code generated variables or models, then it will be very difficult to export it as a library. See **Handling the compilation errors** topic below.*

## • Extraction

Create a myTest component.

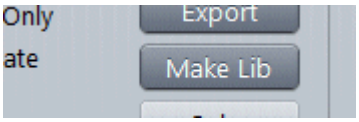
In Declaration, do the following (this is just to add some data in order to give something to do later):

```
// Method Section:
public:
     void init();

// Data & Interface Section:
public:
     float my_speed; //&&  DEF[5]
```

And in Runtime, add the following:

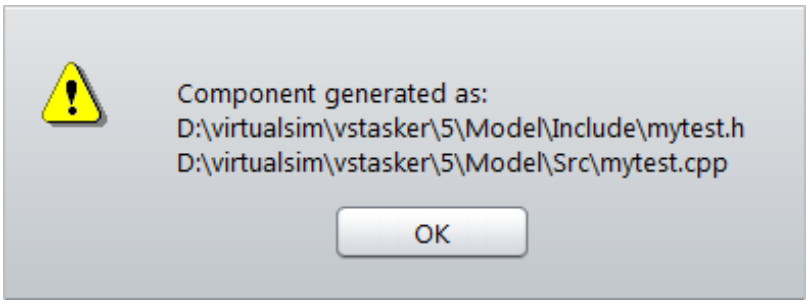
```
my_speed = E:getDyn()->getSpeed();
```

Then, click on **Make Lib** button (and click **Yes** on the question window)




You will be asked to specify where to store the header file of the model, then the source file. It is a good practice to create your own directories inside model/include and model/src ones. You also have the chance to rename the files. By default, the name of the model is taken as is. If you rename it, you will need to change the `#include` directive in the .cpp file.

You will get this result:



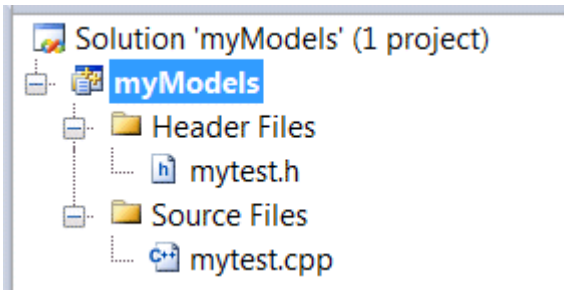

## • Importing to Solution

Now, you need to add these two files into a solution to create a library.
You have the choice to use an existing solution, although it is not a good idea as the next update of vsTASKER will erase all your entries.

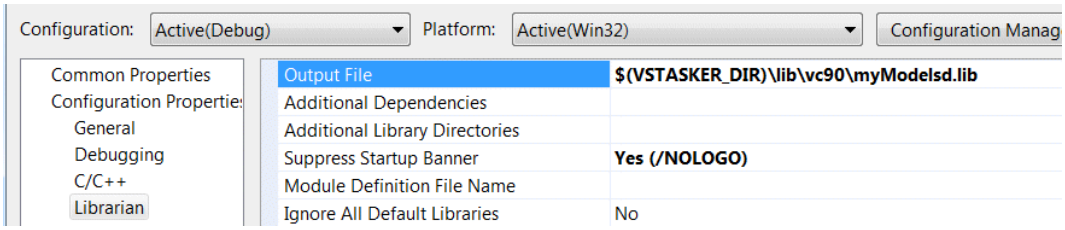
The best approach is to create **your own** folders and solutions.
You can have as many as requested, all organized in different folders.
Here we are going to create a myModels folder in both Model\Include and Model\Src one.
Then, we will move the mytest.h file into Model\Include\myModels and mytest.cpp into Model\Src\MyModels (you can also combine .h and .cpp into the same folder somewhere else).

Now, rename the template solution UsrLib.vcproj (in Model\Make\vc90|100|140) into myModels.vcproj.
Open your C++ VisualStudio and load this solution.

Then, add mytest.h and mytest.cpp in the correct filters (you can remove the template_*.* entries that are useless)




In the myModels properties window, change the name of the library to: $(VSTASKER_DIR)\lib\vc90\myModelsD.lib (vc90|vc100|vc140...) and **D** for debug, nothing for release.




Compile and build myModels.lib and myModelsD.lib libraries.

## • Handling the compilation errors

Although the code generator tries to do its best to out source the model code, there are things that cannot be automatically translated. You will have to manually fix that from Visual Studio.
The most common errors you will encounter are the following:

**Entity** class only exists inside vsTASKER database. From outside (Sim or Libraries), only Vt_Entity is known. Entity is code generated and inherit from Vt_Entity, defined in include/engine/vt_entities.h
So, in the code, `Entity*` must be replaced with `Vt_Entity*` wherever it is possible. If your model is using specific variables defined in Entity class and are code generated, then it will be difficult to access them from a library. You need to reconsider the design of your model or leave it code generated.

## • Updating Directory

### Outsourcing a Model

In order to use the new component from vsTASKER, it must be added into a directory.
In Model folder, you will find some directory files (default_cp.lst, default_dm.lst...)

It is possible to modify these files to add user Components or Data Models but with the risk to lose the entry at each update of the product.
The best way is to create its own entry and add it into the preferences of vsTASKER.

Rename (or duplicate) user_cp.lst into myModels.lst and open it.

The entry in the directory list must be understood as is:

```
+myModel.Components
MyTest      myTest      myModel/mytest.h      myModel.lib
```

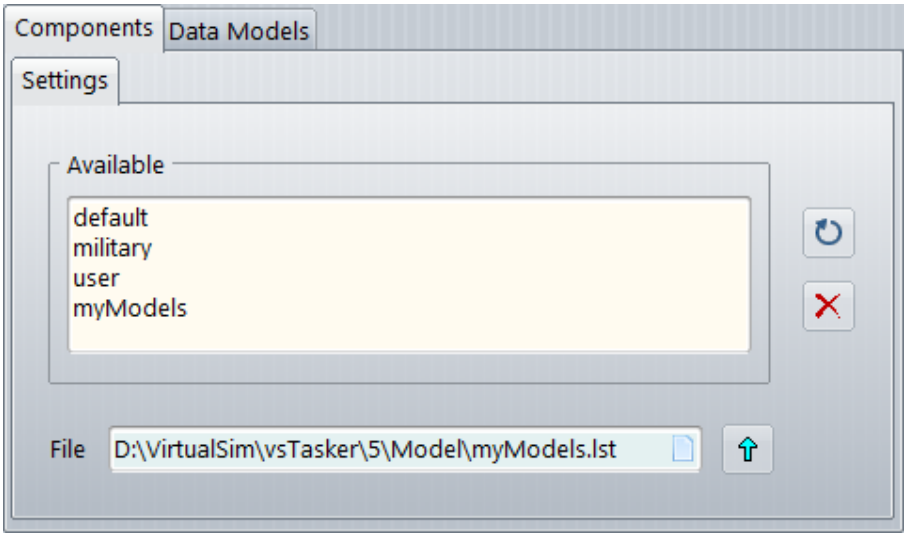
**+myModel.Components**: **+** to name a tree list entry with myModel as the first entry and Components as a embedded list. All following lines (data) will fill the entry until a new list is encountered.

A data line will be made of 4 items separated by spaces or tab:

1) **myTest**, label used in the tree list
2) **myTest**, name of the Component class
3) **myModel/mytest.h**, header file, with relative path from /Model/Include/
4) **myModel.lib**, name of the library where the Component is stored

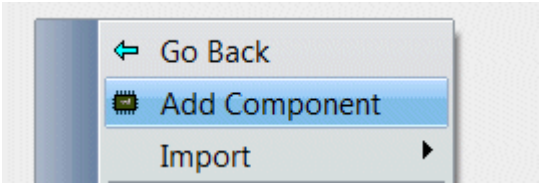
In order to be able to use this new directory list (myModels.lst), it is mandatory to include it into vsTASKER.
**Tools::Preferences::Models...**

Open the D:\VirtualSim\vsTasker\7\Model\myModels.lst file and add it to the list with  , then **Save**.

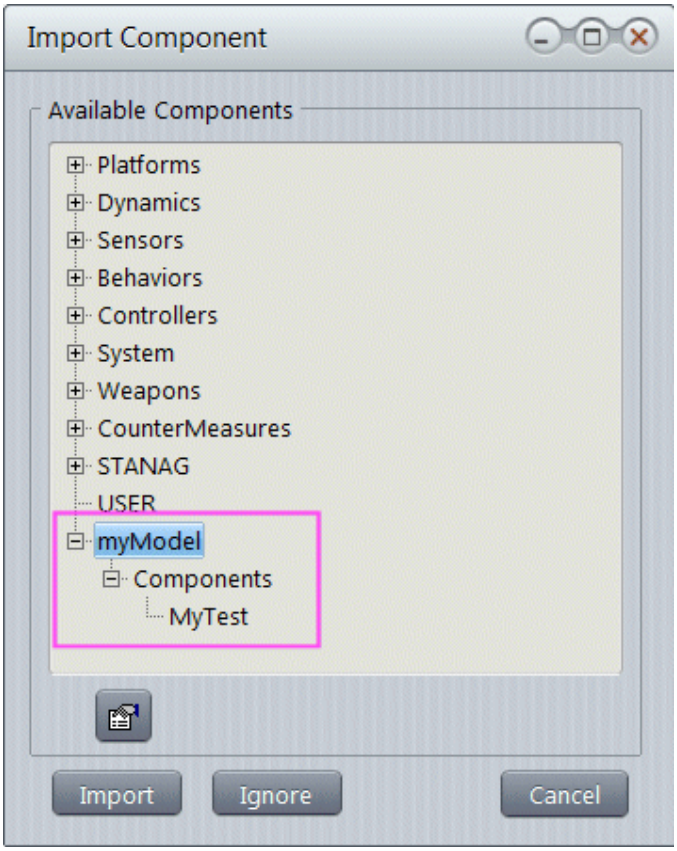



## • Updating the Database

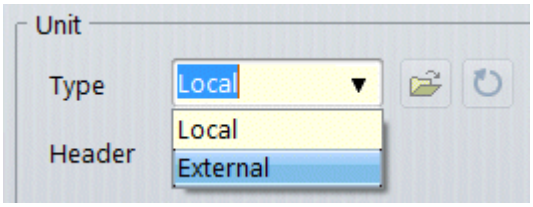
Now, in vsTASKER, importing predefined Components will include myModels component list (1 for now, but the list can grow):



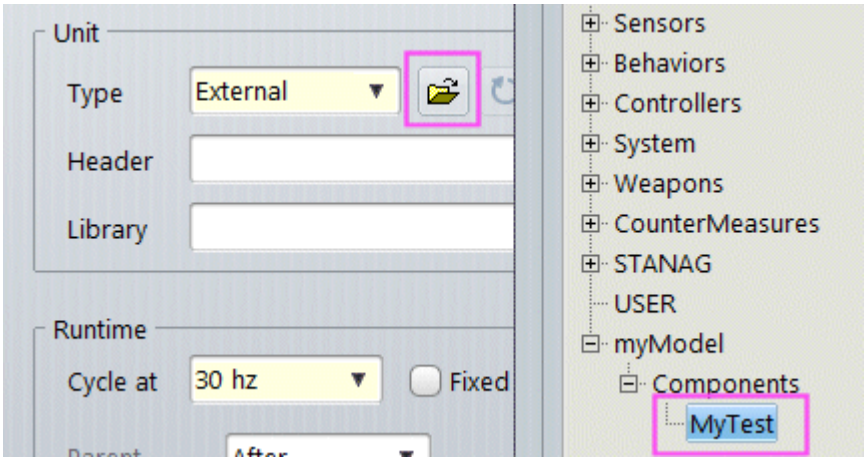
### Outsourcing a Model



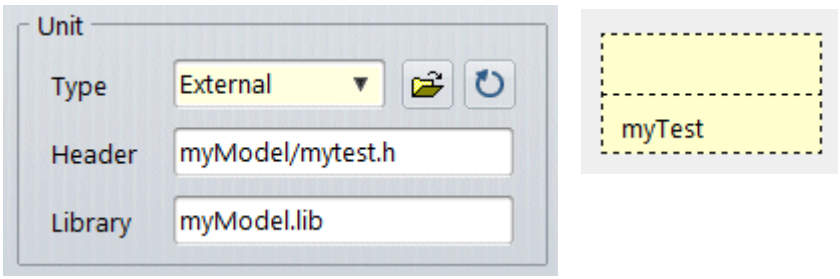

Now that the Component is available as an external source, it can be either imported from the above list, or converted if already in the database (like in this example).
In the myTest Component, change the Local to External:




Then open/select the myTest component to replace the actual Local one:




The myTest Component will then change to External, all references will remain unchanged and the symbol will be changed to read-only.




## • Updating the Model

If a new Dyn-UI variable must be added into the Component once it is external, some precaution must be taken.

In the following example, if we need to add the altitude variable, let's do it this way:

In mytest.h, add the following entry: `float my_altitude; //&& DEF[0]`

In mytest.cpp, do the following:

- inside function: `int myTest::tic()`

add:

```
my_altitude = entity->pos.alt;
```

- inside function: `void myTest::setDefault()`

add:

```
my_altitude = 0;  // same as DEF[] value
```

- inside function: `void myTest::setUserData(UsrInterface* setup)`

add:

```
data = intfc->getValue("my_speed"); if (GOTSTR(data))
setMySpeed(atof(data));
```

- inside function：`void myTest::update(char* ifc)`

add:

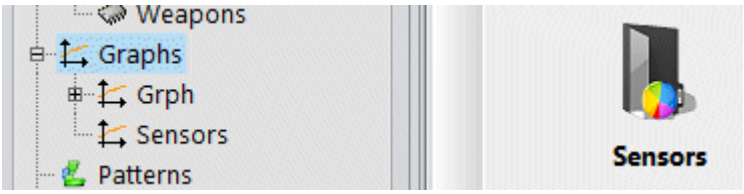
```
Vt_Component::update("my_speed", trimFloat(my_speed));
```

Repeat this process for all Dyn-UI variable additions or changes.

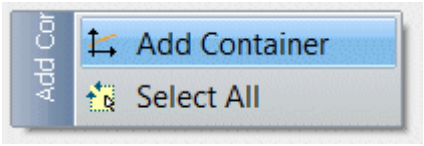
# Graphs

Graphs are combining input curves and output charts capabilities to increase the parameters definition and conveniently display integrated data at runtime with line curves.
Charts and Curves are stored inside Containers (like Models).

## • Environment




First, create a Graph container using either the  button on the vertical toolbar, or with the context menu (mouse right click)

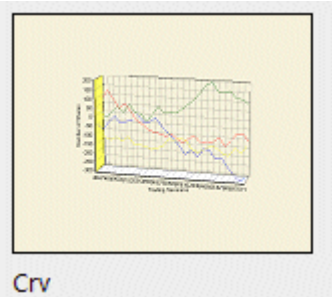



Give a name to this (graph) container (basically, a container will gather charts and curves of the same type or usage).

Then open the container or double click the symbol to get inside.
A container can hold several Curves and Charts.

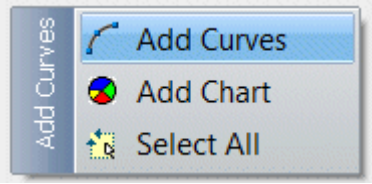
# Curves

Curves are Graph objects intended to represent data in a form of one or multiple curves sets. Curves are drawn on the screen using the mouse or imported from other sources.
Curves data are stored in the runtime database and loaded at runtime.
An API allows any runtime object (Entity, Logic, Component, etc.) to query a curve to get interpolated or extrapolated data.
Curves are a powerful way to replace a mathematical function when only empiric data is available.




## • Environment

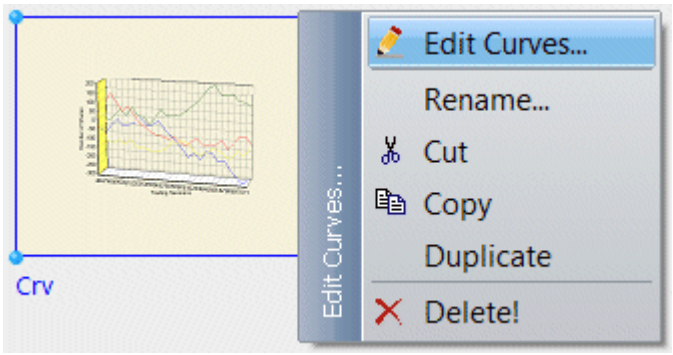
To add a new Curve (set) to a Container (once inside), use the button on the vertical toolbar or use the context menu (mouse right click)




Give a name to the Curve.

## • How to Define

First, select the Curves and use the **Edit** popup menu to set the properties




**Edit Chart**: see Properties and Data for details

**Rename**: change the name of the current selected Curves

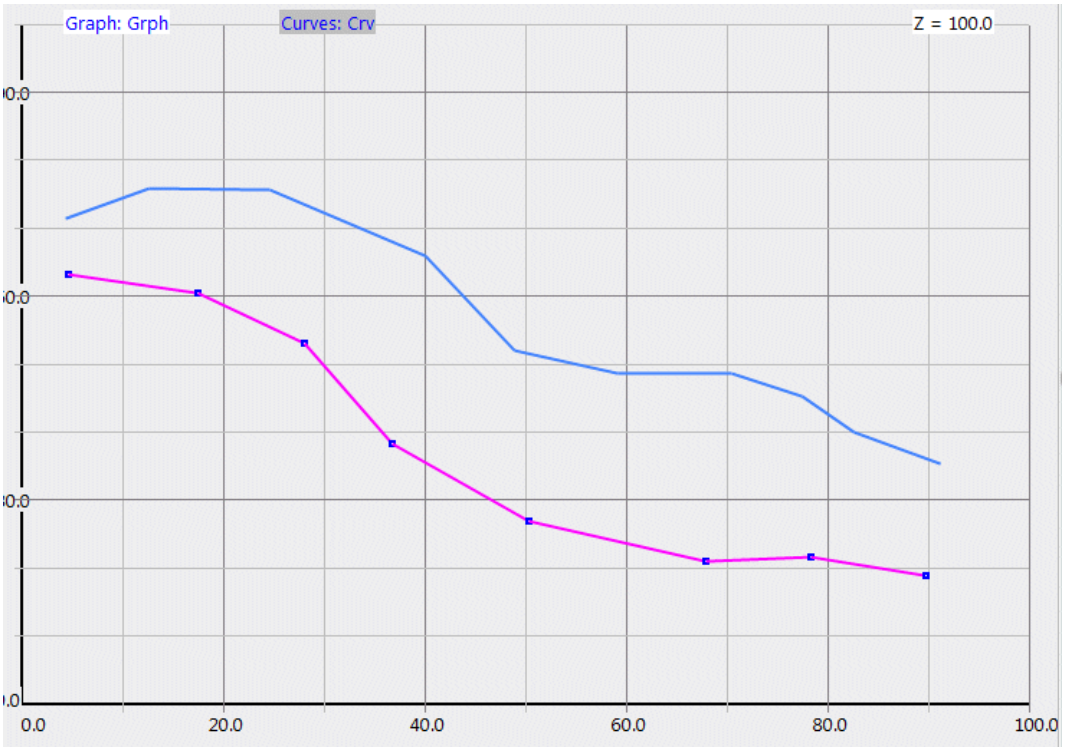
**Cut**: copy the Curves to the clipboard and remove it from the current container

**Copy**: copy the Curves to the clipboard

**Duplicate**: do a copy then paste on the same container

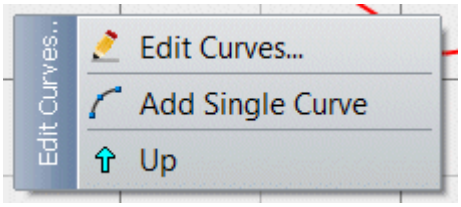
**Delete**: remove the selected Curves from the container and Database.

**Double click** on the Curve icon to edit the Curve(s) shapes.



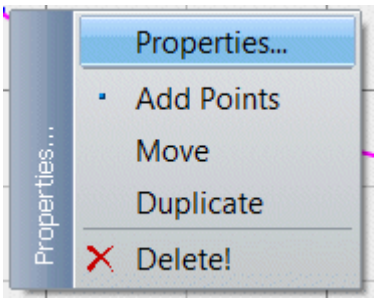

**When nothing is selected:**

### Curves




**Edit Curves**: call the properties window.
**Add Single Curve**: select this option to create a new curve with a Z value (available only for multiple curves). Click on the graph to enter points until done.
**Up**: Close the Curves and return to the container view.

To select a Curve, just click on the curve itself. Color changes to magenta and Points are displayed.
**When a Curve is selected**:




**Properties**: call the properties window.

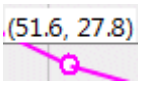
**Add Points**: Use this option then click on the graph to enter points until done.

**Move**: select this option then click on the Curve and with the mouse button down, drag the curve on the graph.

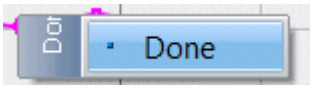
**Duplicate**: create another similar Curve, offset on the graph. Note that Z value is kept unchanged.

**Delete**: remove the whole curve from the set.

To **reshape** the Curve, just select a Point and drag it around with the mouse.

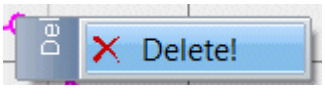



To **add new** points in a selected Curve, click the button on the vertical toolbar (or use **Add Points** on the popup menu) then click on the graph everywhere new points must be added.
To finish the insertion, just right click and use **Done**.




\

To remove a Point, select it and use **Delete**.



## • How to Use

When a Curve (or a set of Curves) is defined, the purpose is using it to retrieve the Y value given the X (and maybe Z) value(s).
From the source code of any vsTASKER object, do the following to get the curve from the graphs library:

```
 Vt_Curve* my_curve = (Vt_Curves*) vt_rtc->graphs->find(TT_Curves,
"container::curve"));
```

where container is the name of the container holding the Curve and curve is the name of the curve. If container is not provided, the first curve which will match the name will be returned.

Using the returned curve (pointer), use the simple `getValue()` function to retrieve the Y:

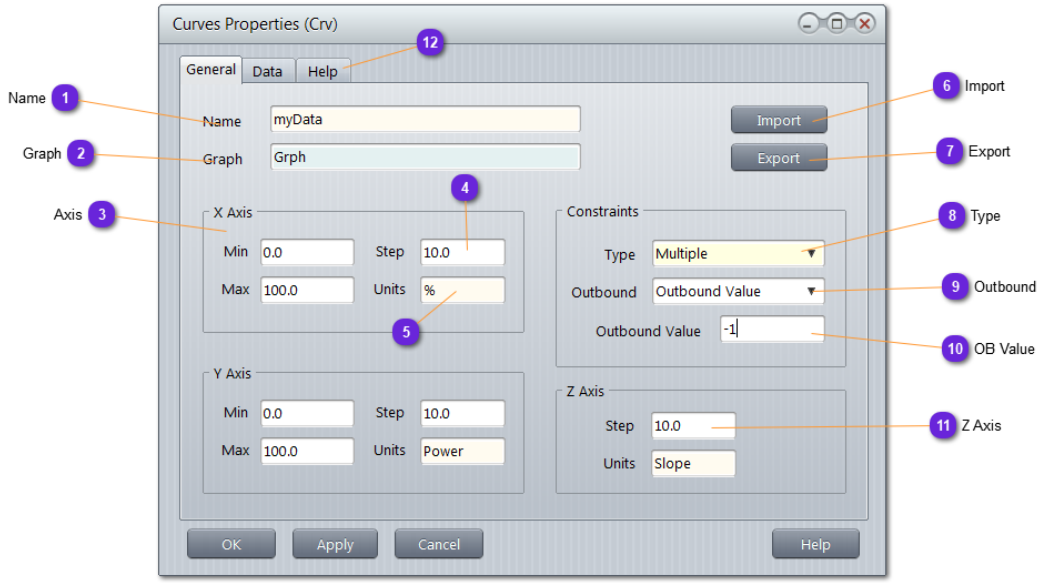
```
 float y = my_curve->getValue(x);
```

or

```
 float y = my_curve->getValue(x,z);
```

You can get the full API list in /include/engine/vt_curves.h

# Properties




## 1 Name

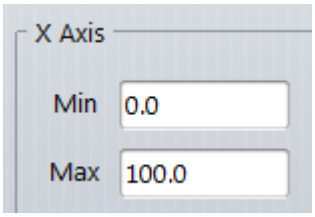

Name of the Curves that could be duplicated as the name of the Graph (container) must prefix the Curve name to give the full name needed by the finder.


*It is allowed to give the same name to different Curves in different Graphs. The unique name to find them will be the concatenation of the Graph name and the Curve name (separated with two colons), otherwise, only the first Curve with the matching name will be returned.*

## 2 Graph

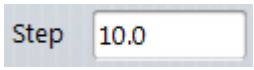

Name of the Graph container that holds the current Curves.

## 3 Axis



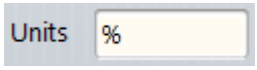

X (horizontal) and Y (vertical) axis setting are used to draw the Curves canvas.
Min and Max values are boundaries for all curves. Use float values.

## 4 Step



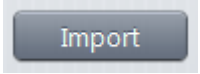

This value is used to draw the grid. It indicate the value between two grid lines (vertical or horizontal).
Do `(max-min)/step` to guess the number of lines that will be displayed.
Should not be zero.

## 5 Units




This string value is only used for display to give a meaning to the value used in X or Y axis.

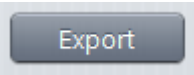
## 6 Import




Use this button to import a previously exported Curves in /Data/Shared.


*An exported Curves has extension .crv*

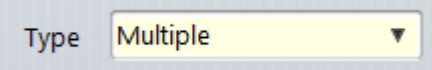
## 7 Export




Use this button to export a Curves for reuse or share between databases or users.
The Curves is exported by default in /Data/Shared but can be saved in any directory or support.


*An exported Curves has extension .crv*

## 8 Type

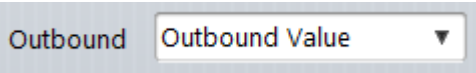



Specify here how many curves will be defined:

- Single: only one curve will be available. No Z value then.
- Multiple: several curves will be available, all with a unique Z value.


*Single curve is for simple data, like the mean temperature (Y) over the year, from January to December (X).*
*On this same example, if the data is available, Multiple curves will provide temperatures per year (Z), from let's say 1990 to 2010.*

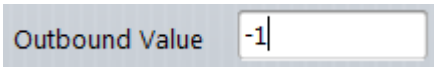
## 9 Outbound




Specify here the strategy for the function returning the **Y** value (given **X** and maybe Z), when **X** (or Z) is below Min or above Max:

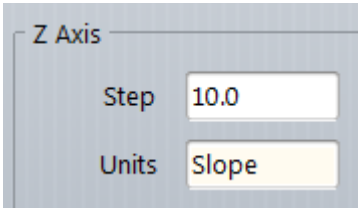
- Extrapolate: use the slope of the curve to compute the projected value.
- Last Valid: take the first or last Y value and returns it. No extrapolation.
- Out bound Value: returns the user value defined in (10)

## 10 OB Value



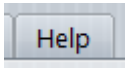

When out bound strategy is set to out bound Value (see above), enter here the value that will be returned whenever **X** (or Z) is below Min or above Max.

## 11 Z Axis



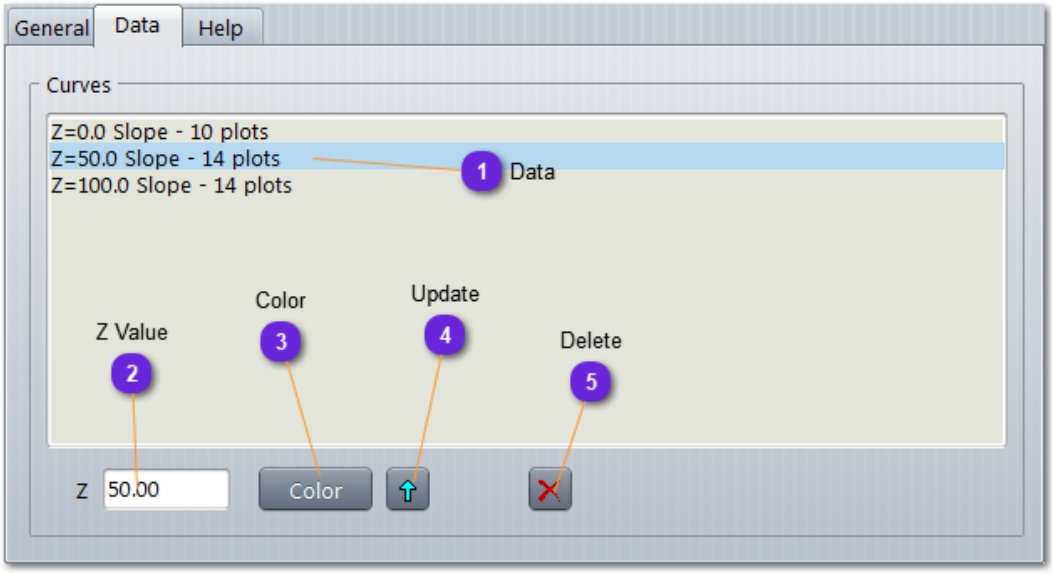

Same as for X and Y Axis, but for the Z axis.
Only available for Multiple Curves (8).

## 12 Help

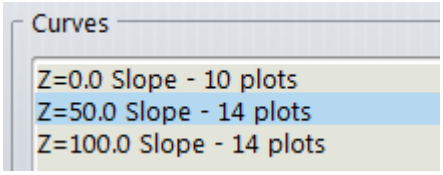



Any description of the Curves can be put here and will be used for the automatic document generation (see Make Documents)

# Data

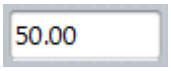



## 1 Data



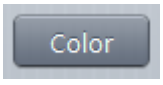

List of all the Curves (in case of Multiple) or the only one (Single).
One only can be selected.

## 2 Z Value




Display the Z value of the selected Curve (1).
You can change the value and update with ⇧.

## 3 Color




Define the color of the selected Curve, as it will be drawn on the Diagram.
Set the color from the selection panel and update with ⇧.

## 4 Update

If you change the Z value or the Color, use this button to update the selected Curves (1) or the changes are lost.

## 5 Delete

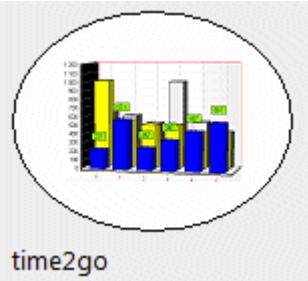
Delete the selected Curves.

*There is no questioning and no undo, so, be cautious.*

# Chart

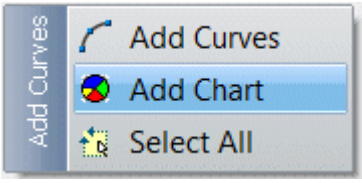
Chart is a runtime drawing object intended to provide to attached entities, simple mechanism to store data in runtime and have them displayed during the run or after the simulation end. Chart is typically used for Data Analysis as it can be perfectly combines with the Batch Mode.

A Chart is given to entities that have capabilities to write data, altogether, into a unique file that will be kept open for the duration of the simulation. It will be close and simulation end.
The Chart can load the last relevant file and display its content, during the runtime or after.
It also can load any other chart file produced by previous simulation runs.



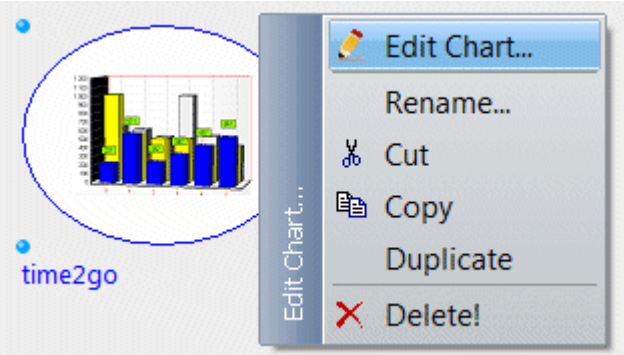

## • Environment

To add a new Chart to a Container (once inside), use the button on the vertical toolbar or use the context menu (mouse right click)




Give a name to the Chart.

When a **Chart is selected**:




**Edit Chart**: see Properties

**Rename**: change the name of the current selected Chart

**Cut**: copy the Chart to the clipboard and remove it from the current Container

**Copy**: copy the Chart to the clipboard

**Duplicate**: do a copy then paste on the same Container

**Delete**: remove the selected Chart from the Container and Database.

## • How to Use

When a Chart is defined, the purpose is to be fed by data to display. The simulation engine, at runtime, produces a cht file in the database directory itself. Every new run produces a new cht file whose name format is <scenario name>_<uniq number>_<database name>.cht

From the source code of any vsTASKER entity that has been registered to the Chart, do the following:

```
E:setChartData("time2go", X, Y);
```

Where `E:` stands for the Entity, `time2go` is the name of the Chart (could also be <graph name>::<chart name>) and `Y` value on the `X` axis.

You can get the full API in /include/engine/vt_charts.h

## • Runtime Entities

An entity must be registered into a Chart in order to be able to use it. At design time, entity must be manually added into the Chart using the button  .
If entity is created at runtime, it must self register using specific code:

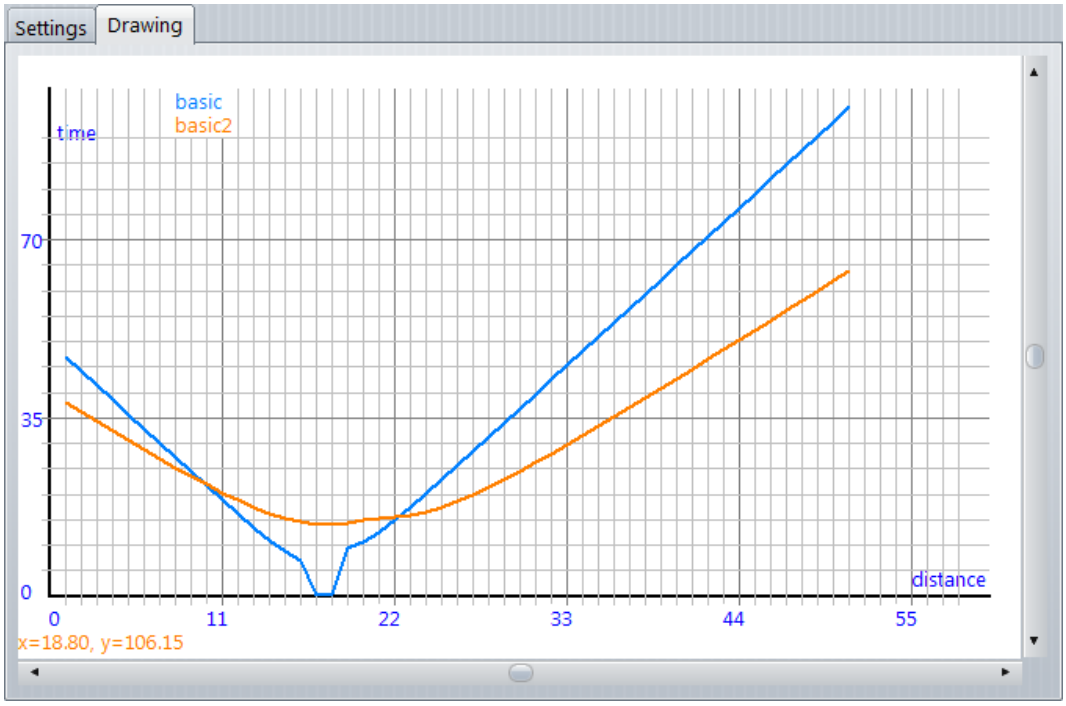
```
R:graphs->registerTo("myChart", this->entity);
```

The runtime data file will then need to be loaded manually in order to be displayed as the GUI will not be informed.

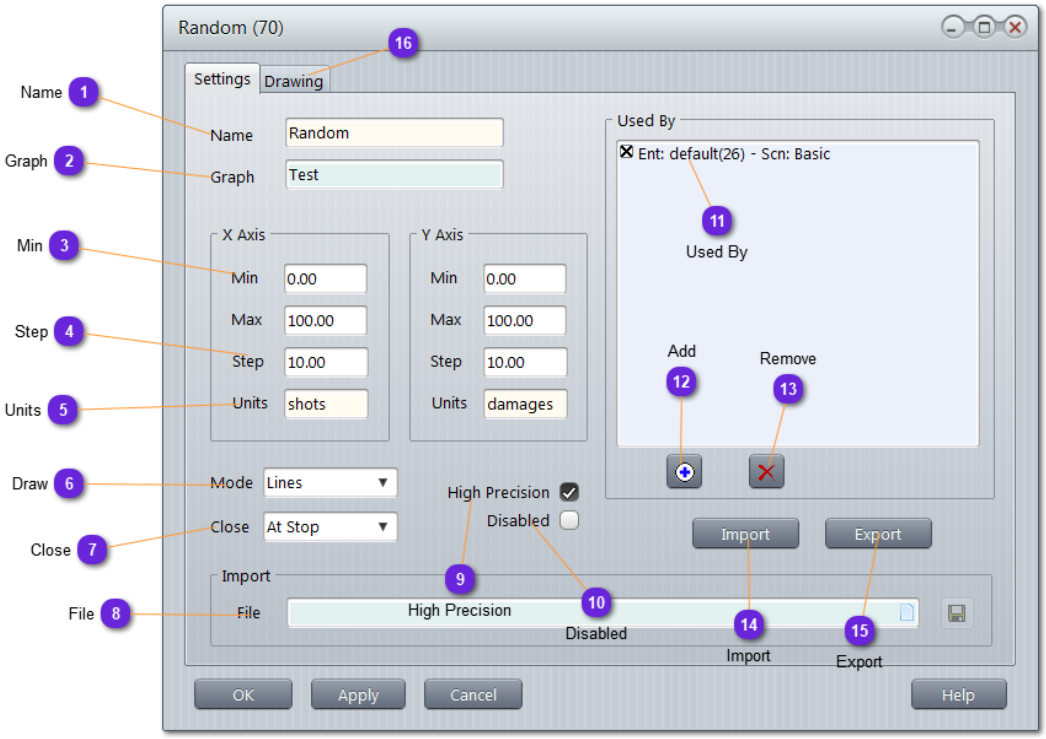
## • Result

During the simulation run, open the Chart and click on the Drawing tab to see the actual content of the file as being written by the registered entities.
Each curve will represent the data written by each entity with its own color.

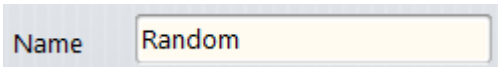
## Chart


Settings
Drawing
basic
basic2
time
70
35
0
0
11
22
33
44
55
distance
x=18.80, y=106.15

# Properties



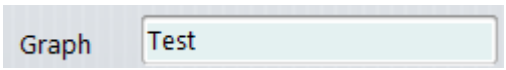

## 1 Name




Name of the Chart that could be duplicated as the name of the Graph (container) must prefix the Chart name to give the full name needed by the finder.

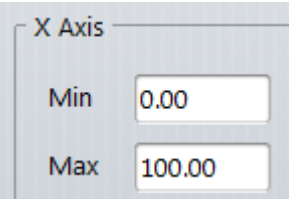
*It is allowed to give the same name to different Charts in different Graphs. The unique name to find them will be the concatenation of the Graph name and the Chart name (separated with two colons), otherwise, only the first Chart with the matching name will be returned.*

## 2 Graph



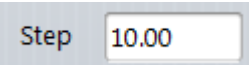

Name of the Graph container that holds the current Chart.

## 3 Min



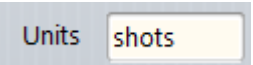

For X (horizontal) and Y (vertical) axis, specify here the boundaries of the Chart graph itself, in how it will be displayed.
If values are out bound, they will not be displayed. The area defined with the boundaries defines the window.

## 4 Step



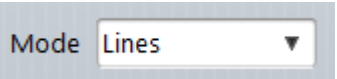

This value is used to draw the grid. It indicate the value between two grid lines (vertical or horizontal).
Do `(max-min)/step` to guess the number of lines that will be displayed.
Should not be zero.

## 5 Units




This string value is only used for display to give a meaning to the value used in X or Y axis.

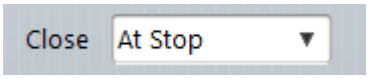
## 6 Draw




Select here the kind of drawing that must connect entries of the chart, for each entity.
Color used is the same as the entity the chart data set is attached to:

- Lines: all entries are connected, from first to last
- Dots: all entries are displayed as a single dot

## 7 Close



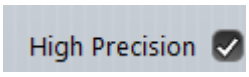

Specify the closing strategy for the data file:

- At Stop: data file will be closed at simulation stop. This choice is good when one file per run is requested.
- At Exit: data file will be closed at exit. This choice is good with batch, when several run (start and stop) are requested to provide the data to the file.

## 8 File


During the simulation run, each Chart received the name of the last data file written (for runtime display).
Offline (or after action), use this file selector to retrieve any previous data file recorded.

## 9 High Precision

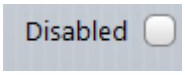



When High Precision is checked, all data inputs (from the same entity) will be stored.
If unchecked, only data with a **X** variation of **1/2** of the **X** step will be kept. This will result in smaller data files.


*In Normal Precision: If the X Axis Step value is 1:*

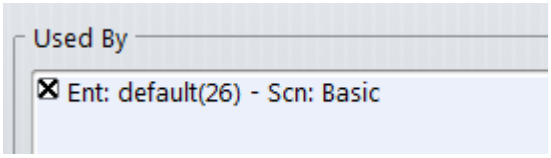
```
     E:setChartData("name", 0, 0);      // Accepted, first
call
      E:setChartData("name", 0.25, 1);  // Skipped, 0.25-0 <
      1/2
      E:setChartData("name", 0.5, 2);   // Accepted, 0.5-0 >=
      1/2
```

## 10 Disabled




If checked, the data file will not be used at runtime and Entities will not write anything to it.

## 11 Used By




List all the Entities from all the Scenarios that are entitled to send data to this Chart, at runtime.
All Charts are private. An Entity not registered cannot record anything to a Chart, even if the data is sent.

*When a listed entity is unchecked, the corresponding data will not be displayed on the Data panel, but at runtime, the entity will still be able to fill in new data.*

## 12 Add

Use this button to register a new Entity (from the current Scenario) to the Chart.

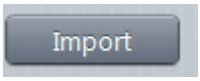
*To add the Scenario **Player** (if this one only can write on the data file), select ScnPlayer from the list.*

## 13 Remove

Use this button to unregister the selected Entity from the Chart.

*There is no questioning and no undo. If you unregister one entity by mistake, just put it back (11).*

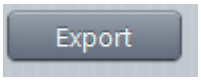
## 14 Import




Use this button to import a previously exported Chart in /Data/Shared.


*An exported Chart has extension .cht*

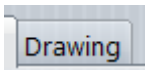
## 15 Export



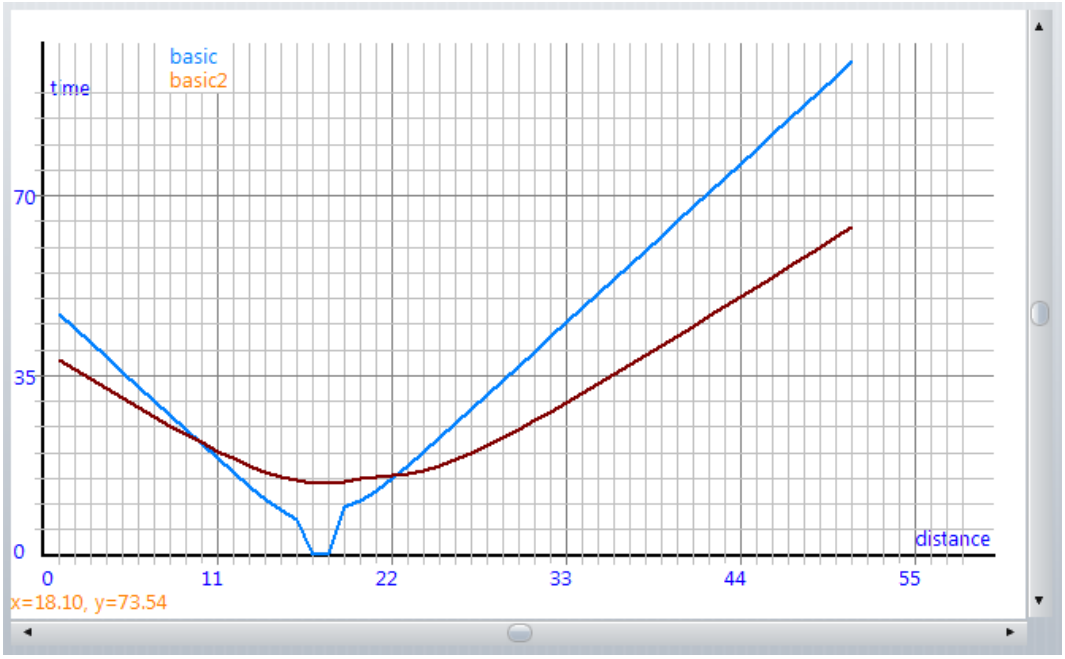

Use this button to export a Chart for reuse or share between databases or users.
The Chart is exported by default in /Data/Shared but can be saved in any directory or support.


*An exported Chart has extension .cht*

# 16 Drawing




This drawing area displays all entity output data files combined and shown with different colors (each entity gives to its curve its own color).
At runtime, the update rate is 1 Hz by default.
Moving from General to Data panel refresh all data by forcing the reading of the file.

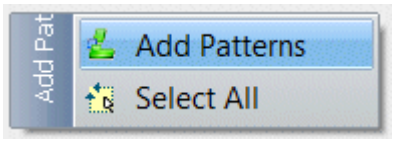


# Patterns

Patterns are use to define either formation shapes or search paths.
They are conveniently located here for instant drop into the Scenario (search paths) or to visually represent in a plan sheet the shape which must be kept by members.

## • Environment


First, create a Pattern container using either the  button on the vertical toolbar, or with the context menu (mouse right click)




Give a name to this (pattern) container (basically, a container will gather charts and curves of the same type or usage).

Then open the container or double click the symbol to get inside.
A container can hold several Formations and Search-Pattern.

# Formation

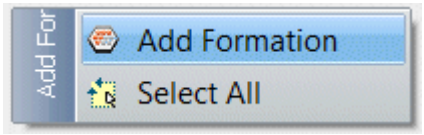
A Formation is a pattern subset that defines a geometrical representation of positions in space, regarding a master position, used to make several entities move in a coherent or requested shape. The Entities belonging to a Formation does not need to be organized as a Unit (although they can and often, they are). Each Entity are given a Slot relative to each other and during the simulation run, each Entity will try to remain as close as possible to their Slot according to their own dynamic.



*The formation moves with the Master and according to the heading change of the Master, the formation shape might suddenly change in location. This is normal behavior and only the component in charge of the Formation following shall reduce the impact of such a change.*

## • Environment

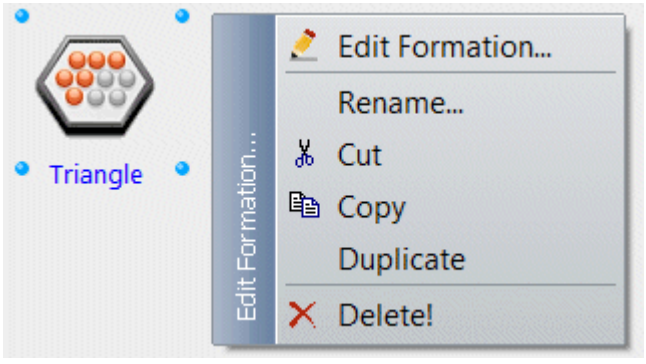
To add a new Formation to a container (once inside), use the button on the vertical toolbar or use the context menu (mouse right click)




Give a name to the Formation.

## • How to Define

First, select the Formation and use the **Edit** popup menu to set the properties




**Edit Formation**: see Properties for details

**Rename**: change the name of the current selected Formation

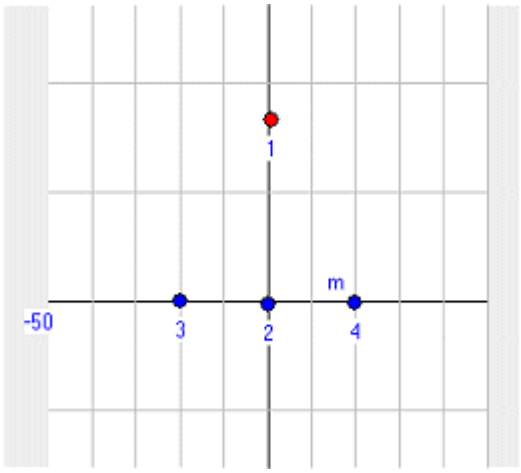
**Cut**: copy the Formation to the clipboard and remove it from the current container

**Copy**: copy the Formation to the clipboard

**Duplicate**: do a copy then paste on the same container

**Delete**: remove the selected Formation from the container and Database.

**Double click** on the Formation icon to edit the slots:

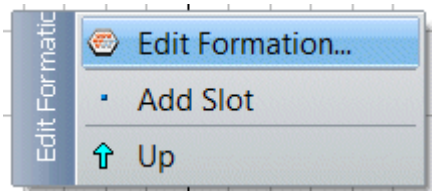



The formation shape is defined in the GUI by adding and moving slots on the drawing.

- To **create** a new slot, right click over the drawing area then select **Add Slot**.
- To **reposition** a slot, select it and move it.
- To **remove** a slot, select it, right click and **Delete**.

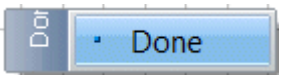
The first added slot is the **master** one (red). All successive slots are **members** (slaves).
You can create as many as needed.

**When nothing is selected:**




**Edit Formation**: call the properties window.
**Add Slot**: click on the diagram to add Slots, until **Done**. First one is the Master (red), successive are slaves (blue).
**Up**: Close the Formation and return to the container view.

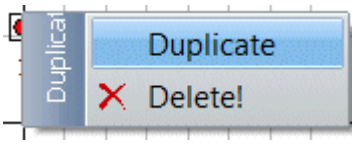
To stop add Slots, right click, then:




**Done**: stop insertions.

To select a Slot, just click on the Slot itself. A little square will surround it.

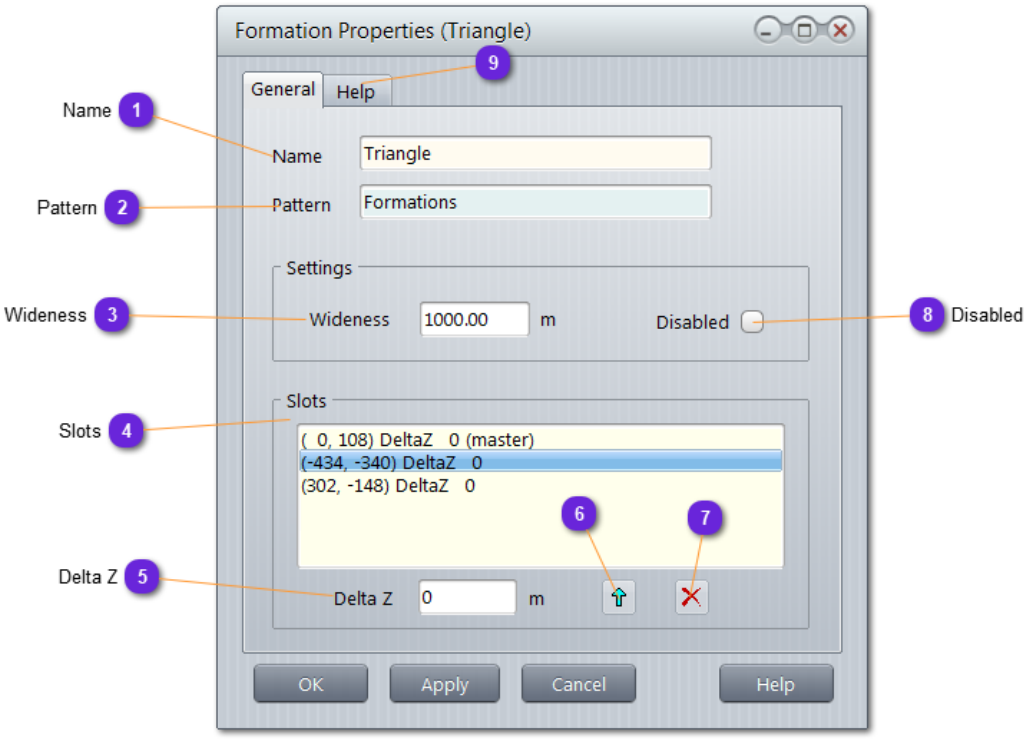
**When a Slot is selected**:




**Duplicate**: create another similar Slot (slave in all cases).
**Delete**: remove the Slot from the Formation (no question).

To **reshape** the Formation, just select a Slot and drag it around with the mouse.

# Properties




## 1 Name


Name of the Formation that could be duplicated as the name of the Pattern (container) must prefix the Formation name to give the full name needed by the finder.

*It is allowed to give the same name to different Formation in different Patterns. The unique name to find them will be the concatenation of the Pattern name and the Formation name (separated with two colons), otherwise, only the first Formation with the matching name will be returned.*

## 2 Pattern

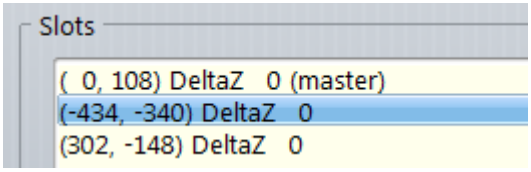

Name of the Pattern container that holds the current Formation.

## 3 Wideness


Size of the area where the Formation will be defined.

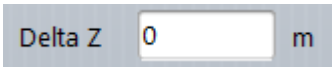
*The formation component will have the ability to scale this Formation. Default is 1 so, it is a good guess to consider that the design scale is the default one.*

## 4 Slots




List of all the Slots of the Formation.
(**position** respectively to the center of the Formation) **Z offset** regarding the master altitude

## 5 Delta Z




Specify here the altitude offset value of the selected slot regarding the master altitude. Change the value and update with button.
This can only apply to air or submarine entities.

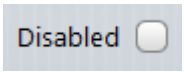
## 6 Update

Use this button to update the Delta Z value (5) for the selected Slot.

## 7 Delete

Delete the selected Slot. No questioning.

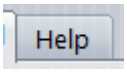
## 8 Disabled




If checked, the Formation is disabled and the component in charge will not run.


*Disabling a Formation is an easy way to prevent entities to join the formation, even if they have the component setup in their behaviors.*

## 9 Help




Any description of the Formation can be put here and will be used for the automatic document generation (see Make Documents)

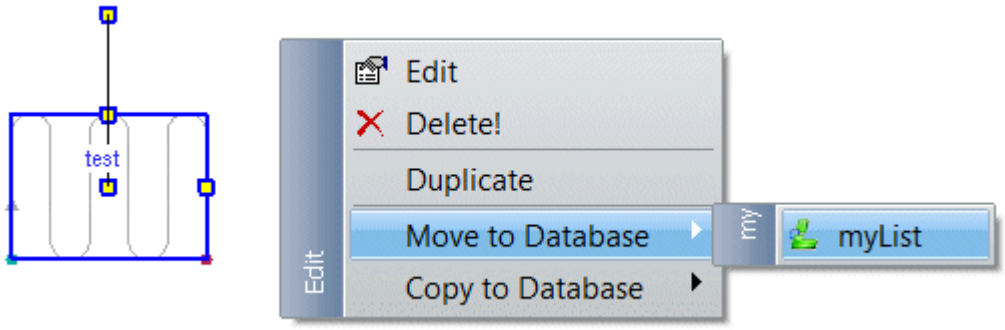
# Search Path

Search patterns can be stored into the database as Search-Path in one or several Pattern containers.
Stored Search-Paths can be reused and copied into the current scenario without having to redefine them.


## • How to Store

Create a Search-Pattern on the scenario, as a Feature.
See here on how to do.

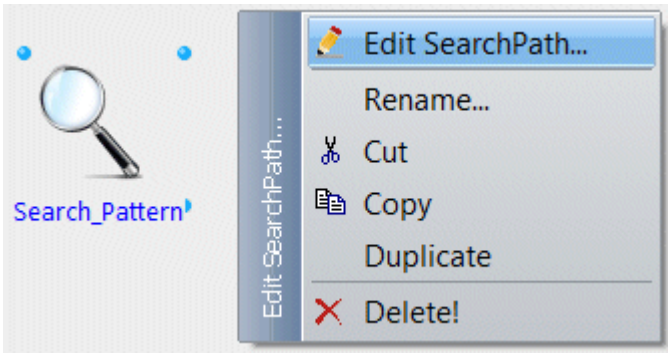
Now select the Search-Pattern (test), right click, select **Move to Database** and from the drop down list, select the myList container where to Move or Copy the selected Search-Pattern.
**Move**: copy and delete from the scenario
**Copy**: copy and leave it.




The test Search-Pattern is now in the myList container as a Search-Path.


## • How to Use

When selected, popup menu from the mouse right click:




**Edit Search-Path**: see Properties for details

**Rename**: change the name of the current selected Search-Path

**Cut**: copy the Search-Path to the clipboard and remove it from the current container

**Copy**: copy the Search-Path to the clipboard

**Duplicate**: do a copy then paste on the same container

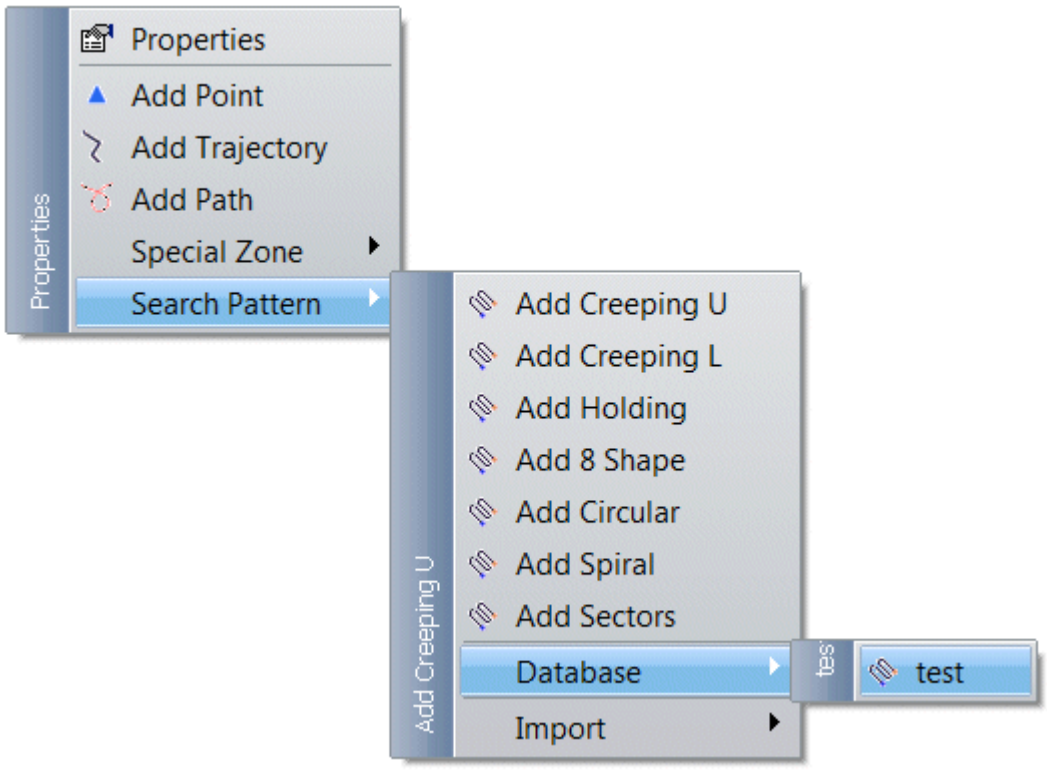
**Delete**: remove the selected Search-Path from the container and Database.

Double click on any Search-Path opens their property window.

## • How to Instanciate

**On the scenario map, under Feature tab, use the popup menu and select the Search-Path to instanciate from the Database drop down list.**

### Search Path

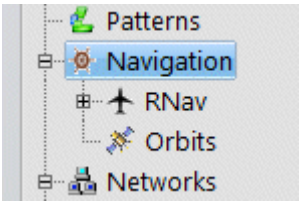


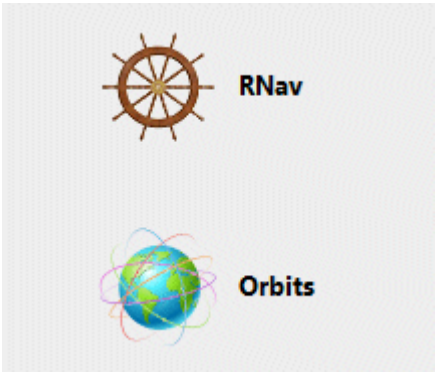

*You can instanciate at design and runtime.*

# Navigation

Navigation category gathers general capabilities for Area Navigation and Orbiting spacecrafts.

- **Environment**



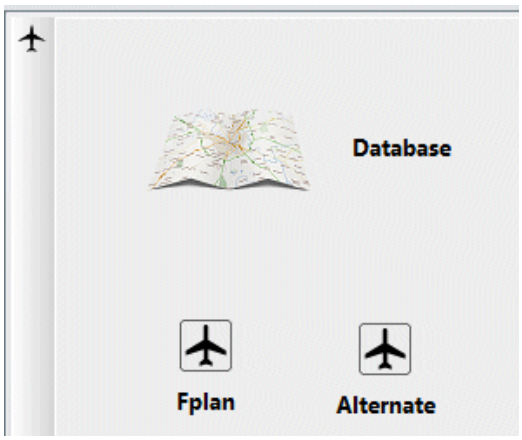




Select Navigation in Environment, then on the diagram panel, click on each symbol to open each definition.

# RNav

RNav (Area Navigation) database includes data imported from ARINC 424 databases using vsTerrainBuilder and Flight Plans used by aircraft entities doing flight managed navigation.
These definitions are commonly used for ATC (Air Traffic Control) simulation.


## • Environment




Click on the Database (map) icon to popup the RNav property window.
Go here for more explanations.

Go here for more details.

# Flight-Plans

Flight Plans (FP) provide a way to define a FMS planning for Aircraft, based on aeronautical data and procedures.
A Flight Plan is made from a definition of waypoints (including standard Terminators) that can be followed by any named aircraft.

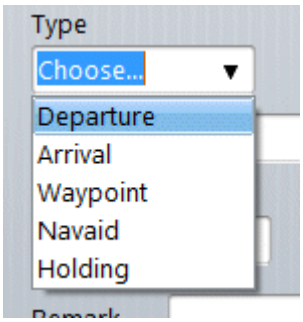
*An ARINC 424 RNAV database must be loaded into the scenario to allow Flight Plan definition.*
*Import RNAV using the TerrainBuilder or create your own database item by item.*

## • Building a Flight Plan

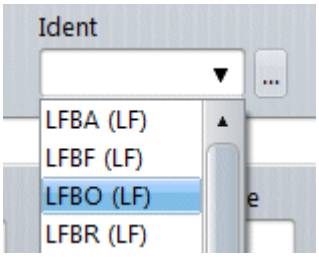
To define a Flight Plan, use the button on the tool bar (or use mouse right click, then **Add Flight Plan**)
Give the name to the Flight Plan then create it.


Double click the icon (or use mouse right click, then **Edit Flight Plan > Fplan**) to open the property window.

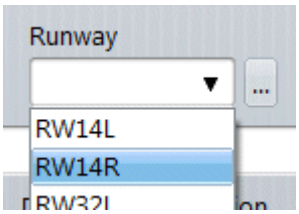
Add the first point which is the **departure**:



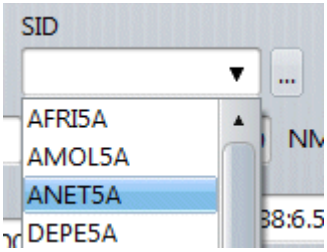

Choose then the **airport** (ICAO) ident:

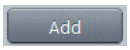



Then the **runway**:




and, if available, a standard **departure** procedure (SID):

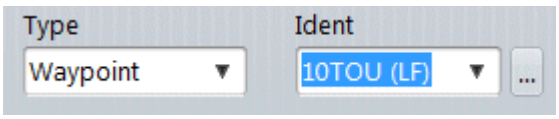



then press  button.

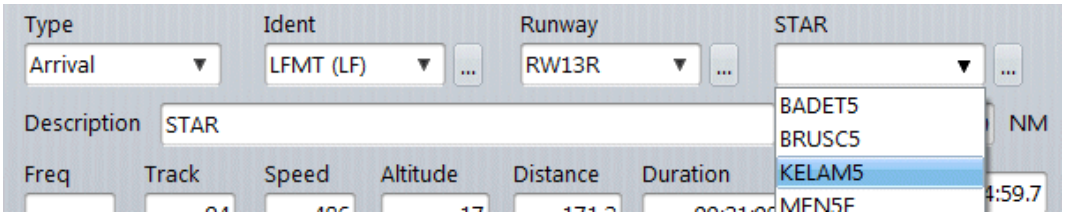

The Flight Plan will be filled accordingly:

| # | Ident | ICAO | Description | Freq | Lat | Lon | Term | Track | Speed | Distance | Altitude | ETA | Remark |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | LFBO | LF | Departure | --- | 43.64 | 1.37 | IF | 0 | 486 | --- | 499 | --- | |
| 2 | TS | LF | SID | 423 | 43.51 | 1.49 | CF | 137 | 0 | 10.5 | 4000 | 00:01:13 | |
| 3 | X094 | LF | SID | --- | 43.30 | 1.20 | TF | 235 | 0 | 21.5 | 4000 | 00:03:42 | |
| 4 | X217 | LF | SID | --- | 43.31 | 0.94 | TF | 272 | 0 | 15.3 | 4000 | 00:05:29 | |

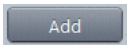
Now, add a **waypoint**:




Then the arrival **procedure**:




*If you cannot see the destination airport (here LFMT), increase the Filter value:*

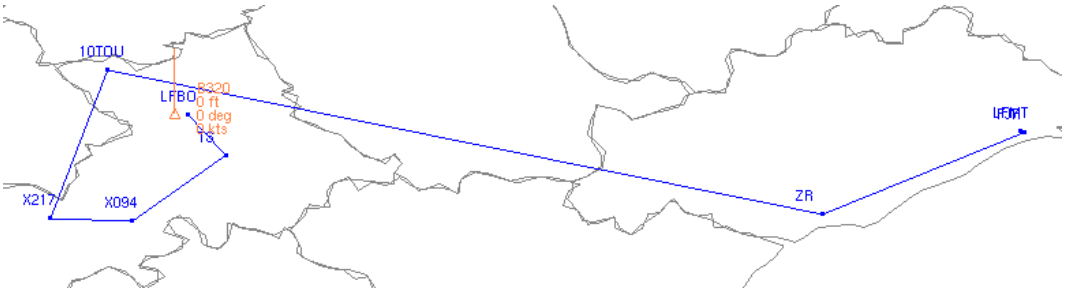

then press 

button.

The total flight plan will be the following one:

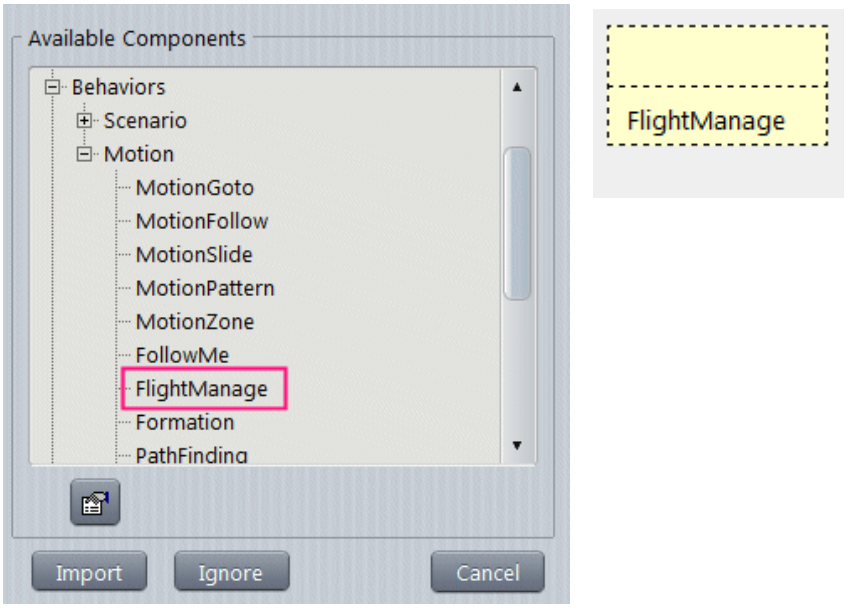
| # | Ident | ICAO | Description | Freq | Lat | Lon | Term | Track | Speed | Distance | Altitude | ETA | Remar |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | LFBO | LF | Departure | --- | 43.64 | 1.37 | IF | 0 | 486 | --- | 499 | --- | |
| 2 | TS | LF | SID | 423 | 43.51 | 1.49 | CF | 137 | 0 | 10.5 | 4000 | 00:01:13 | |
| 3 | X094 | LF | SID | --- | 43.30 | 1.20 | TF | 235 | 0 | 21.5 | 4000 | 00:03:42 | |
| 4 | X217 | LF | SID | --- | 43.31 | 0.94 | TF | 272 | 0 | 15.3 | 4000 | 00:05:29 | |
| 5 | 10TOU | LF | | --- | 43.77 | 1.12 | TF | 21 | 0 | 29.9 | 4000 | 00:08:56 | |
| 6 | ZR | LF | STAR | 1093.5 | 43.32 | 3.35 | TF | 101 | 0 | 136.6 | 0 | 00:24:48 | |
| 7 | FJR | LF | STAR | 114.4 | 43.58 | 3.97 | TF | 68 | 0 | 40.8 | 0 | 00:29:32 | |
| 8 | LFMT | LF | Arrival | --- | 43.58 | 3.96 | TF | 290 | 486 | 0.9 | 0 | 00:29:38 | |

Now, for each of the Flight Plan entries, modify the speed and the altitude (select the line, change in Edition then update)
The map will finally display the defined Flight Plan:

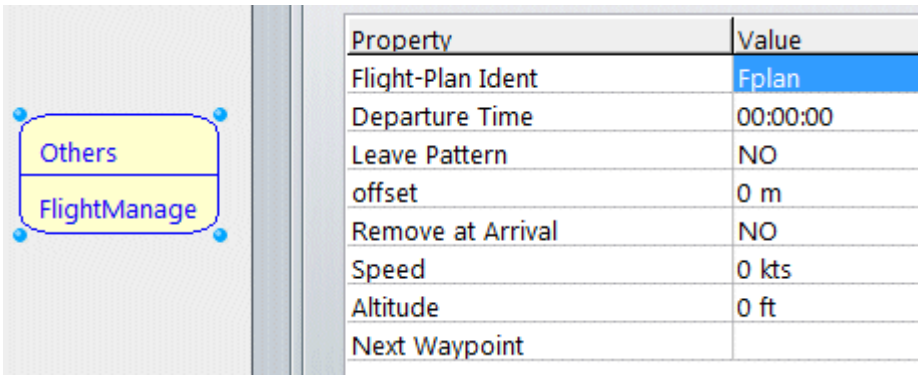



Now, it is necessary to give an aircraft a kind of FMS component and assign to it the new Flight Plan.
In the Models category, add the FlightManage Component:

### Flight-Plans




Select an entity on the map then give the FlightManage component in its Behavior. Open the component then, set the following parameters:



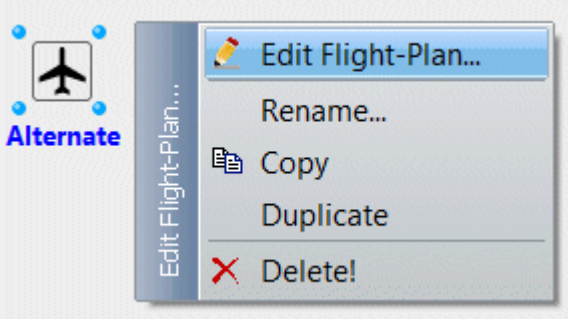

**Flight-Plan Ident**: select the Flight Plan to follow from the drop down list
**Departure Time**: give the simulation time at which the aircraft must depart
**Remove at Arrival**: If Yes, the aircraft (entity) will be automatically removed from the scenario.

*The FlightManage component will instruct the aircraft (entity) to follow the plan by sending control commands to the dynamic. If other controllers are also working (entity plan or trajectory following), the result will be undetermined.*

## • Popup Menu

Double click on the Flight Plan icon to display its property window.

When a Flight Plan is selected:




**Edit Flight Plan**: see Properties for details

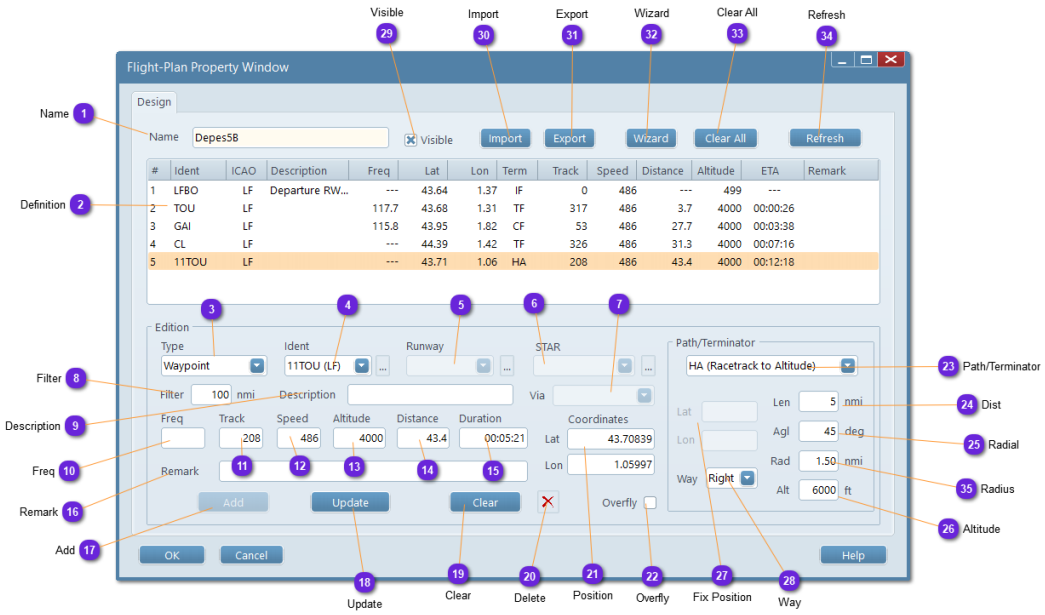
**Rename**: change the name of the current Flight Plan

**Copy**: copy the Flight Plan to the clipboard

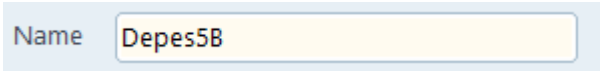
**Duplicate**: do a copy then paste it aside

**Delete**: remove the selected Flight Plan from the Database.

# Design




## 1 Name



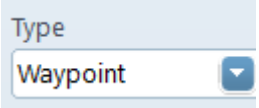

Name of the Flight Plan, must be unique in this database.

## 2 Definition

| # | Ident | ICAO | Description | Freq | Lat | Lon | Term | Track | Speed | Distance | Altitude | ETA | Remark |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | LFBO | LF | Departure RW... | --- | 43.64 | 1.37 | IF | 0 | 486 | --- | 499 | --- | |
| 2 | TOU | LF | | 117.7 | 43.68 | 1.31 | TF | 317 | 486 | 3.7 | 4000 | 00:00:26 | |
| 3 | GAI | LF | | 115.8 | 43.95 | 1.82 | CF | 53 | 486 | 27.7 | 4000 | 00:03:38 | |
| 4 | CL | LF | | --- | 44.39 | 1.42 | TF | 326 | 486 | 31.3 | 4000 | 00:07:16 | |

A Flight Plan (FP) is make as a list of entries (and then legs) from take off to landing.
All entries are referring to existing navigation elements (like Navaids) from the loaded ARINC 424 database.
Any of the Flight Plan line can be selected.
Path Edition box is the updated with the data of the selected line.
User can then modify most of the parameters (if they are not predefined) and **Update** the FP entry.

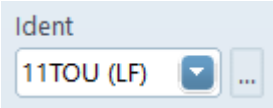
## 3 Type




List the kind of entries available to define the Flight Plan. Once selected, Ident (4) will display a list of matching devices extracted from the database:

Departure: list all airports of the database
Arrival: list all airports close to the last Flight Plan point
Navaid: list all radio navigation device close to the last point
Waypoint: list all waypoints close to the last point
Holding: list all holding patterns close to the last point

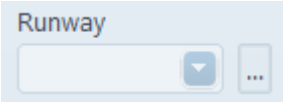
## 4 Ident




According to Type (3) selection, list here all devices that match the selection and the vicinity expressed in Filter (7).

*ICAO code is mentioned between parenthesis.*

## 5 Runway




When the Type is an Airport (departure or arrival), the Runway might be selected.

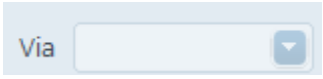
*According to the runway, corresponding SID/STAR/APPR is available and can be chosen in (6).*
*Once defined and bound with a SID or STAR, it will not be able to change it without removing the SID or STAR completely.*

## 6 Procedures


According to runway selection (but not necessarily), predefined flight procedures are listed here.
They are extracted from the ARINC 424 database.

*A procedure (departure or approach & landing) cannot be modified if it is extracted from the database.*
*In order to create your own procedure, do not use predefined ones.*

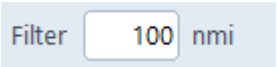
## 7 Via/Trans




Some Departure, Arrival and Approach procedures contains different leg groups with a common part. User must select the VIA point in order to extract a single flight path.
The list is automatically limited to plausible choices. The common part is not displayed.
Once added, user will see all the points pertaining to the selected SID, STAR or Approach procedure.

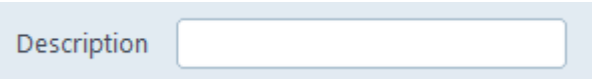
*A procedure (departure or approach & landing) cannot be modified if it is extracted from the database.*
*In order to create your own procedure, do not use predefined ones.*

## 8 Filter



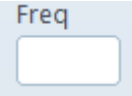

Specify here the radius for the search in order to reduce the list of devices listed in Ident (4).
Just increase the value to get more, when routes definitions have very long legs.
Unit is nautical miles.

## 9 Description



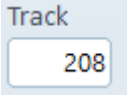

Add here any information on the FP entry.
Can be automatically filled for predefined procedures.

## 10 Freq



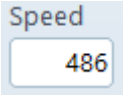

When the device selected is a radio emitter, this is the frequency in Hz or KHz. Informative only.

## 11 Track




Leg orientation in true heading, in degrees.

## 12 Speed

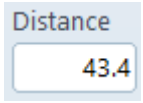



Speed that must be attained for the current leg until next change, in nautical miles.

## 13 Altitude

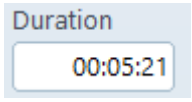

Altitude that must be attained at this point and kept. Normal rate of climb/ descent is used from the altitude change request to reach the specified altitude.
Unit is feet unless prefixed with FL (Flight Level) which express hundreds of feet (FL300 = 30000 feet).

## 14 Distance



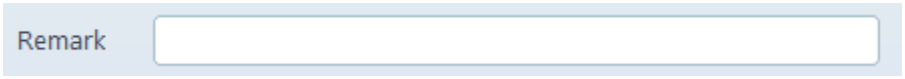

Distance of the leg, in nautical miles.

## 15 Duration



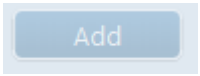

Estimation of the time requested to travel the leg.

## 16 Remark



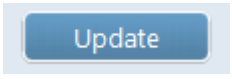

Value extracted from the ARINC database.

## 17 Add



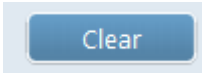

When enabled, allow to **add** (or **insert** if a line is selected in the Flight Plan) an entry into the Flight Plan.
See Building a Flight Plan here.

## 18 Update




When enabled, replace the FP entry (selected line) with the data in Edition box.

## 19 Clear

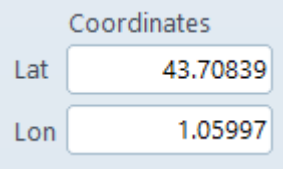



Clears all the data fields of the Edition box. Useful to define a FP entry from scratch.

## 20 Delete

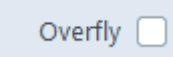
When enabled, delete the FP entry (selected line).
If disabled, the FP entry cannot be deleted. This is normally the case for predefined procedures.

## 21 Position




Actual Lat/Lon position of the device (airport, navigation aid, waypoint...).
Cannot be changed.

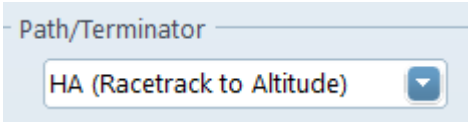
## 22 Overfly




When this option is checked, the turn will be initiated only after the aircraft will be vertical to the point and not before like for a normal navigation.


*This option is handled by the FMS manager.*
*Without this mode activated, the FMS makes the aircraft engage a turn before the point to avoid over shutting the trajectory.*

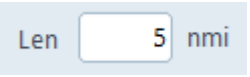
## 23 Path/Terminator




Path/Terminator definition define how a path has to be flown and terminates. According of the supported segment type (drop down list), the FMS (component) will fly the aircraft accordingly.
IF, TF and CF are the most used segment types used.

- IF (Initial Fix)
- TF (Track to a Fix)
- RF (Constant Radius Arc)
- CF (Course to Fix)
- FC (Fix to Distance)
- FD (Fix to a DME distance)
- CR (Course to Radial)
- VR (Heading to Radial)
- PI (Procedure Turn)
- HA (Racetrack to Altitude)
- HF (Racetrack to Fix)

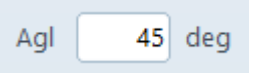
## 24 Dist




When enabled, specify the maximum distance (in NM) for the procedure.

*Enabled only for RF, FC, FD, HA and HF.*

## 25 Radial




When enabled, specify the radial to be crossed (or followed) in degrees.

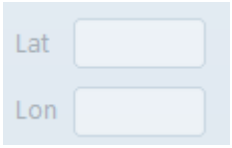
*Enabled only for CR, VR, HA and HF.*

## 26 Altitude


When enabled, specify the altitude to be reached (or kept) in feet or FL.

*Enabled only for HA*

## 27 Fix Position




When enabled, specify the position of the arc center of the path procedure.

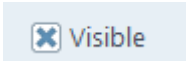
*Enabled only for RF*

## 28 Way


When enabled, specify how the racetrack must be followed from the entry fix.
Left (turn) at entry or right (turn) at entry.

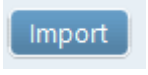
*Enabled only for HA and HF.*
*Standardized entry and exit procedure are followed by the FMS.*

## 29 Visible



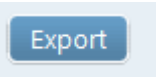

When checked, the Flight Plan will be always drawn on the map (and highlighted when selected).

## 30 Import



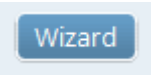

Use this button to replace the current Flight Plan with a previously exported Data Model in /Data/Shared/Models.
All current content will be lost.

## 31 Export




Use this button to save the current Flight Plan for later import. Default location is /Data/Shared/Models.

## 32 Wizard

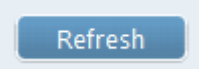



Use this button to automatically compute a vertical profile for the Flight Plan. When building a FP from scratch, it can be faster to join the SID and the STAR with a coherent flight.

## 33 Clear All

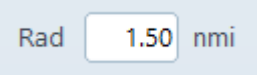

Empty the FP from all its entries.

## 34 Refresh




Compute again all speeds, distances and ETA of the Flight Plan.

## 35 Radius



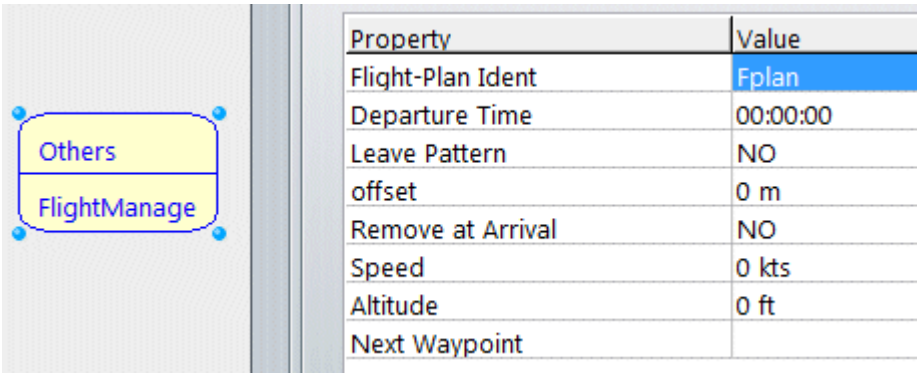

When enabled, specify the radius of the arc when flying RF or the race track radius when flying a racetrack (in nautical miles).

# Runtime

During runtime, Flight Plan can be monitored and changed. It is the only way to change the course of an aircraft in a managed mode.

## • How to Use

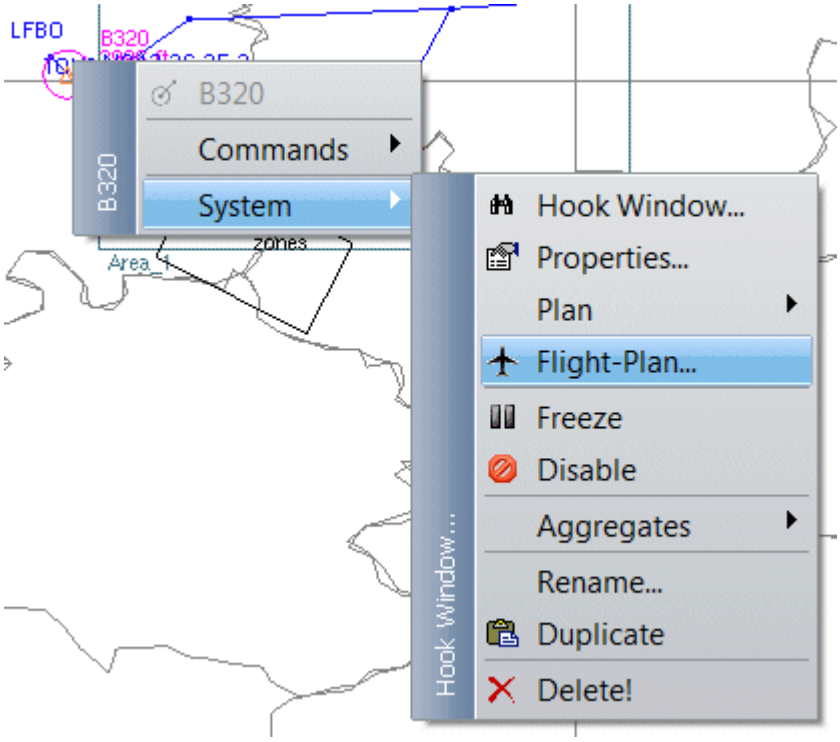
During the flight, select the aircraft on the map, select then the FlightManage component (in its behavior pane) and double click on it.
On the displayed value interface, change the following value, then press **Send** button.



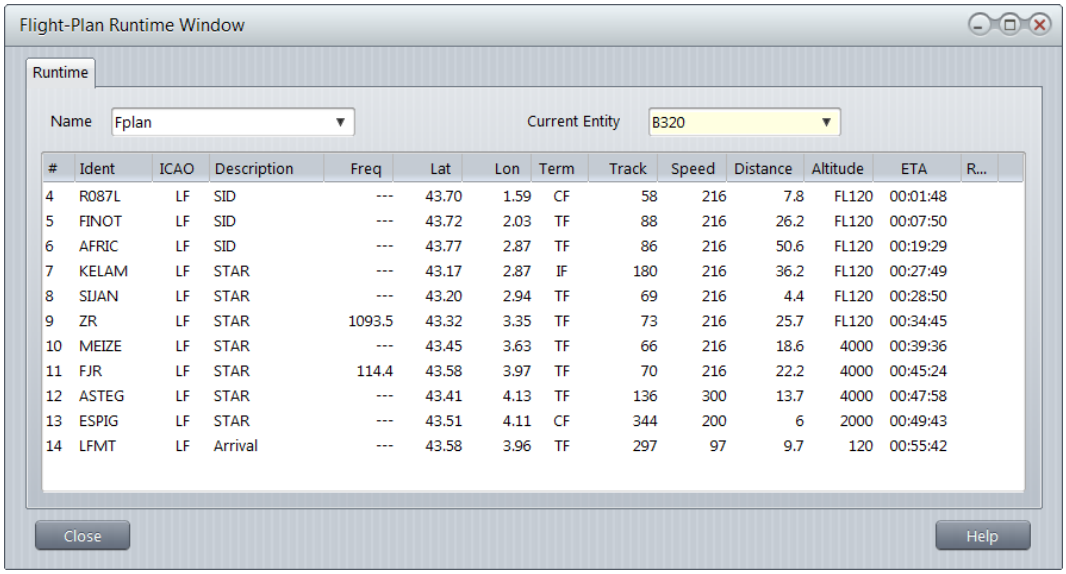

Leave Pattern: runtime command that force the aircraft, if cycling into an holding pattern, to manually leave it and continue
Offset: distance in meters aside from the Flight Plan line. Positive values stand to right offset; negative for left.
Speed: speed offset from the flight plan defined one (this value is for runtime)
Altitude: altitude offset from the flight plan defined one one (this value is for runtime)

To display the Flight Plan and monitor it, select the aircraft, right click and select **System::Flight-Plan**:

## Runtime




The runtime window lists the remaining flight plan with continuous updated data like Distance and ETA




| # | Ident | ICAO | Description | Freq | Lat | Lon | Term | Track | Speed | Distance | Altitude | ETA | R... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | R087L | LF | SID | --- | 43.70 | 1.59 | CF | 58 | 216 | 7.8 | FL120 | 00:01:48 | |
| 5 | FINOT | LF | SID | --- | 43.72 | 2.03 | TF | 88 | 216 | 26.2 | FL120 | 00:07:50 | |
| 6 | AFRIC | LF | SID | --- | 43.77 | 2.87 | TF | 86 | 216 | 50.6 | FL120 | 00:19:29 | |
| 7 | KELAM | LF | STAR | --- | 43.17 | 2.87 | IF | 180 | 216 | 36.2 | FL120 | 00:27:49 | |
| 8 | SIJAN | LF | STAR | --- | 43.20 | 2.94 | TF | 69 | 216 | 4.4 | FL120 | 00:28:50 | |
| 9 | ZR | LF | STAR | 1093.5 | 43.32 | 3.35 | TF | 73 | 216 | 25.7 | FL120 | 00:34:45 | |
| 10 | MEIZE | LF | STAR | --- | 43.45 | 3.63 | TF | 66 | 216 | 18.6 | 4000 | 00:39:36 | |
| 11 | FJR | LF | STAR | 114.4 | 43.58 | 3.97 | TF | 70 | 216 | 22.2 | 4000 | 00:45:24 | |
| 12 | ASTEG | LF | STAR | --- | 43.41 | 4.13 | TF | 136 | 300 | 13.7 | 4000 | 00:47:58 | |
| 13 | ESPIG | LF | STAR | --- | 43.51 | 4.11 | CF | 344 | 200 | 6 | 2000 | 00:49:43 | |
| 14 | LFMT | LF | Arrival | --- | 43.58 | 3.96 | TF | 297 | 97 | 9.7 | 120 | 00:55:42 | |


*A Flight Plan cannot be changed during runtime.*

# Orbits

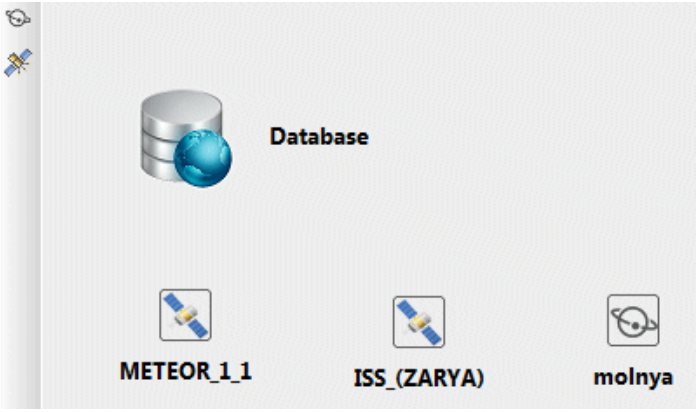
vsTASKER provides orbits definition and propagators to expand the simulation environment to space objects.
The feature is commonly used for communication between satellites or with ground site.
It is does not address specific missions like launching satellite or changing its attitude.



*Apart from Earth entities that are clamped to the ground or surface or even move inside the atmosphere, orbiting objects like satellites are not impacted with the rotation of the Earth.*
*Thus, the ECI reference system is used for these objects, with Lat/Lon/Alt conversion methods that take into account the date (Julian) of the simulation time.*
*For this reason, when playing with orbiting objects, date and time settings matter.*

## • Environment

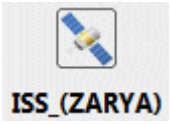
In this category, Database lists all available orbits and all instanciated ones.




Click on Database icon to create orbits.

vsTASKER supports two kind of orbits:

### Orbits




TLE orbits, represented by the Two Line Elements description from NASA or NORAD. These formatted line accurately specify all 6 Kepler parameters and information for the propagator (SGP4) to spin the satellite on the orbit over time. Typically, TLE orbits are real one, constantly updated by ground tracker devices.
Example of a TLE:

0 INMARSAT 2-F1

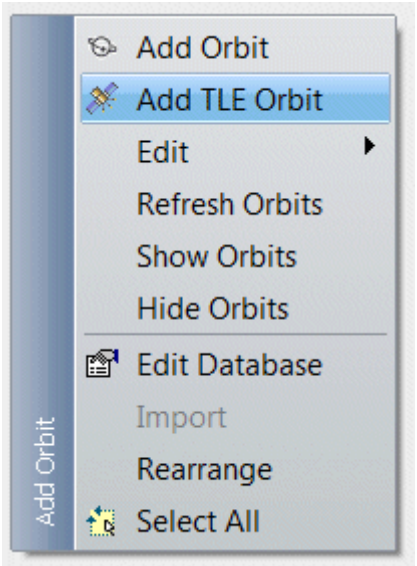
1 20918U 90093A 14260.01177448 -.00000013 00000-0 00000+0 0 586

2 20918 010.2265 036.7076 0004148 208.3310 155.1961 00.98876047 83829


Keplerian orbits, that allow the user to define and change the orbital elements in order to create a specific orbit for study (or to approach a known one). Keplerian orbits can be used to modify the satellite course over time, by changing one of the 6 parameters. The propagator (Two-Body or J2) works well with such orbits.


*In vsTASKER, orbit and satellite are separated although in the TLE dataset, the satellite makes the orbit and both are combined. With vsTASKER, several satellite can follow the same orbit with a different anomaly.*

## • Popup Menu




**Add Orbit**: create a Keplerian orbit.

**Add TLE Orbit**: create a TLE orbit

**Edit**: edit a current orbit from the drop down list

**Refresh Orbits**: rebuild all orbits based on the new Julian date

**Show Orbits**: display all orbits (and not only the selected one) on the map and globe

**Hide Orbits**: show all orbits (except the selected one) on the map and globe

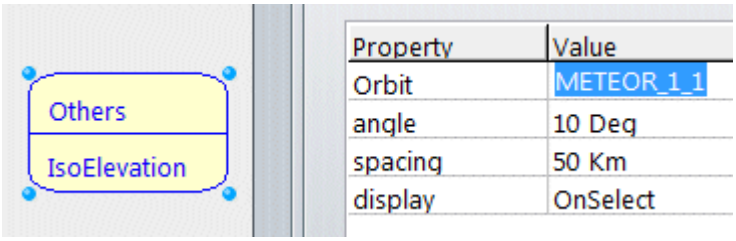
**Edit Database**: popup the database property window.

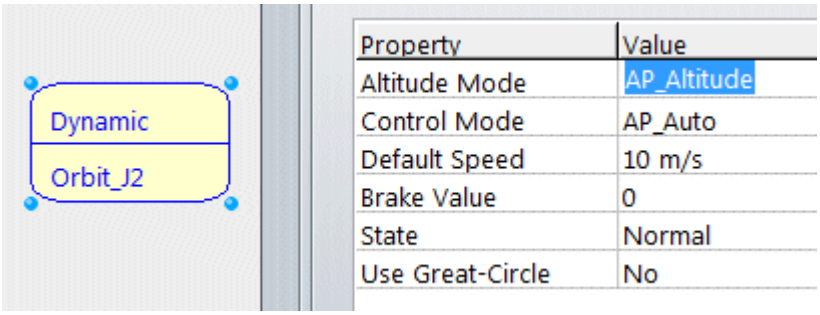
**Import**: list all exported orbits in /Data/Shared (if any)

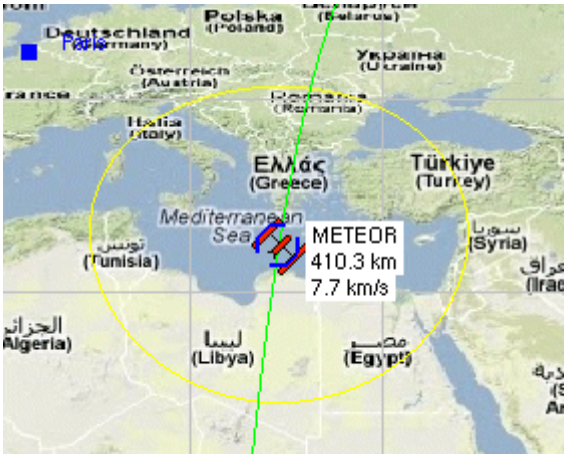
**Rearrange**: reorganize all the orbits icons on the diagram display

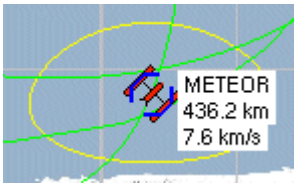
**Select All**: select all orbit icons

## • How to Use

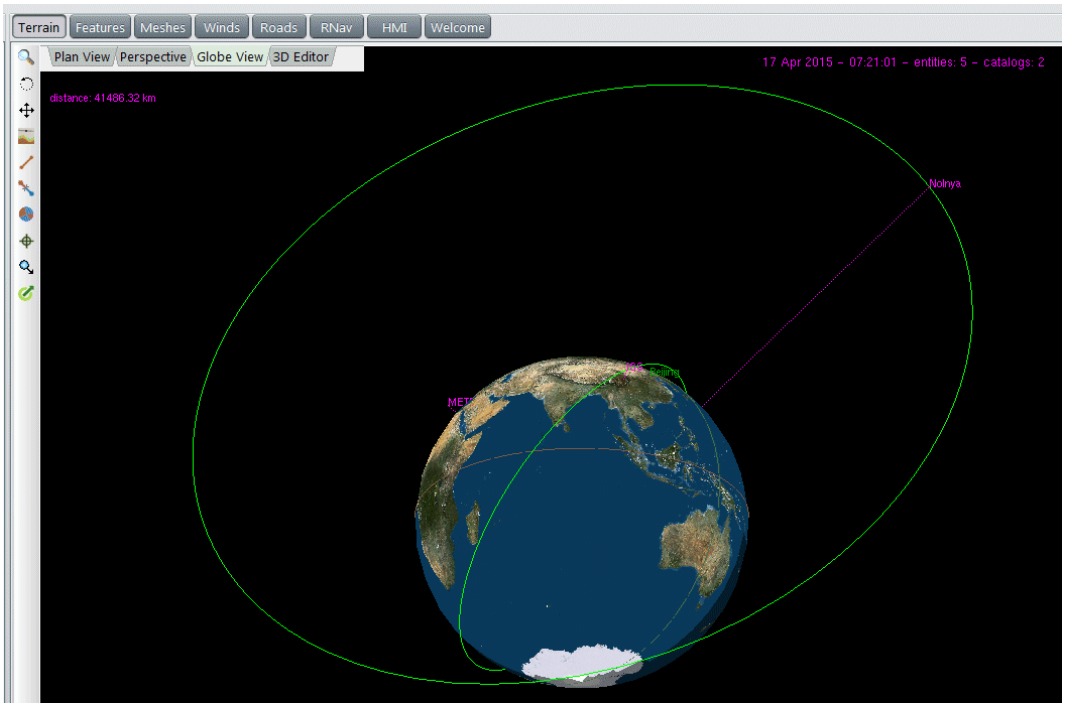


## Orbits


Terrain
Features
Meshes
Winds
Roads
RNav
HMI
Welcome
Plan View
Perspective
Globe View
3D Editor
17 Apr 2015 - 07:21:01 - entities: 5 - catalogs: 2
distance: 41486.32 km
Nolnya

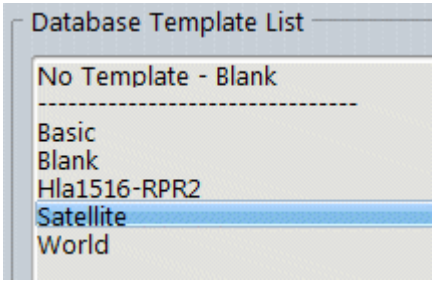
# Space Database

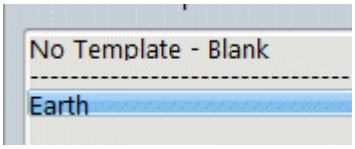
Use this property window to import files from /data/satellites directory of vsTASKER. These satellite database files contain TLE information provided by Celestrack website (http://www.celestrak.com/NORAD/elements/)
Some of these satellites are decayed and thus, SGP4 propagator will reject them.

Nevertheless, the Two-Body (or J2) propagator will be able to revive them based on the orbital elements.

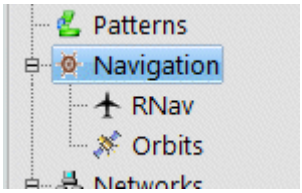

## • How to Use

Create a new Database.
Select the Template Satellite, then Earth.



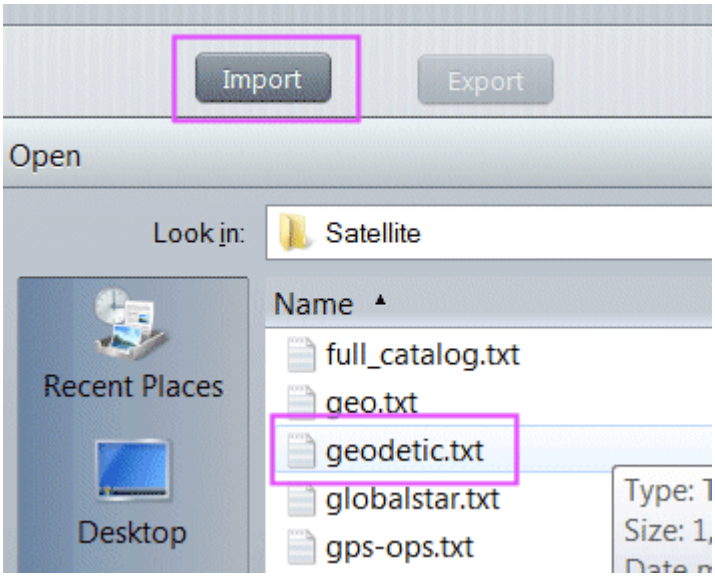

Go to Navigation category, then click on **Orbits** icon, then **Database** icon and finally **Import** to select the TLE database to load.



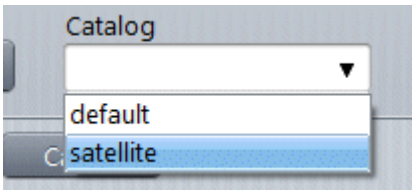



In the list, select the orbits you want to instantiate locally (for ie: geodetic.txt)

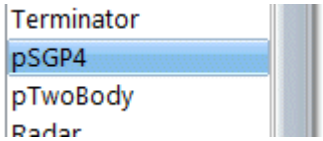
### Space Database




Select the Satellite entity defaulted in the Catalog (or any other you have created)



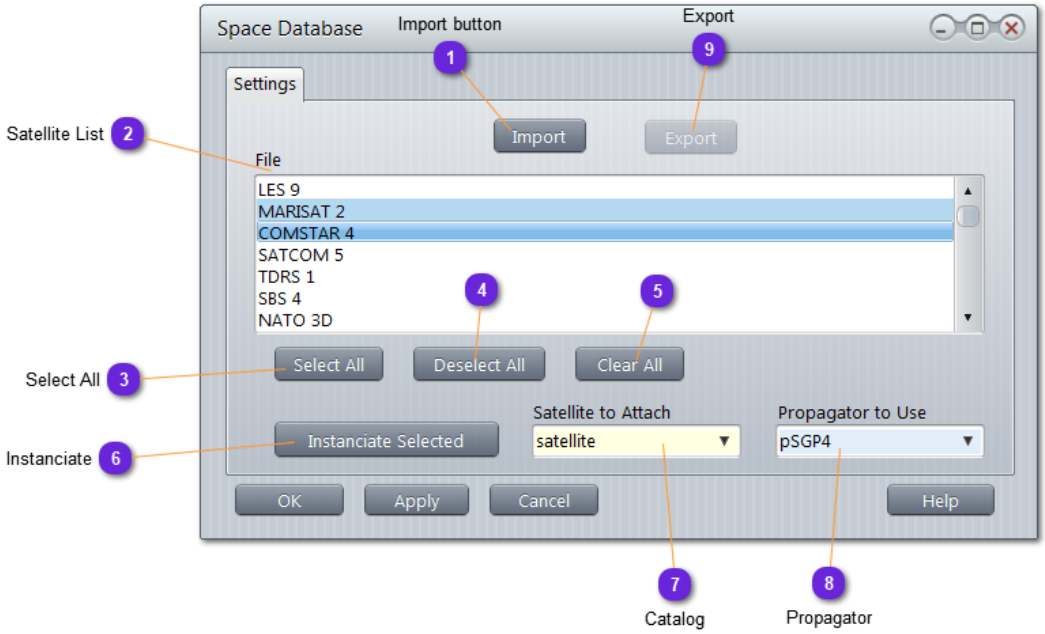

Select the Propagator component you will use. It must be SGP4 for any TLE orbit, Two-Body for Keplerian orbits



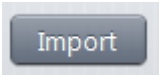

Click on the **Instanciate** button.

vsTASKER will create the orbits and will associate a new satellite Entity.

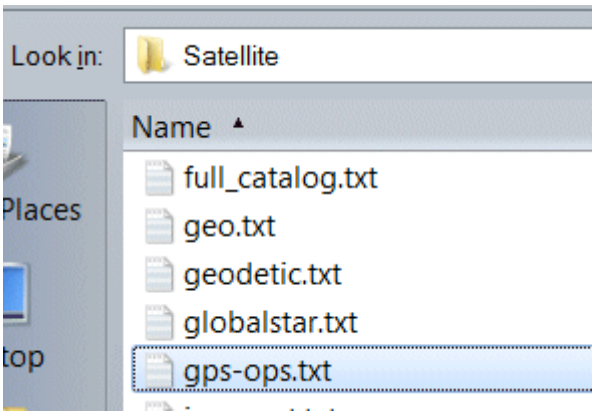
# Space Database




## 1 Import button

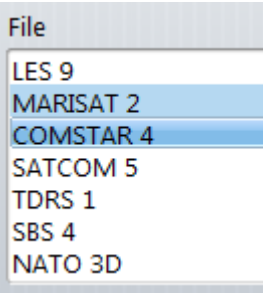



First, use this button to import a TLE definition file located in /Data/Satellites (or anywhere else)



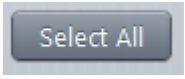

These files are NORAD TLE formatted entries.
They can be extracted from the celestrack website or manually added from any other source.
Files must list all TLE including lines 0-1-2 for each satellite (orbit) definition.

## 2 Satellite List




List all satellites from the loaded (imported) database file.
This list is selectable (multiple select using the Ctrl key, continuous with the Shift key)

## 3 Select All

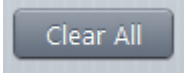



Select all satellites in the above list.

## 4 Deselect All

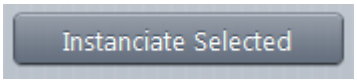

Deselect all selected satellites in the above list.

## 5 Clear All



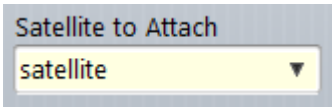

Remove all imported satellites from the list.
Do this before loading another database. If not, newly imported ones will be added.

## 6 Instanciate



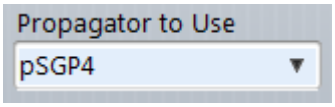

When depressed, all selected satellites (2) will be instantiated as TLE orbits. If set, associated satellite entity will be created along with the proper propagator.

## 7 Catalog



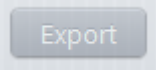

Specific here from the drop down list which entity (preferably a satellite) will be dropped on each instantiated orbits.

## 8 Propagator




Specify here from the drop down list which propagator (supporting TLE) will be added for each instanciated satellite.

## 9 Export



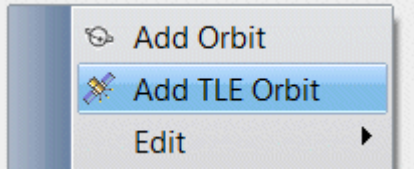

*Not available yet*

# TLE Orbit

A two line element set (TLE) is a data format encoding a list of orbital elements of an Earth orbiting object for a given point in time, the epoch.
Using suitable prediction formula, the state (position and velocity) at any point in the past or future can be estimated to some accuracy.
You can create a TLE orbit from the database import and instanciation (see here) or by creating directly the orbit, adding the TLE data and attaching the satellite.

## • Manual Setting

First, create the TLE orbit




Then go to the TLE tab and enter the two lines of formatted data.
ie:

0 INMARSAT 2-F1

1 20918U 90093A 14260.01177448 -.00000013 00000-0 00000+0 0 586

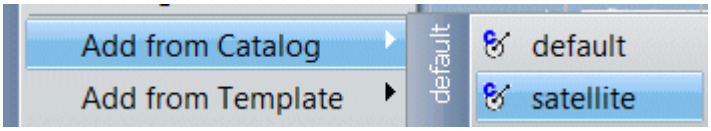
2 20918 010.2265 036.7076 0004148 208.3310 155.1961 00.98876047 83829

Then click on **Update Orbital Element** button.

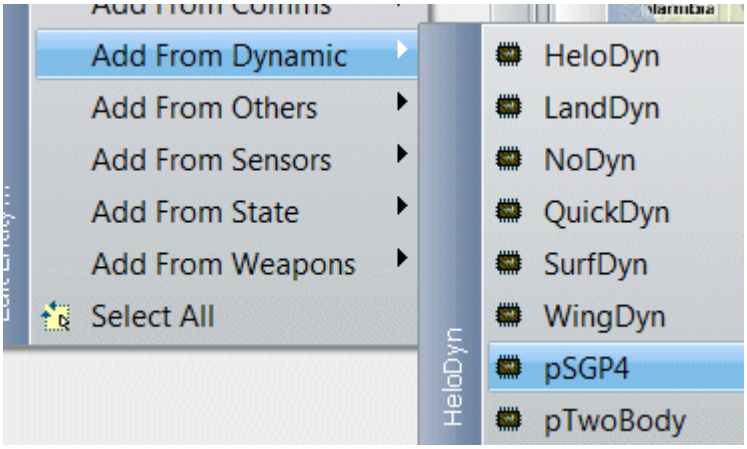

*The only way to rename a TLE orbit is to change the first line of the TLE. 0 is optional.*
*If you only give lines 1 and 2, the name will be kept unchanged.*

Now, if you click on the newly created orbit, you will see drawn in magenta on the map (or on the globe)

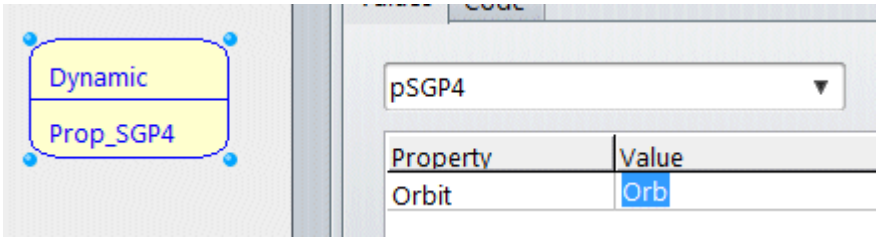
On the map, add one satellite:



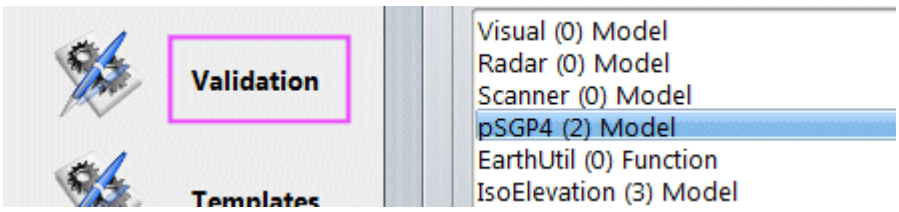

Then in the Model panel, add the SGP4 propagator (dynamic)




Open the Component and attach it to the created orbit (above):



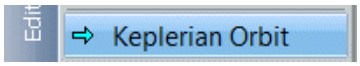

Make sure that the Validation function of the propagator is activated:



# Keplerian Orbit

You can create a Keplerian orbit manually before attaching the satellite.

*A TLE orbit can be converted into a Keplerian orbit in order to allow manual update of the shape of the orbit. The TLE data will be lost.*

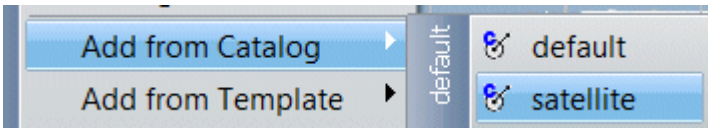



*Select a TLE orbit icon, right click and use* ***-> Keplerian Orbit*** *menu option to replace it. Cannot undo.*

## • Manual Setting

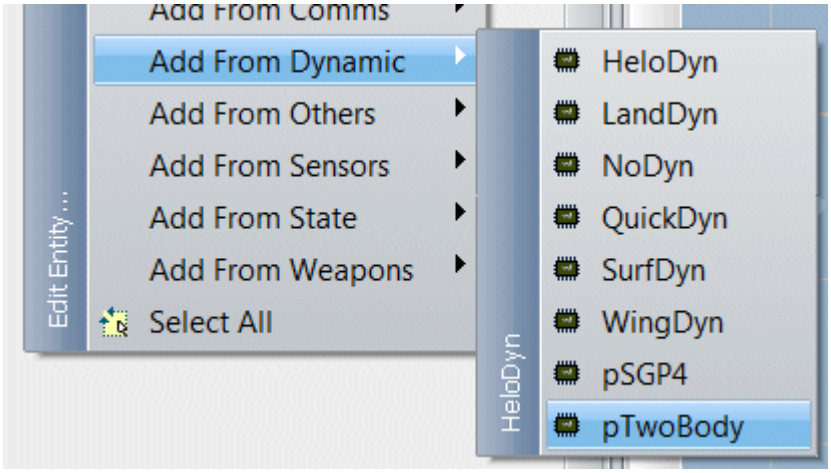
First, create the Keplerian orbit and set the Orbital Elements. Once created, the Orbit must be used by (at least) one satellite entity.

On the map, add one:

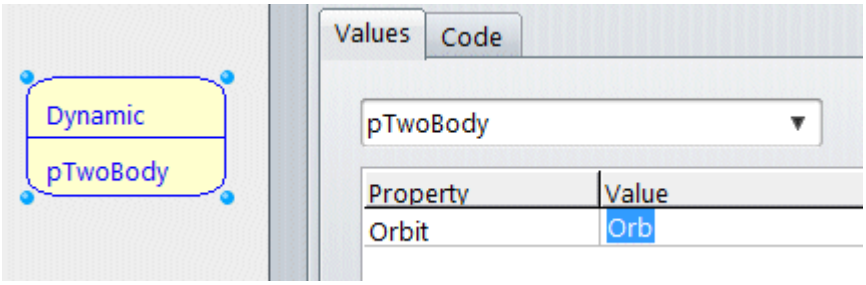




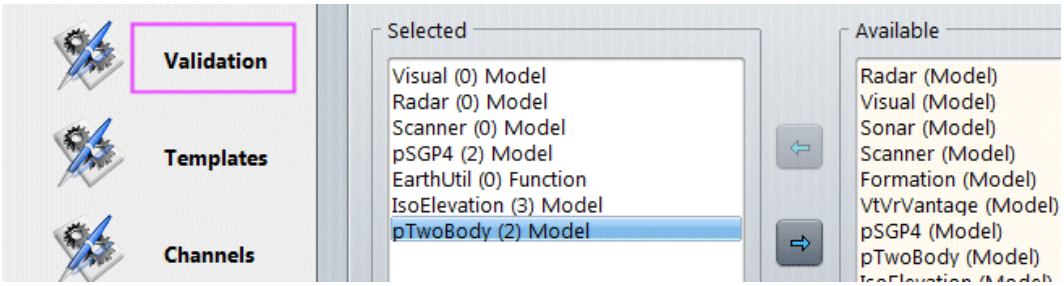
Then in the Model panel, add the Two-Body propagator (dynamic)




Open the Component and attach it to the created orbit (above):




Make sure that the Validation function of the propagator is activated:




Now, clicking on the orbit will display it. The satellite will be clamped on the orbit and changing the Julian date will move the satellite along the orbit, same as during the simulation.

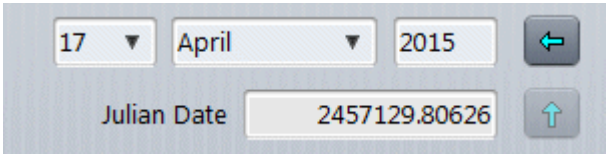


*When you create an orbit, the Epoch time is not the scenario time but can be any time. It is important to set this Epoch time in order to synchronize satellites between simulators.*
*See here for more info.*

# Julian Date

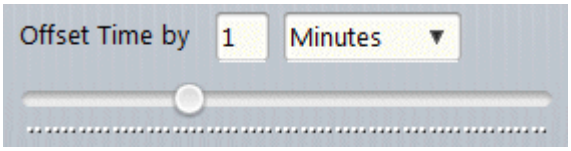
Orbital propagators (SGP4 or J2) are using Julian date for their computation.
It is then important to set the correct date for your simulation.

This is done in the Scenario property window:



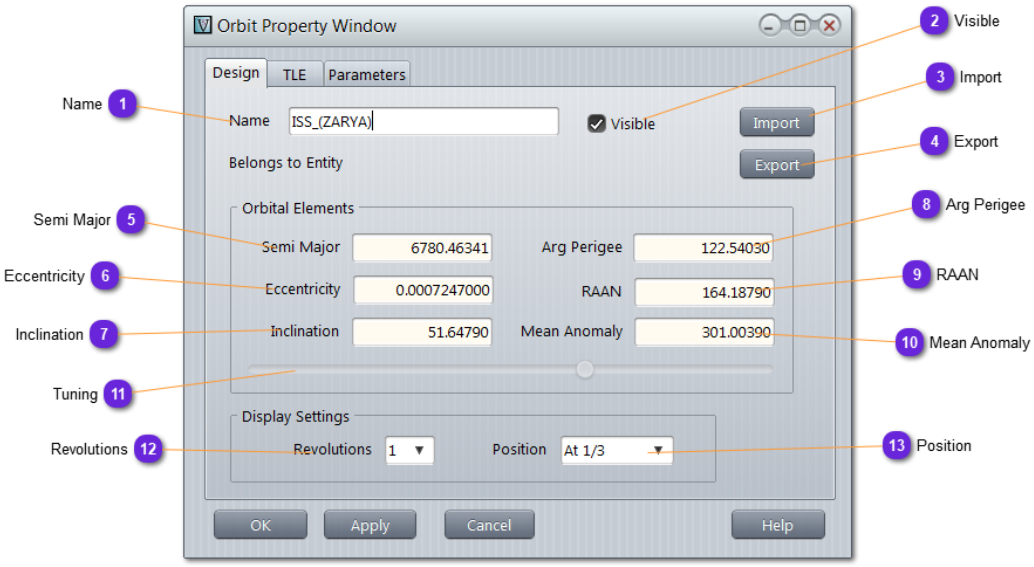

The Julian Date can be directly entered into the text field to update the current date and time using button.
Using Apply on the window, the satellite positions on the map are automatically updated (also on the globe).

Julian Date can be changed using a slider where the step and the unit can be specified. Small changed (seconds) to big jumps (days) can be used to move the satellite in the future (or past) for a specific position (over the earth) or configuration (between several satellites on different orbits), without running the simulation (at this stage, it is only maths)




If the simulation must run in real-time synchronized with the wall clock (to have the exact position of actual satellites), set the Wall Clock option and use the UTC option also.

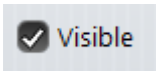
# Orbit Properties




## 1 Name

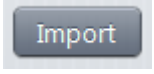

Name of the orbit, but typically, name of the Satellite that will follow the orbit. Cannot be changed for TLE orbit (change the first line and update to do so).

## 2 Visible



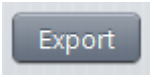

If checked, the orbit will always be displayed on the map and globe.
If unchecked, will be displayed only when selected (or attached satellite selected).

## 3 Import



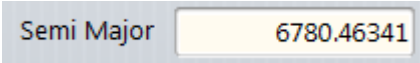

Use to replace the current orbit with an exported one stored in /Data/Shared

## 4 Export




Store the current orbit into a file in /Data/Shared for exchange or later reuse.

## 5 Semi Major

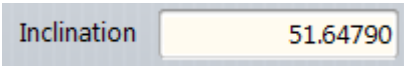



It is the radius of an orbit at the orbit's two most distant points.
For the special case of a circle, the semi major axis is the radius. One can think of the semi major axis as an ellipse's long radius.

## 6 Eccentricity


A measure of how elongated the orbit is.
A circle has an eccentricity of 0.
An ellipse has eccentricity between 0 and 1.
A parabola has an eccentricity of 1.
A hyperbola has an eccentricity greater than 1.

## 7 Inclination




The angle between this orbital plane and a reference plane.

## 8 Arg Perigee


Argument of Perigee. The angle from the ascending node to the pericenter (perigee), measured in the plane of the orbit

## 9 RAAN


Right Ascension of the Longitude Node. Angle from the origin of longitude of the reference plane to the orbit's ascending node: the point in its orbit where the orbiting body crosses the reference plane going "upward" or "northward". The time of this crossing is often used as the "elements' epoch"

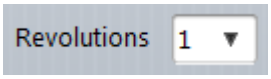
## 10 Mean Anomaly


The location of the body in the orbit: the product of mean motion and time since pericenter passage.
(ie., the mean anomaly is 0.0 when the orbiting body is at pericenter)

## 11 Tuning

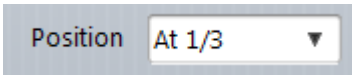
Select with the mouse on of the six orbital elements and move the slider to change the value.
Immediate update on the map and the globe.
Can only be done for Keplerian orbits.

## 12 Revolutions



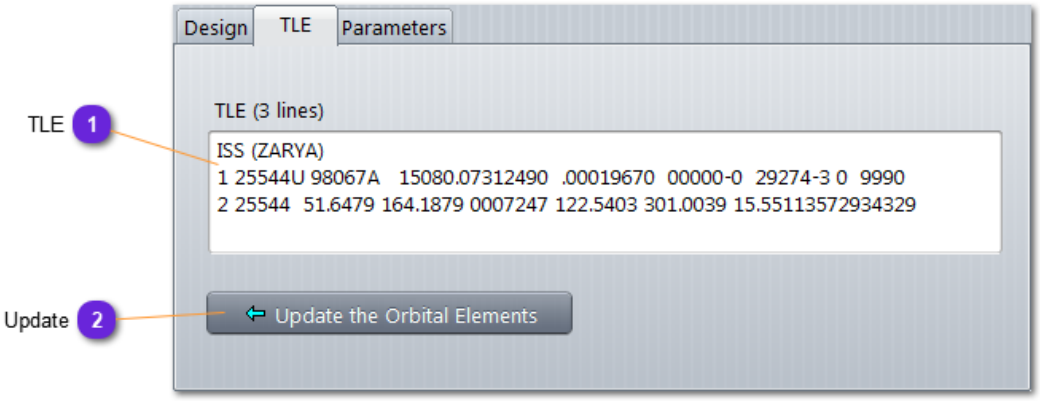

Select how many earth revolutions the orbit must be computed and displayed.

## 13 Position

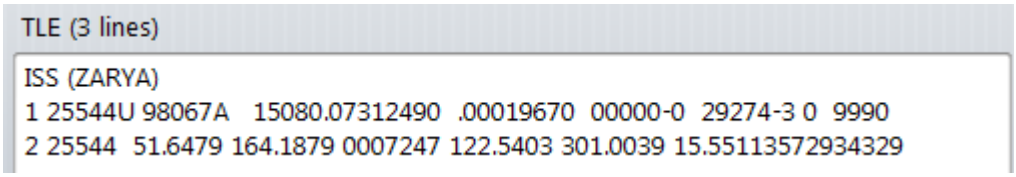



Specify where should be located the satellite, at current Epoch, regarding the number of revolutions computed and displayed of the orbit.
Centered: orbit path will be equal future and past Epoch.
At 1/3: before Epoch and 2/3 after Epoch
At Start: orbit will be displayed from Epoch
At End: orbit will be displayed until Epoch.

# TLE



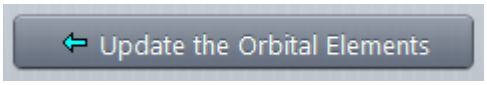

## 1 TLE




Copy paste here the TLE file as it is provided from reliable sources and Internet.
Definition can be get here: https://en.wikipedia.org/wiki/Two-line_element_set
Public data can be extracted here: http://www.celestrak.com/NORAD/elements/
and: http://www.tle.info/joomla/index.php

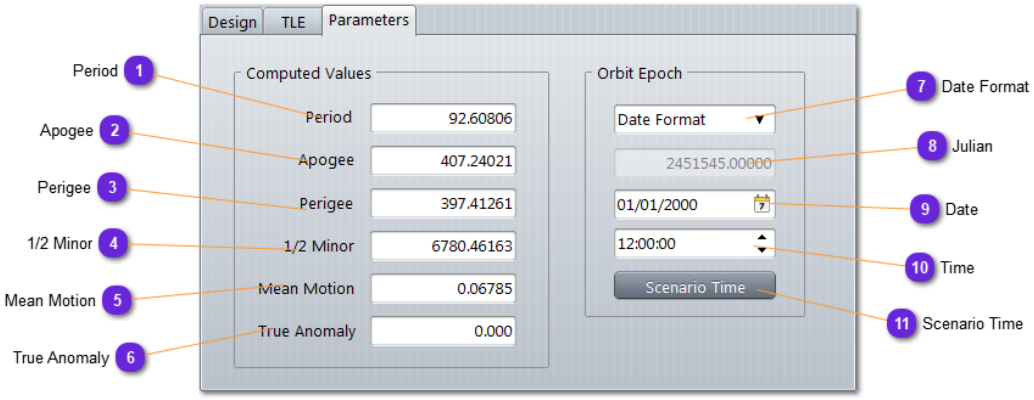
*The first line does not need the leading 0. It describes the orbit (satellite) name.*
*Use cam change it in order to rename the TLE satellite.*

## 2 Update




Use this button to feed the parameters and the propagator with the two line data, or the name if this one has been changed.
Must be used any time a new TLE is pasted here to update the old one.

# Parameters




## 1 Period


The time to complete one orbit.

```
P = 1/n = 1/ Mean Motion
```

More generally: `P = sqrt( K a^3 )`
where `a` is the semi major axis of the orbit, and `K` is inversely proportional to the sum of the masses of the central body and the orbiting body.

## 2 Apogee


The location of the greatest distance between the orbiting body and the central body when the orbit is an ellipse.
The apocenter is diametrically opposite the pericenter on the major axis of the orbit.

## 3 Perigee


The shortest distance between the center of the orbiting body and the center of the orbited body.

```
q = a * (1 - e)
```

where `q` is the pericenter distance
`a` is the length of the semi major axis and
`e` is the eccentricity of the orbit.

## 4 1/2 Minor


The semi minor axis of an ellipse runs from the center of the ellipse (a point halfway between and on the line running between the foci) to the edge of the ellipse.
The semi minor axis is half of the minor axis.

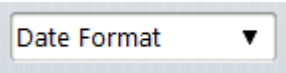
## 5 Mean Motion


Average angular velocity needed to complete one orbit.

## 6 True Anomaly


True anomaly is an angular parameter that defines the position of a body moving along a Keplerian orbit. It is the angle between the direction of periapsis and the current position of the body, as seen from the main focus of the ellipse (the point around which the object orbits).

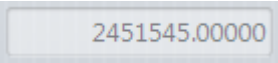
## 7 Date Format




Select here the format to define the orbit Epoch (which might not be the scenario Epoch).

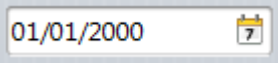
- Julian Format: use the Julian number for the date
- Date Format: use the DD/MM/YY HH:MM:SS for the date.

## 8 Julian




When Julian Format is selected, enter here the Julian date number of the Epoch, then apply. You can then check the corresponding Date and Time that are computed.

## 9 Date

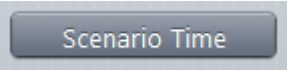



When Date Format is selected, enter here the DD/MM/YY date of the Epoch.

## 10 Time


When Date Format is selected, enter here the HH:MM:SS time of the Epoch.

## 11 Scenario Time




Click this button to use the current scenario date as the orbit Epoch.

# Distribution

vsTASKER can distribute the simulation during runtime, or connect to other system using various standard protocols.
The paradigm is to provide capabilities to the simulation environment without mingling these capabilities into it.

The same simulation, for example, can do at the same time HLA and LAN, or DIS and 4586 and CIGI.
They are all separated and all can work with the same entities or scenario environment.



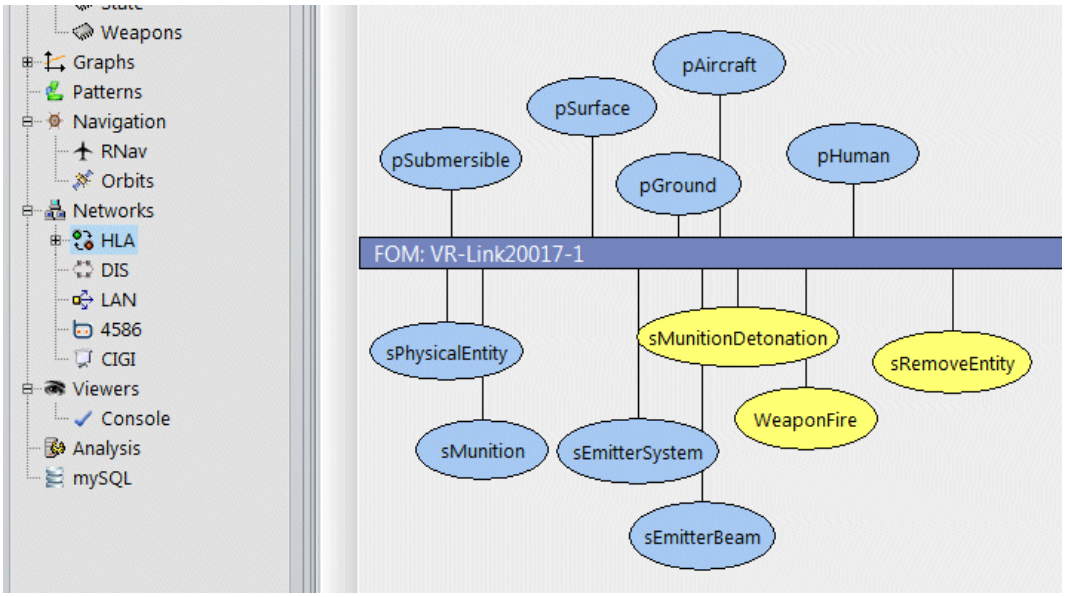
*HLA and DIS are mutually exclusive.*
*All others can be combined.*

# HLA

vsTASKER can connect to several RTI maker and is able to generate specific code according to the target RTI and the version chosen.
Going HLA or connecting to an existing Federation becomes straightforward as diagrams and other abstractions are used to cover the difficulty of RTI programming.
With the OMT builder, user can even define from scratch their own Federation.


It is also possible to use vsTASKER simulation engine as an external model (imported as a Component, a DLL or an external library) to go HLA.
The same way, vsTASKER can be used as a bridge from LAN only capable system to HLA, drastically reducing the development time.


*HLA is an add-on in vsTASKER. The simulation engine and structure is not designed for HLA. Thus, one should not expect the full coverage of all services and RTI capabilities, neither a out-of-the-box support of some HLA federations (like RPR-FOM).*

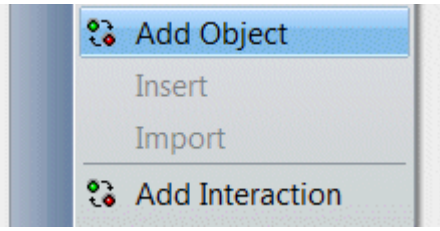



## • Environment

Add a Object FedItem

 Add an Interaction FedItem

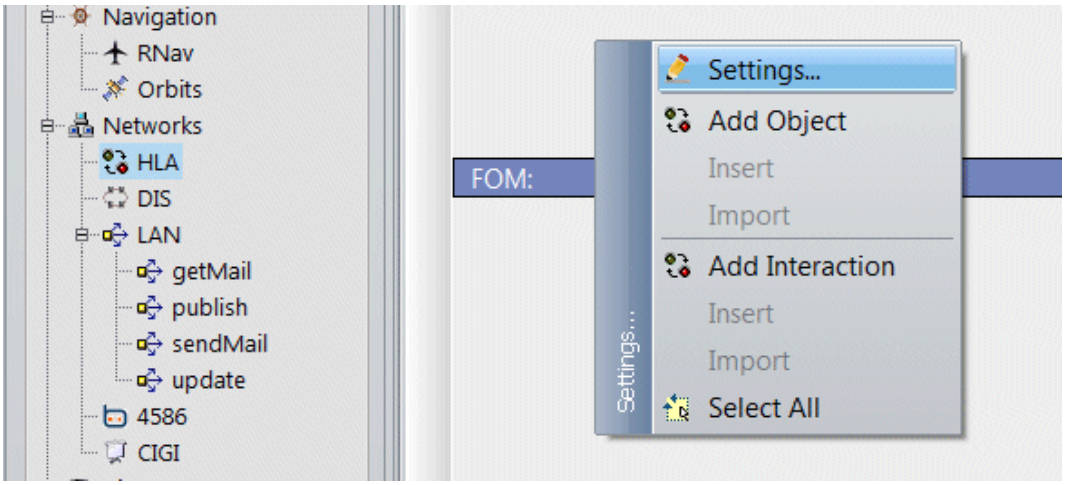
Click on the toolbar icon and drop the object into the Publisher or Subscriber zones. Right click on the background diagram bring a popup menu to do the same thing.



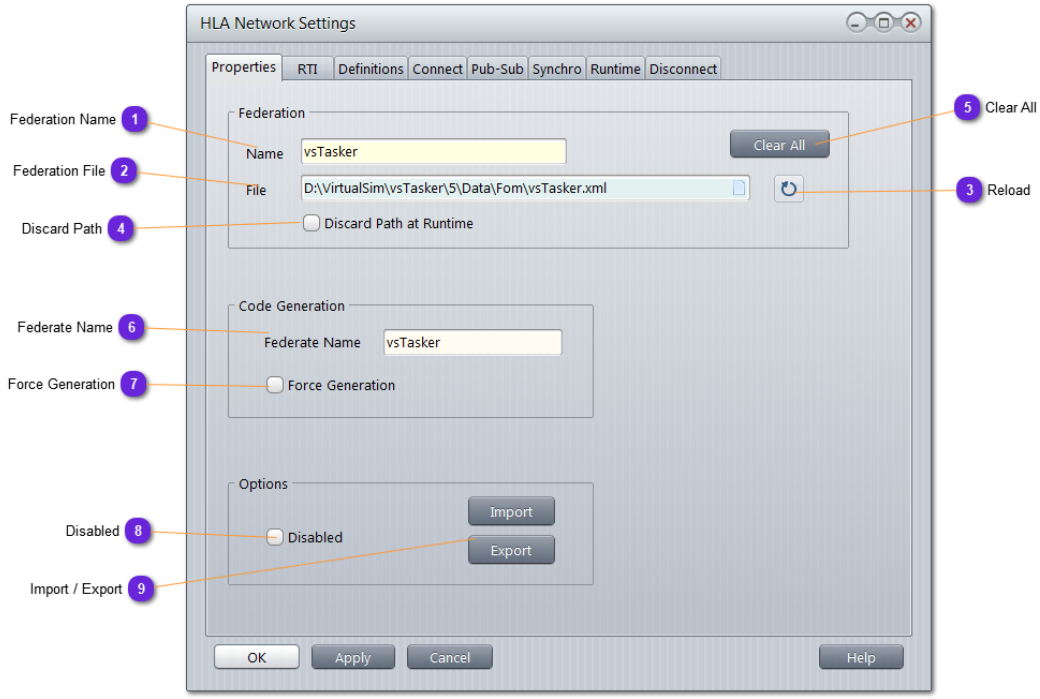

Refer to the **Tutorial document** to learn how to develop your first HLA simulation.

# Settings

General setting of the HLA feature is done a the general window.
You can access it from the HLA diagram using the contextual Settings menu, or double clicking the background or even the HLA symbol in Environment



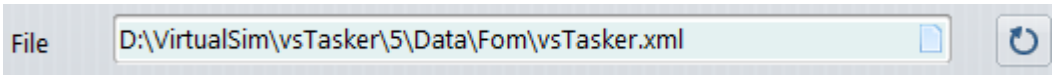
# Properties




## 1 Federation Name


Use this field to change the name of the Federation.
It is normally extracted from the loaded Federation File.
vsTASKER will try to join/create the Federation named according to this string.

## 2 Federation File




Use  to import a Federation file located by default in /data/Fom directory. The file might be .xml (1516) or .fed (for 1.3)

Use  to refresh all OMT and FedItems objects with the content of the current file. This can be useful inside an update cycle:

1 - Add/Remove/Update an Object and the OMT level (Object, Interaction, Data definition, ...)
2- Export the OMT definition as an FDD file.
3- Refresh the SOM part (FedItems) with the new data from the file.

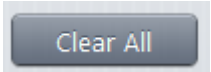
## 3 Reload

Reload the FDD and rebuild all OMT counterpart data structures.
This does not change any FedItem definitions.
In case of Federation change (or update), manual changes must be done by user on the SOM part.

## 4 Discard Path

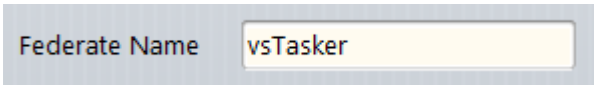

When this option is selected, the FDD filename only will be given to the rtiAmbbasador to create the federation execution.
This can be useful when the FDD will be available on the same directory as the executable (standalone exe or not).

## 5 Clear All



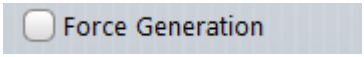

Use this button to remove all vsTASKER HLA definitions, including all OMT data and FedItems.
This action does not remove Data Models generated by the OMT converter.
Also, Federation files are **not** removed from the disk.

## 6 Federate Name



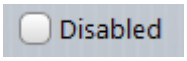

By default, when no name is provided, vsTASKER will use the label vsTasker + a uniq ID.
If something is specified in this field, vsTASKER will use this string and will append an ID to make it uniq on the Federation.

## 7 Force Generation




By default, vsTASKER does not generate any HLA code when no FedItem is defined (or when all have been disabled).
When this checkbox is ticked, HLA code will be generated.
An RTI must be specified, even if no FDD is loaded.

## 8 Disabled

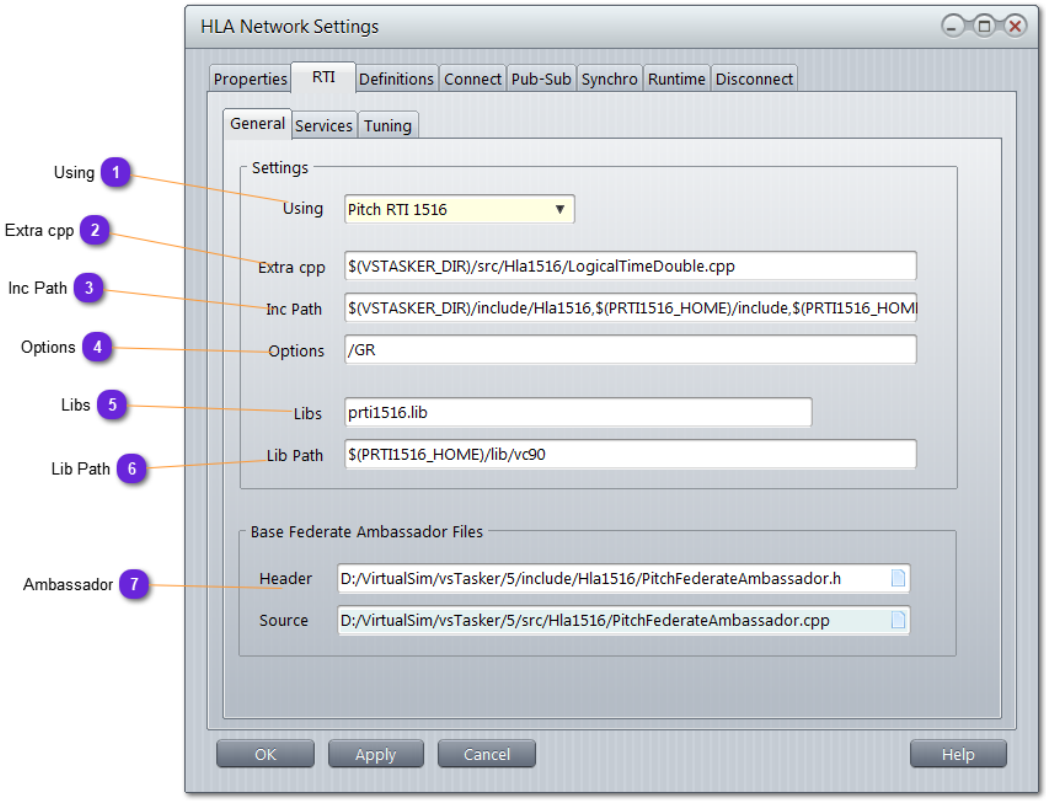



When this checkbox is ticked, the HLA code is not generated whatever the FedItem definitions are.
Also, RTI will not be joined in any way.

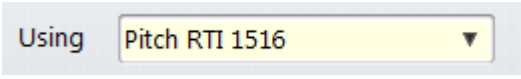
## 9 Import / Export


Export or Import to a file all FedItems definitions (SOM).
This can be useful to exchange between databases, users (or versions) the full HLA environment.
When an HLA environment has been setup for a specific Federation, saving it for reuse with another database.

# RTI




## 1 Using

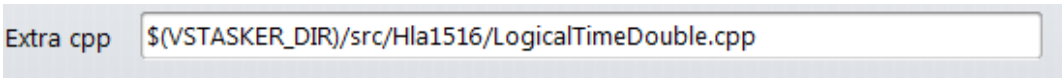



vsTASKER can generate code for various RTI.
Depending on the RTI you have licensed and intend to use, select it into the list.

Supported RTI are the following:

- **Pitch RTI 1.3**
- **Pitch RTI 1516**
- **Pitch RTI Evoled (1516e)**
- **KD RTI**
- **Mak RTI 1.3**
- **Mak RTI 1516**

## 2 Extra cpp

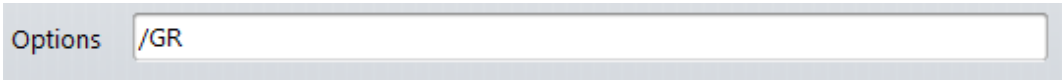



If you need to add some extra modules that must be compiled with the generated files, it is a good place to specified them here and each module separated with a comma.
When the complete path is not used, it must be specified in the next field (Inc Path).

## 3 Inc Path


By default and according to the RTI chosen, vsTASKER will fill set the field with predefined values.
Check that you have the environment variables defined.
All paths must be separated with commas.

## 4 Options




Here stand the compilation options (and mostly defines) requested by the RTI modules mainly.

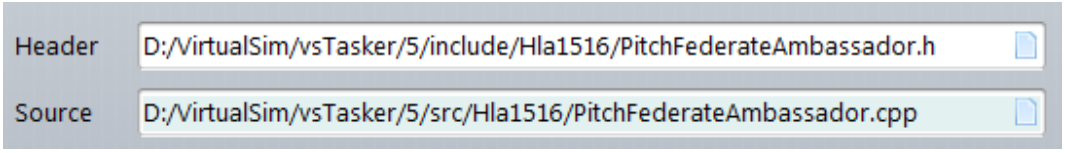
## 5 Libs


List here all necessary libraries requested to link the application.

## 6 Lib Path

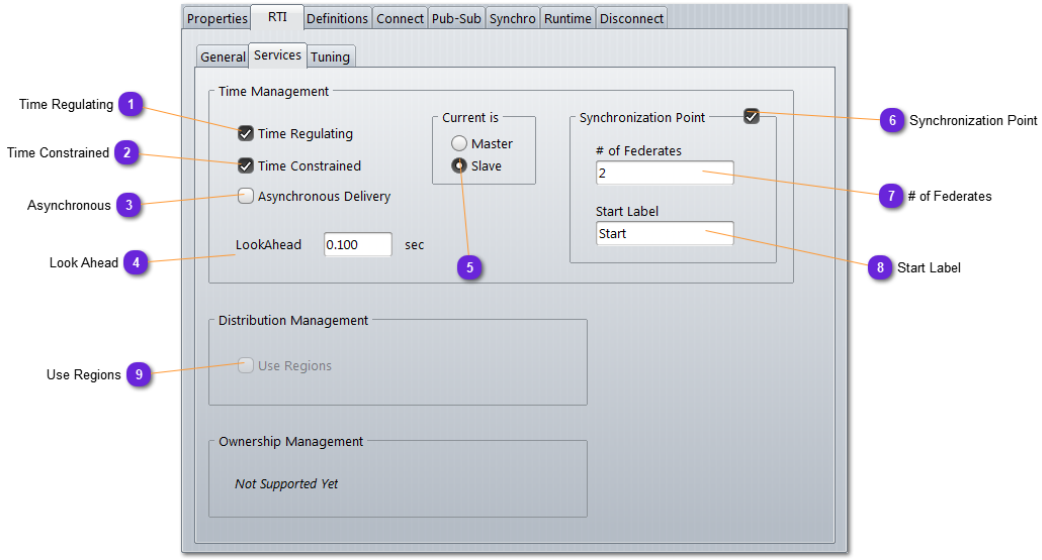

This field specifies the library path for the libraries listed above.

# 7 Ambassador

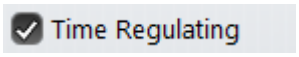



vsTASKER will generate a `vsTaskerAmbassador` class that will inherit from `BaseFederateAmbassador` defined in these files.
They are normally belonging to each RTI manufacturer and are compliant with the standard.
However, vsTASKER is using locally copied files that the user can replace with the latest ones from the manufacturer of their API.

# Services




## 1 Time Regulating



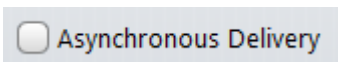

Select this option when the Federate must give time stamp and time advance signal

.

## 2 Time Constrained

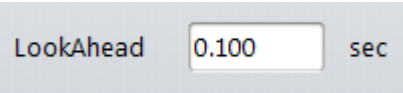

Select this option when the Federate must synchronize with another Time regulating Federate.

## 3 Asynchronous




Select this option to force asynchronous delivery for time stamped publishes when time regulating or constrained option is selected.

## 4 Look Ahead




Specify the look ahead time in seconds for time regulating or constrained federates.

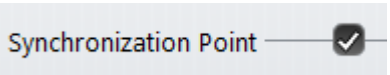
## 5 Current is


Set the Simulation Engine as **Master** or **Slave** when the Synchronization Point is ON.

The **Master** will call the following function:
`int vsTaskerAmbassador::callForSyncPoint(Start Label);` that will register a synchronization point and will wait for all slave federates to achieve the synchronization point before starting the simulation.

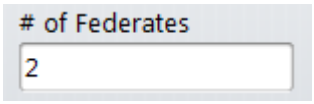
The **Slave** will call the following function:
`int vsTaskerAmbassador::waitForSyncPoint(Start Label);` that will achieve the synchronization point.

## 6 Synchronization Point




When this checkbox is ticked, vsTASKER will generate synchronization point mechanism, based on the Start Label string and the number of federates so that all federates will start at the same time. The count of the number of federates is done by the master, based on the discovery of specific synchronization points each slave federates generate. These synchronization points are named `_slave_xxxx` where `xxxx` is a random number between 1000 and 9999. This mechanism is automatically generated by vsTASKER. If you need to add in the federation another federate (not generated by vsTASKER) to be counted in the group, it must register such point using `registerFederationSynchronizationPoint()` `call`.
The master federate must be started first.

## 7 # of Federates

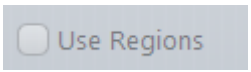



Number of federates that the Master must wait to reach the Synchronization, including itself.

## 8 Start Label

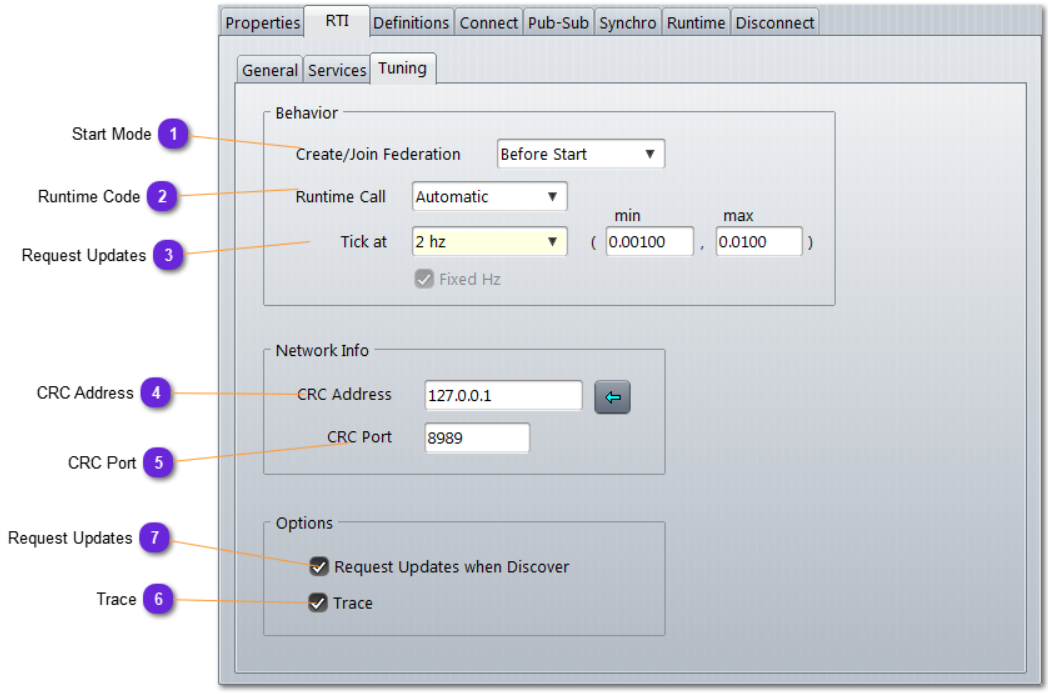

Name of the Synchronization Point to use for vsTASKER synchronization, in case the default one (Start) is already used.

## 9 Use Regions

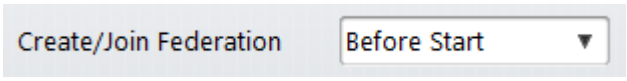



*Not supported yet.*

# Tuning



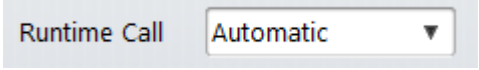

## 1 Start Mode




Used to specify when the Federation shall be joined or created.

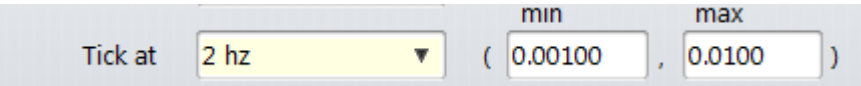
Before Start: Federation will be created or joined then, simulation engine will wait for the button to be pressed.
After Start: As soon as the button is pressed, Federation will be created or joined.
Manual: Federation as to be created or join from the user code, typically inside the Connect panel of the User Code.

## 2 Runtime Code




When **Automatic**, time synchronization (including also Point) is handled by vsTASKER.
When **Manual**, user code will be processed, typically into the Synchro panel of the User Code.

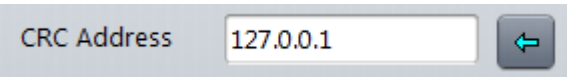
## 3 Request Updates




When this option is selected, vsTASKER will generate the code to force Object publishers to resend all Attribute data to the current Federate, for any discovered Object.
This is very useful when a Federate joins a Federation during runtime and must get the latest data from published Objects.

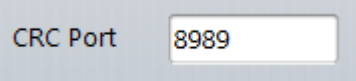
If Fixed Hz is checked, the frequency selected is the wall clock frequency and not the simulation frequency. It remains steady even if the simulation is accelerated or deccelerated.

## 4 CRC Address



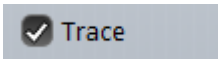

IP address of the RTI server (or client/proxy).

## 5 CRC Port



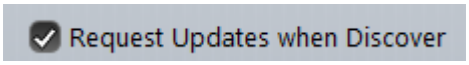

TCP/UDP port used by the RTI server (or client/proxy).
8989 is the default port used by default by RTI providers.

## 6 Trace




Select this option for verbose mode during runtime.
More trace will be visible on the console window and on the output file (sim_out.txt)

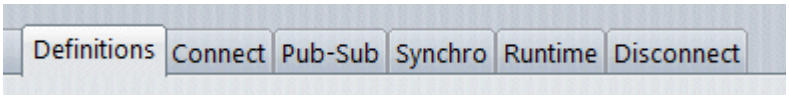
## 7 Request Updates




Force all FedItems to request update for all discovered objects.

# User Code

These different source code panes let the user pinpoint his own code inside various part of the RTI generated code.
You must use with caution as the result might really impact the final simulation.




You can retrieve all the generated code in the below functions located into the file [database name]_code.cpp

## • Definitions

Add here your FedAmbassador `#include, #define` and `#typedef`.

It will be inserted in the beginning of file [database name]_code.h

## • Connect

Add here your FedAmbassador user definitions.

It will be inserted in the file [database name]_code.cpp
under `int vsTaskerAmbassador::usrConnect()`

## • Pub-Sub

Add here all FedAmbassador user methods.
Do not forget to use: Rti::...

It will be inserted in the file [database name]_code.cpp
under `int vsTaskerAmbassador::usrPublishSubscribe()`

## • Synchro

Add here the FedAmbassador user code at connect time.

It will be inserted in the file [database name]_code.cpp

under `int vsTaskerAmbassador::usrSynchronize()`

## • Runtime

Add here the FedAmbassador user runtime code.
`Event* event` is available.

It will be inserted in the file [database name]_code.cpp
under `int vsTaskerAmbassador::usrRuntime()`

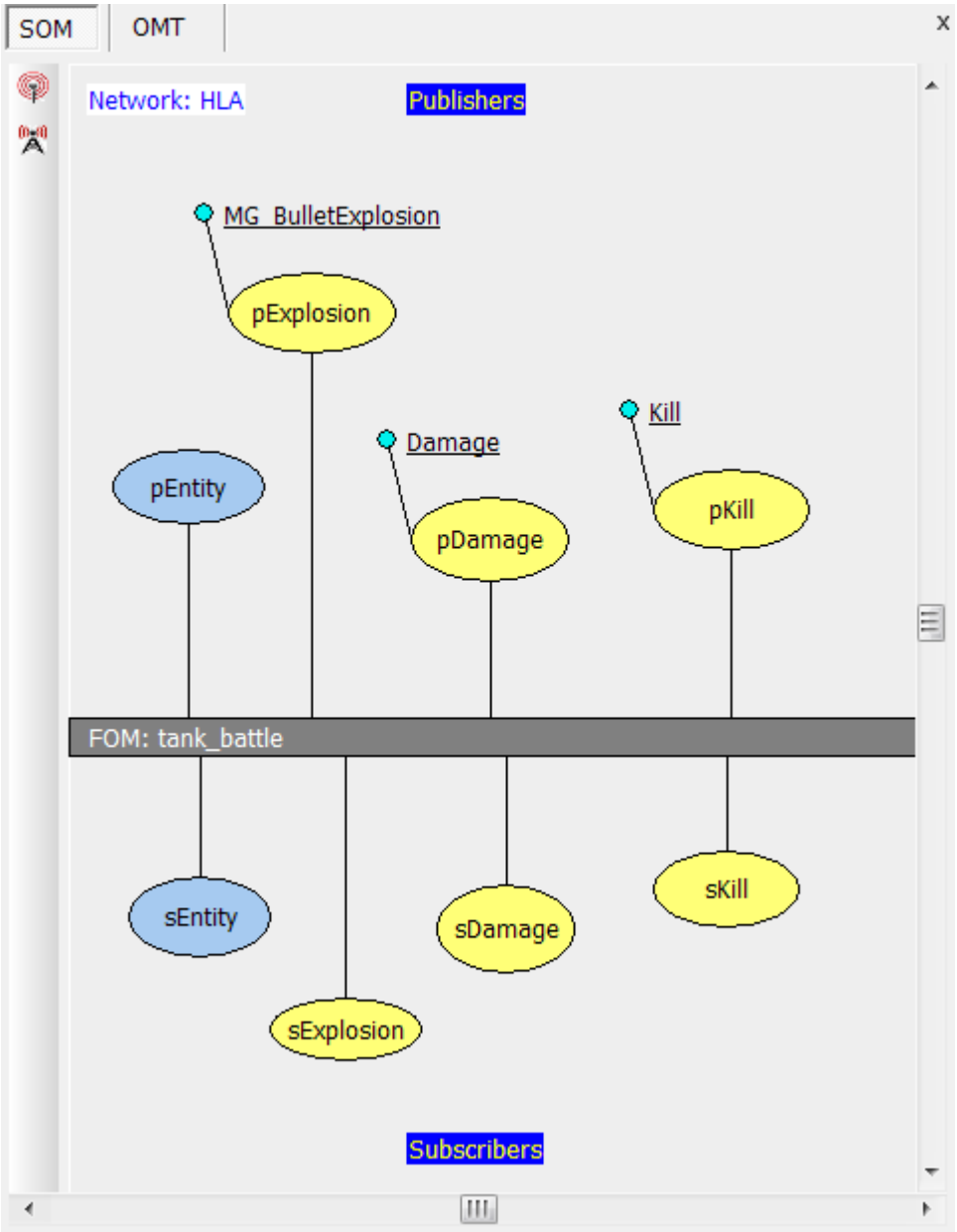
## • Disconnect

Add here the FedAmbassador user runtime code.
`Event* event` is available.

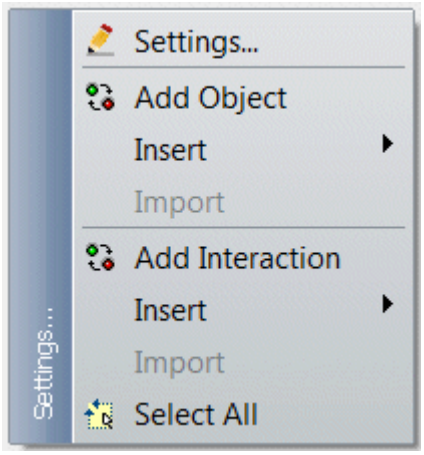
It will be inserted in the file [database name]_code.cpp
under `int vsTaskerAmbassador::usrDisconnect()`

# Federate Item

This window is used to setup the federate object (as a Publisher or Subscriber) and the way it is going to act in the Federation.
Even is vsTASKER instance will be seen as a unique Federate in the simulation, internally, many Federate [objects] will run as publishers or subscribers.




- **Popup Menu**




**Settings**: call the HLA property window

**Add Object**: create a FedItem object in publishing or subscribing mode according to the position of the mouse on the diagram (**top** for Publisher and **bottom** for Subscriber)

**Insert**: list all objects imported from the Federation file or the OMT definition

**Import**: list all exported FedItem objects in Data/Shared directory

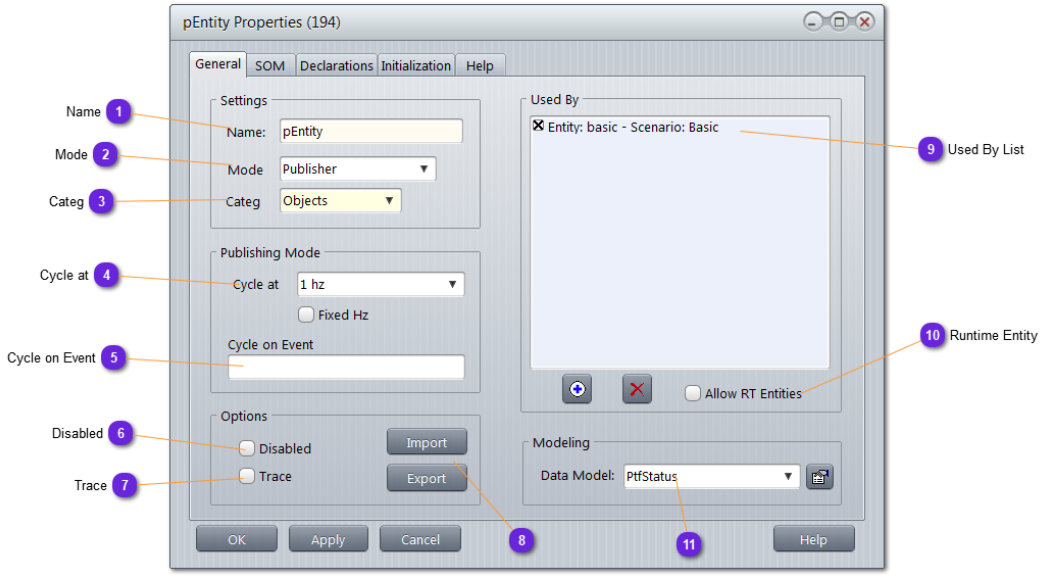
**Add Interaction**: create a FedItem interaction in publishing or subscribing mode according to the position of the mouse on the diagram (**top** for Publisher and **bottom** for Subscriber)

**Insert**: list all interactions imported from the Federation file or the OMT definition

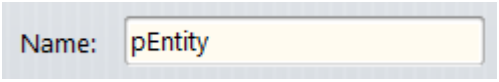
**Import**: list all exported FedItem Interactions in Data/Shared directory

**Select All**: select all FedItems (useful for deletion)

# General

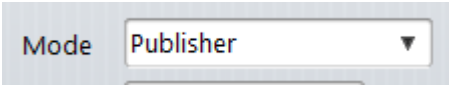



## 1 Name



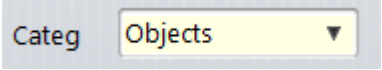

Name of the Federate. Must be unique. A good practice is to put a minus p (or s) before the name to type it *(i.e: pMyPublisher)*.

## 2 Mode



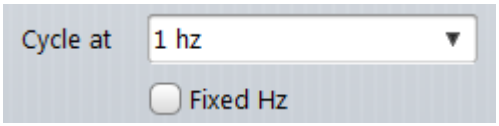

One Federate object cannot publish and subscribe using the same federate.

## 3 Categ




Chose if the Federate will handle Objects or Interactions.

## 4 Cycle at

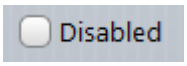



For Publishers, the selected Objects or Interaction [Code] can be processed at requested frequency. If On_Event is selected, only the specified Event will trigger the Federate Code. Both Frequency and Event can be combined.

## 5 Cycle on Event

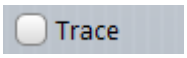

Write here the triggering Event.
If several events can trigger the FedItem, separate them with commas.

## 6 Disabled




If checked, the Federate will not be generated for the Simulation.

## 7 Trace




If checked, runtime console window will get more data to display.
Code generation will include numerous tracing printout.
This is CPU demanding so, useless for release systems.

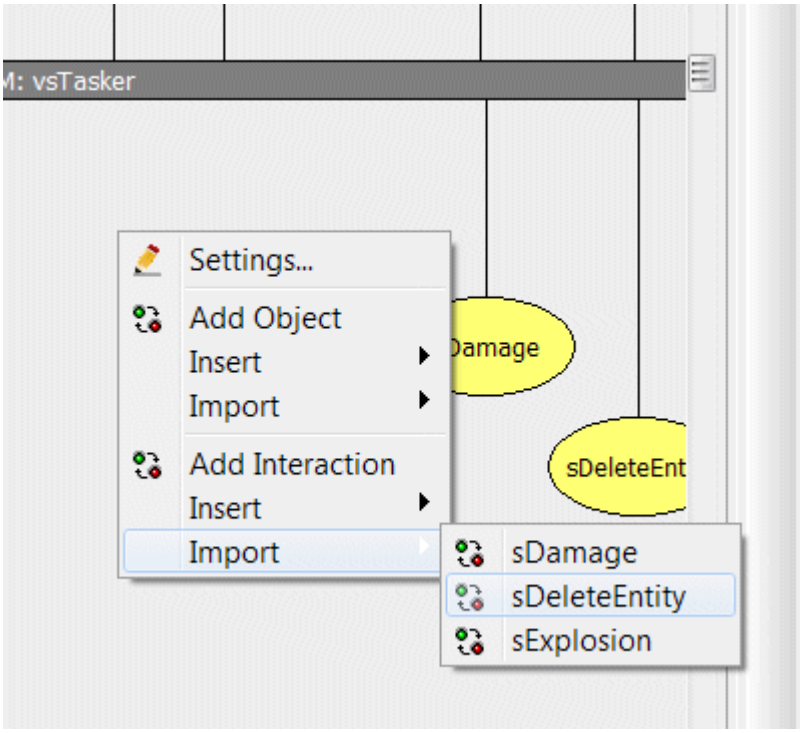
# 8 Import / Export


The FedItem can be exported as a file or can be imported from another (exported) file.
If imported, all current definitions will be replaced.

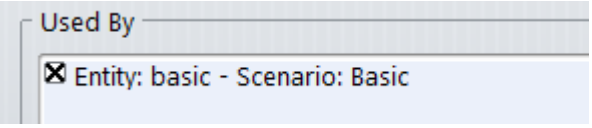
Exporting a FedItem can be useful to build a library or to exchange a FedItem from one database to one another.
It can be a good idea to create a subdirectory into the data/shared directory. One directory per FOM to avoid mixing FedItems from different FOM that would not work.

Exported FedItems can be imported from the contextual menu. Four kind of files are automatically selected:
1- Publishing Object (extension fpo)
2- Publishing Interaction (extension fpi)
3- Subscribing Object (extension fso)
4- Subscribing Interaction (extension fsi)

You can import any of them from the Import popup menu:



## 9 Used By List



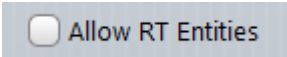

List here all entities (of any scenario) using this FedObject at runtime. This can be useful when combined with a Modeling Object as vsTASKER automatically generates some code to facilitate the data processing.

Use ⊕ to select which entity of the current scenario to add.

Use ✕ to remove from the list the selected entity.

Uncheck entities that must not be processed at runtime.

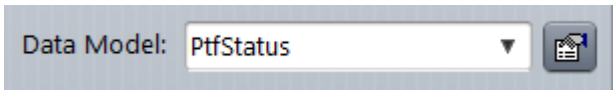
## 10 Runtime Entity




When switched ON on a Publisher, any runtime entity on vsTASKER will be added to the FedItem and be ready to publish, unless discarded from the Register panel of the FedItem by setting the `entity` variable to `NULL`. If OFF, runtime entities will be ignored by the FedItem.

When switched ON on a Subscriber, the `discover` part (see the discover pane) will automatically create a "default" entity and will try to set the `local` pointer to the attached component (if found) for the next focus.

## 11 Data Model



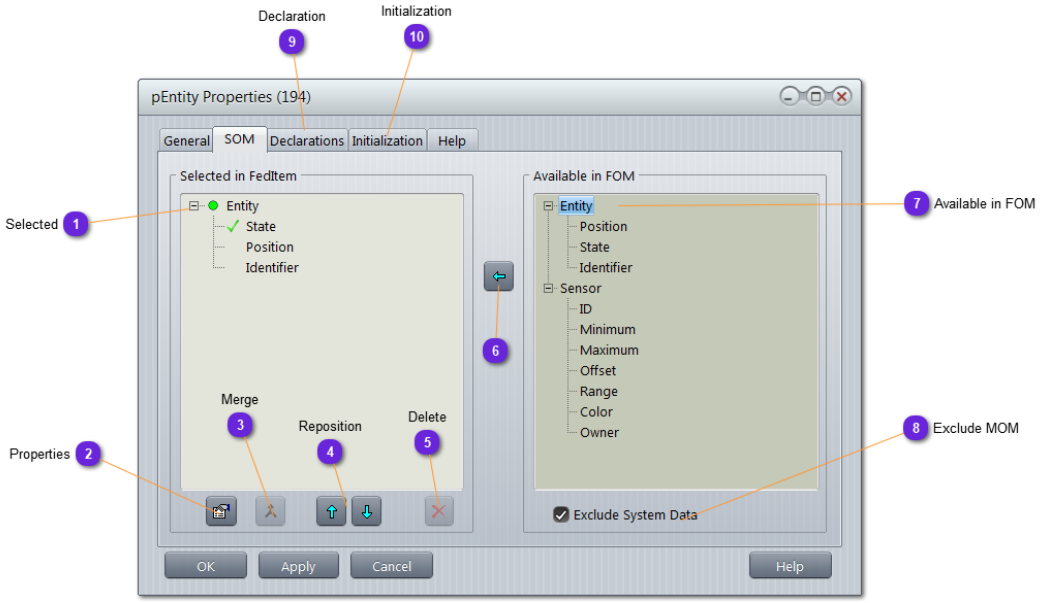

Select the corresponding DataModel.
The selected DataModel shall be a Class that does map the Class/Attributes or Interaction/Parameter of the current FedItem definition.

To understand how DataModel will be used by the code generator, let's get the above sample with `hlaEntity` DataModel selected.

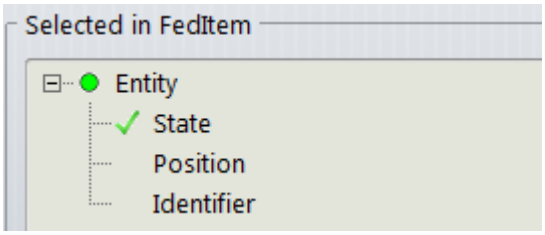
With DataModel, the following variables will be declared into the corresponding FedItem class:

```
hlaEntity* local;     // automatically set for Objects, only under multiple instance mode.
hlaEntity  rti_data;  // only when Entity Based option is turned OFF
```

# SOM




## 1 Selected




This part contains the main Object including all the Attributes (or the main Interaction including all Parameters) selected from the Available in FOM list (right most).
Although several Objects can be selected inside one only FedItem, it is advisable not to do it even if this can work (user must know what they are doing in such process).

The main (or base) Object or Interaction is the one with a circle at the left. When the circle is green, it means that the Object is ready to Publish or Subscribe.

Attributes or Parameters appears listed below the base Object/Interaction. When the Attribute/Parameter name is ticked with a green mark, it means ready to be used for Publish or Subscribe. Otherwise, it will be ignored by the RTI.

## 2 Properties

Use this as a shortcut of the double click on one Item of the above list.

## 3 Merge

Use this button after having selected a child Object or a child Interaction to force the merging of all its Attributes/Parameters with the parent one.
This is mandatory as with RTI, an Object/Interaction must be published as a flat list.
See the RPR-FOM sample.

## 4 Reposition

Use the Up and Down arrow to move up or down a selected Attribute/ Parameter in the list.

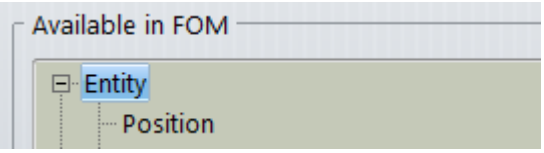
## 5 Delete

Use this button to delete from the list the selected Object/Interaction or Attribute/Parameter (including all user defined content for the item).

## 6 Select

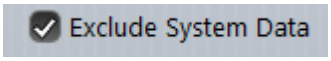
Move to the Selected list any Object/Interaction or Attribute/Parameter selected from the list.
If the selected item already exists in the Selected list, it will not be copied.

## 7 Available in FOM



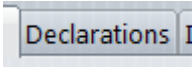

Lists all the items belonging to the FOM, of the same type as the current Federate (Categ).
To select an Object or Interaction from the FOM, just select the base Item name and use  .
You can also select an attribute to add it if missing in the selected Item.

## 8 Exclude MOM



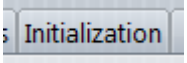

When this option is selected, the list is cleaned from all system (MOM) definitions.

## 9 Declaration




Declare here all local data that would be needed by the Attributes (or the Base item) code to get or store the SOM data.
This data is private to the Federate Object class.

## 10 Initialization



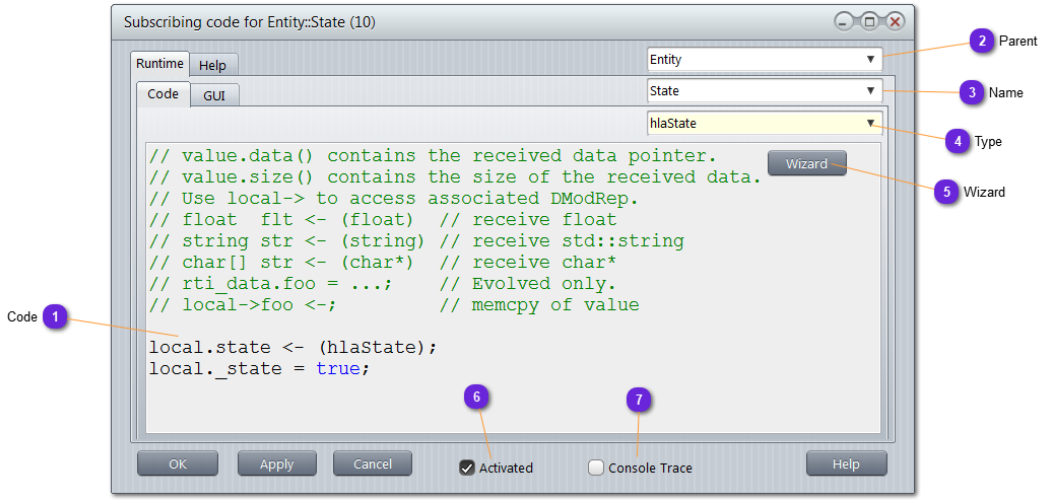

This part can be used to initialize the private data (see Declaration) at Init time or Reset time.
(i.e: if you need to publish a particular entity, set the Entity pointer like this:

```
ent = scen()->findEntity("myEntity");
```

Attribute code will then use `ent` variable to publish to RTI or subscribe to entity internal data.


*Refer to the Developer Guide for more details on system phases/events.*

# Runtime Code




## 1 Code

```
local.state <- (hlaState);
local._state = true;
```

Write here the code to get/put data from/to the RTI.
The comments remind of the syntax to use for publishing or subscribing data in/out.

## 2 Parent


Current parent of the selected Attribute/Parameter (ie: PosY) belonging to Object/Interaction (ie: Participant).
The list displays all parents if available, excluding MOM ones.
No effect if manually changed.

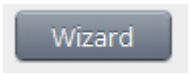
## 3 Name


Name of the selected (current) Attribute/Parameter.
The list displays all Attributes/Parameters having the same Parent (ie: Participant).
User can change with immediate effect. Same thing as closing the window then selecting another Attribute/Parameter from the SOM window.

## 4 Type


Current type of the Attribute/Parameter.
No effect if manually changed.

## 5 Wizard




Click this button to automatically fill the user code part with appropriate data.
The wizard does not work on all configurations and for all RTI versions.

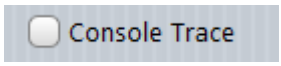
## 6 Activated


When this is ON, vsTASKER will generate the code (above).
If not activated, code will be skipped.

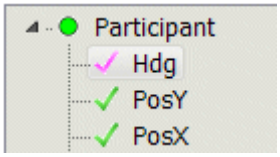
Note that when the Code part is empty (no user code), even if this option is turned ON, code will be skipped.
It can be useful to force the publish mode of an Attribute even without user code.
This can happen when user will publish a value using the generated interface.
To force publishing of the Attribute, just put the following code:

```
void;
```

# 7 Console Trace




This checkbox is for subscribers only.
When checked, even without user code, anytime a data is received from the RTI for the particular Attribute/Parameter, the simulation engine will output a printout.
This can be useful to know what are the data that other federates are producing.
When tracing is ON, the tick mark is shown in magenta:



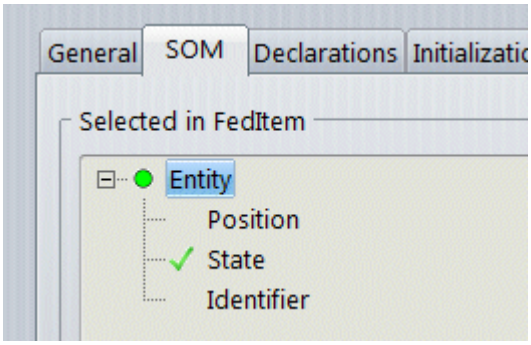

Option shall be turned OFF for release versions.

# Federate Code

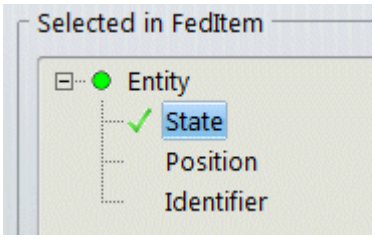
In a subscribing FedItem, all attributes or parameters are fed individually by the RTI before the Object or Interaction is called for update.
With a publishing FedItem, the Object or Interaction is first called before all attributes or parameters fed the RTI.

*For an Object, only changing attributes are distributed.*
*For an Interaction, all parameters are distributed.*

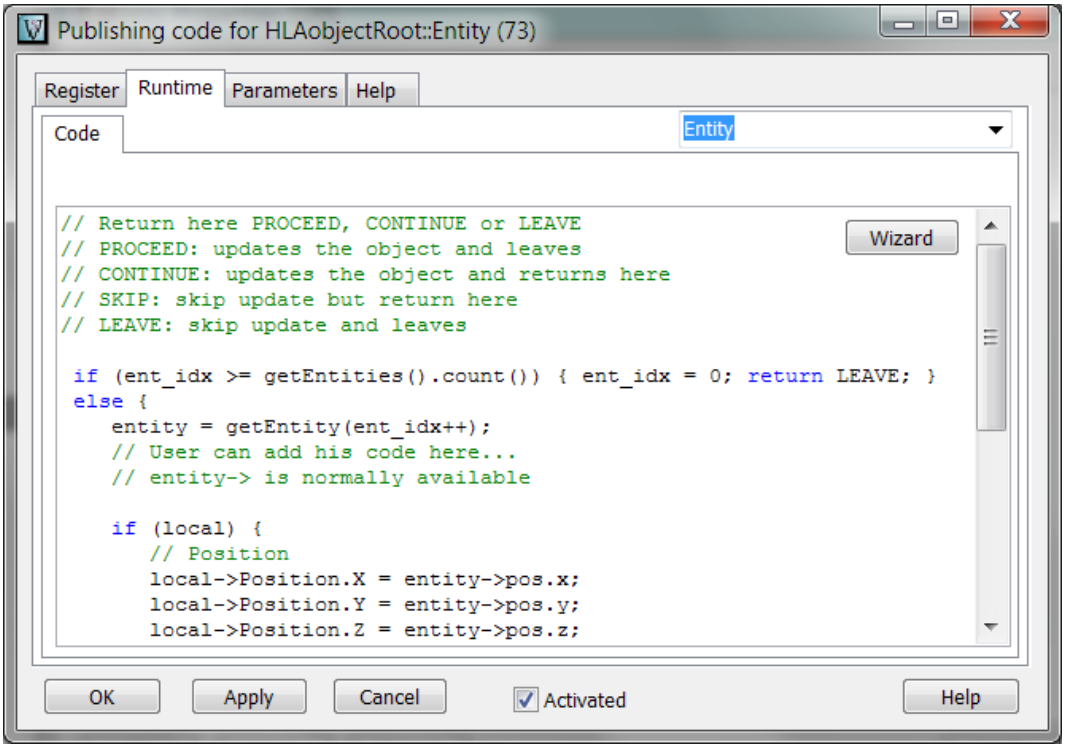
To edit the Object or Interaction and access the user code, double click it and edit.



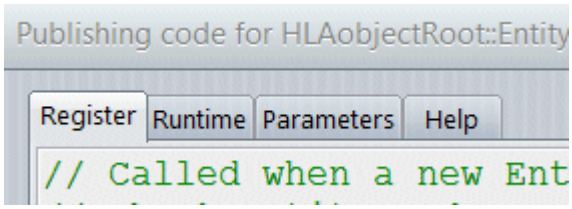

To edit the Attribute or Parameter and access the user code, double click it and edit.



# Object




## • Register

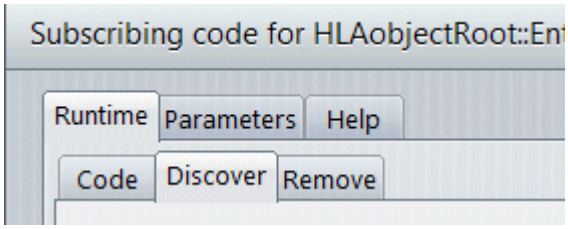



This panel is available only for **publishing** FedItems.
When an Entity is created (either at simulation Start or during Runtime), when the (Allow RT Entities) option is selected, the Runtime part is called right after, with the entity pointer.
If the FedItem shall not retain the new entity, then the code must set it to NULL, otherwise, the new entity will be added to the `used_by` list.

Sample of code for Register:

```
if (!((Entity*)entity)->status ||
     ((Entity*)entity)->status->isType(_Unknown)) entity = NULL;
```

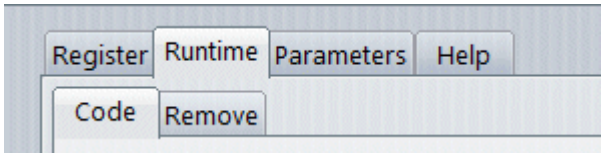
## • Discover




This panel is available only for **subscribing** FedItems.

Whenever a new Object is found on the Federation (Object vsTASKER is subscribing to), this part will be called.
If the Allow RT Entities option is checked, then the discover code will automatically try to create a "default" entity.
If you want to create another entity type, the `entity` pointer setting must be set in this panel before the automatic code is reached.

## • Code




The Object base code is called at the requested frequency (or upon event or manual call) and the return values will tell what to do with the request to **publish**:

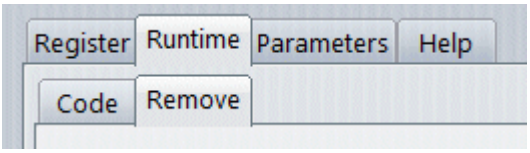
- ***PROCEED***: all attributes will be processed (call of all attribute code in sequence)
- ***CONTINUE***: same as *PROCEED* but the same function will be called again immediately after (and not at next cycle)
- ***LEAVE***: do nothing and stop looping (only way to quit a *CONTINUE* loop).
- ***SKIP***: do not proceed attributes/parameters but continue looping.

For **subscribed** Object the code section will be called after all attributes have been received. Normally, it is in this section that the data of each attribute will be used and stored in the simulation environment.

Each time the code part is called, `ent_idx` index is increased to parse all `used_by` slots. `entity` pointer is then usable to set the data that will be updated.

## • Remove

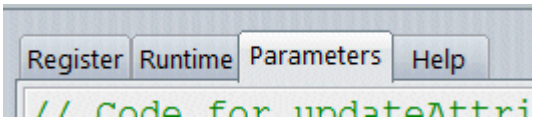
### Object




This code is called when the object is destroyed by the RTI.
Normally, putting this code is enough:

```
del(entity);
```

## • Parameters

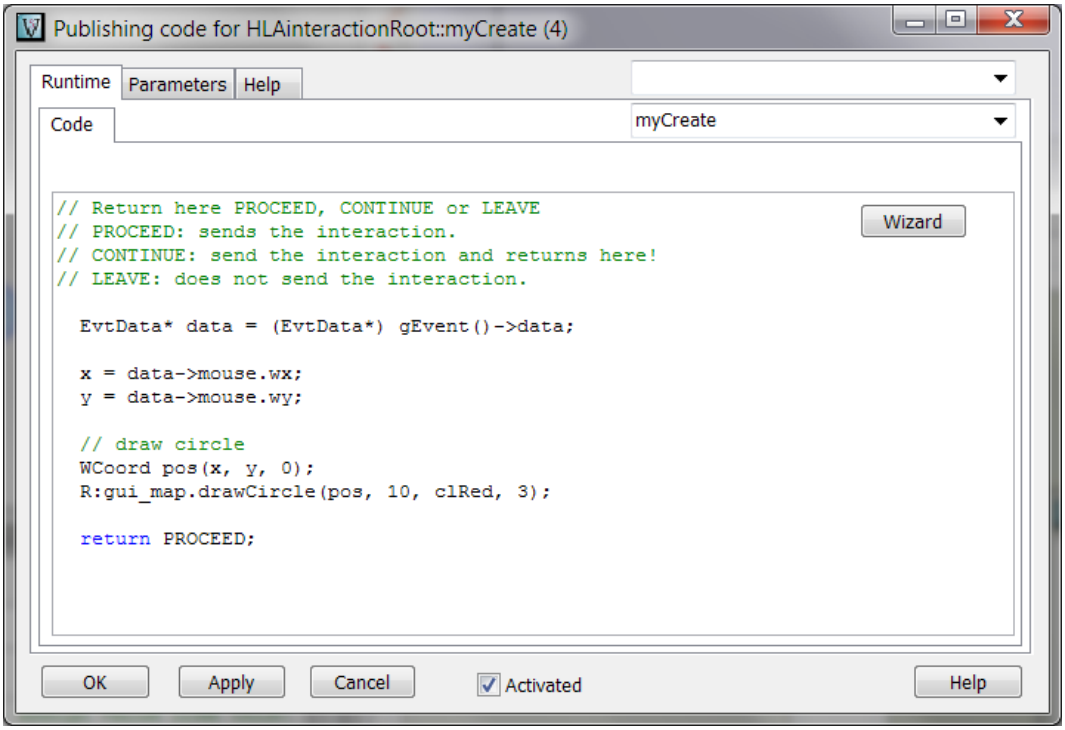



This part of the code is put just before sending the Object Attributes.
Some variables (like `_userTime`) can be locally modified if needed.
You can see this code in the following function of the [database_name].intfc.cpp file:

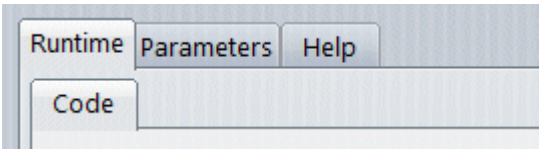
```
int vsTaskerAmbassador::updateObject();
```

just after the comment: `// parameters user code:`

# Interaction




## • Code




The Interaction base code is called at the requested frequency (or upon event or manual call) and the return values will tell what to do with the request to **publish**:

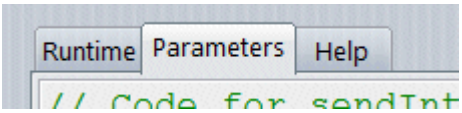
| | |
|---|---|
| PROCEED | All parameters will be processed for publish |
| CONTINUE | Same as *PROCEED* but the same function will be called again immediately after (and not |
| LEAVE | Do nothing and stop looping (only way to quit a *CONTINUE* loop) |

For **subscribed** Interaction: the Code section will be called after all parameters have been received. Normally, it is in this section that the data of each parameter will be used and stored in the simulation environment.

### Interaction

In the above example, `x` and `y` variables (declared in the FedItem) are set with data coming from the Event that has triggered the Interaction (FedItem).
Then, the `drawCircle` runtime graphics function is called (so that to display a circle on the GUI map).
`PROCEED` is returned, meaning that all parameters will be processed to send the data thru the RTI.

## • Parameters



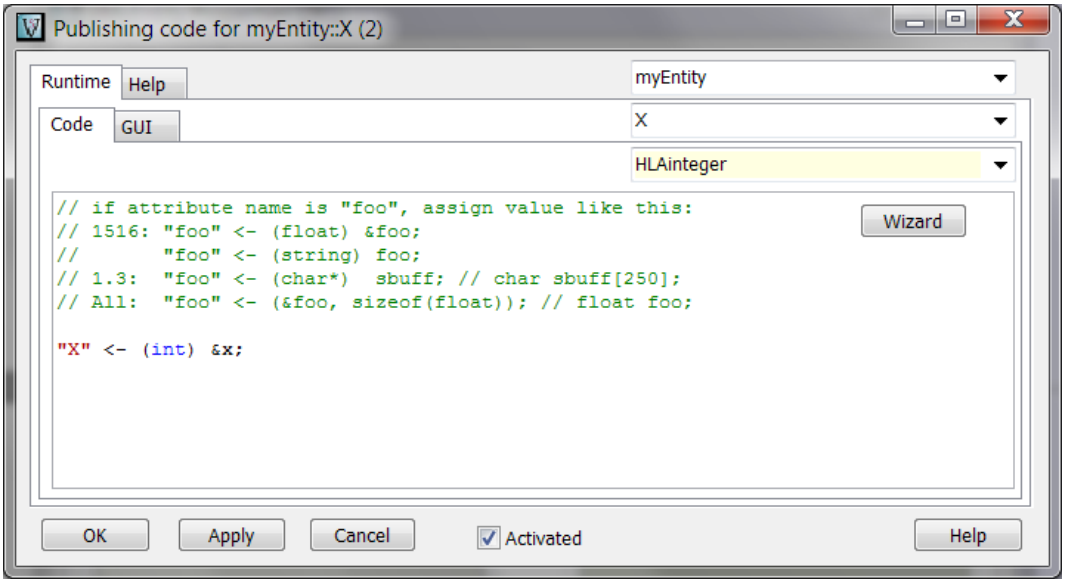

This part of the code is put just before sending the Interaction Parameters.
Some variables (like `_userTime` and `_userTag`) can be locally modified if needed.
You can see this code in the following function of the [database_name].intfc.cpp file:

```
int vsTaskerAmbassador::sendInteraction();
```

just after the comment: `// parameters user code:`

# Attribute - Parameter

## • Publish




To publish, the local (FedItem) data must be copied to the Attribute/Parameter name. Using the following formalism, vsTASKER will preprocess the code and generate the correct C++ code.

`"name"`, where name is the one of the Attribute/Parameter.
`<-` preprocess symbol
`(type) &variable;` local variable holding the data to publish.

## • Fine Tuning

All attributes will be published when the Object FedItem is called.
Most of the time, several Attributes will not need to be updated because the value has not changed enough.
For CPU load control, you can define a Boolean to force or not the publication of the Attribute.

### Declaration in the FedItem Object:

```
struct {
   struct {
       WCoord current;
```

```
        bool update;
    } pos;
} local;
```

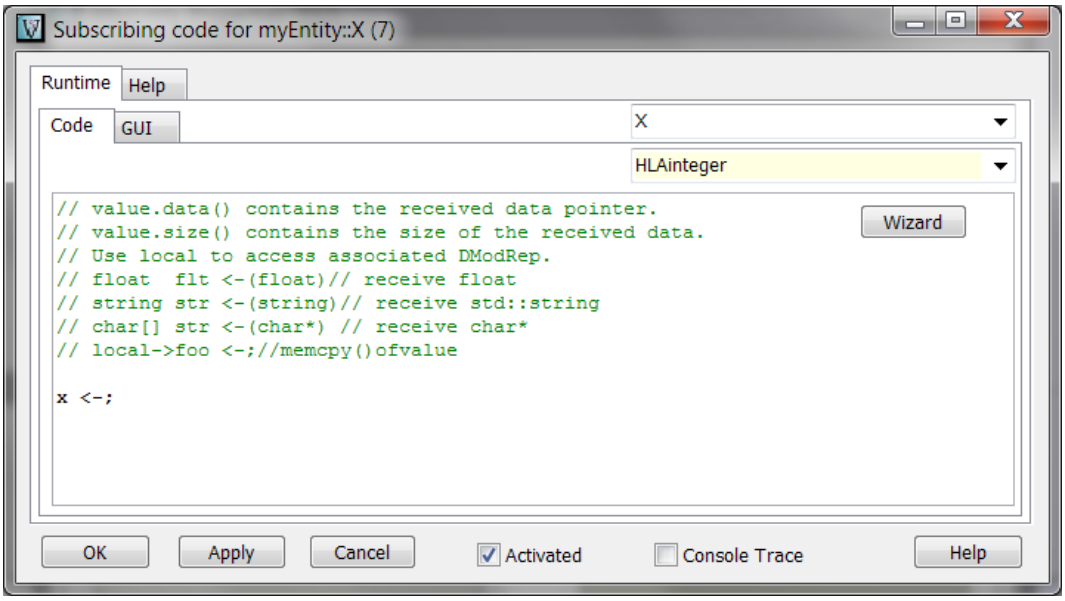
### Code in the FedItem Object:

```
if (entity->pos.distanceTo2D(local.pos.old) < 10) {
    local.pos.update = true;
    local.pos.current = entity->pos;
}
else local.pos.update = false;
```

### Code in the FedItem Object Attribute:

```
if (local.pos.update) {
"entityPos" <- local.pos.current;
}
```

## • Subscribe

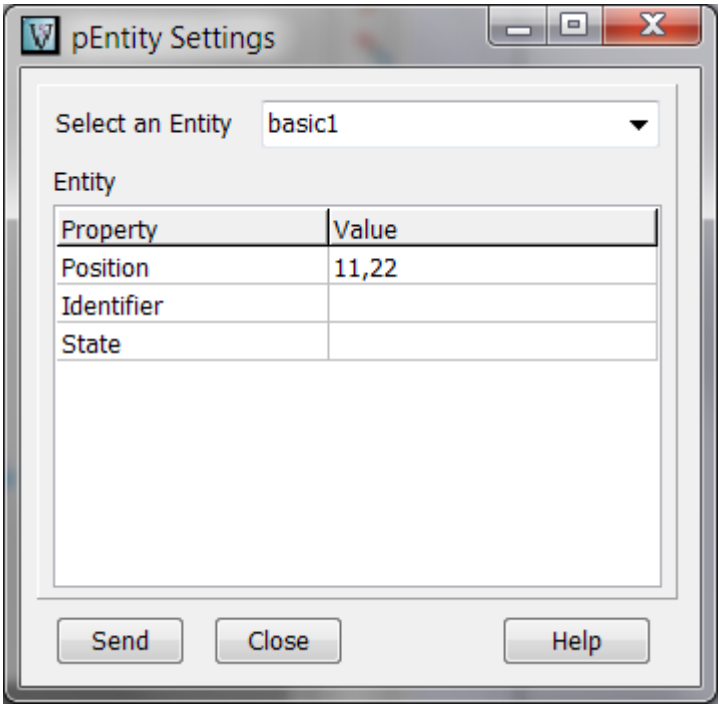



When subscribing to HLA, data coming from the RTI must be stored locally before being used at the Object/Interaction code.
This is done using the preprocess symbol `<-`

`x` local variable that will receive the data. Must be of the correct type to avoid runtime error with the `memcpy`.
`<-` preprocess symbol

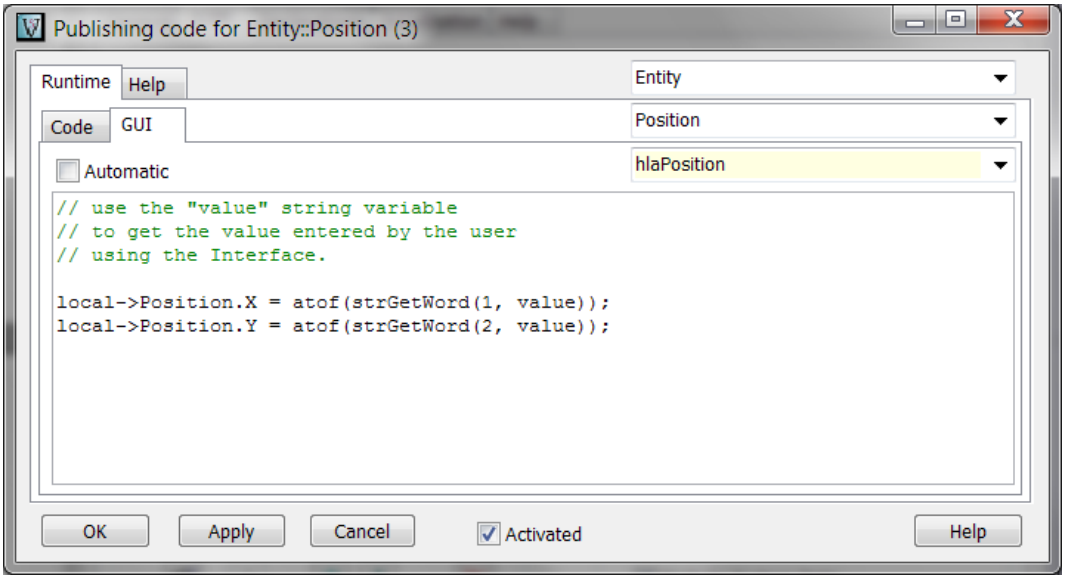
## • GUI

vsTASKER allows user to manually publish Object Attributes or Interactions Parameters using the runtime window of each publishing FedItem.
The generated code works fine for simple types (`string, integer, float, double, long`...) but when it get to complex ones, user must deactivate the Automatic mode and do the coding itself.
It is not difficult as it consist of getting the character string entered by the user in the Dyn-UI and copy it to the proper FedItem data.
The publishing is done right after.




Above is the Runtime FedItem window for a publisher.
See explanations there.

### Attribute - Parameter




here, the Position string will be received in the variable value and will contain: `11,22` (as entered by the user).
`local` variable is automatically set by vsTASKER when a DataModel is associated with the FedItem.

So:

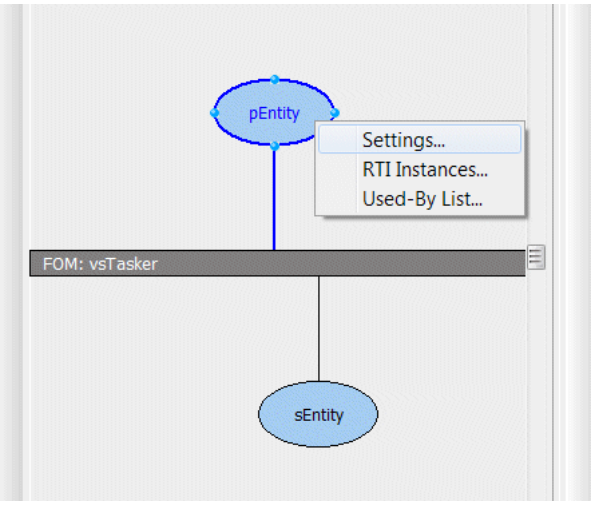
```
strGetWord(1, "11,22) -> 11
```

and:

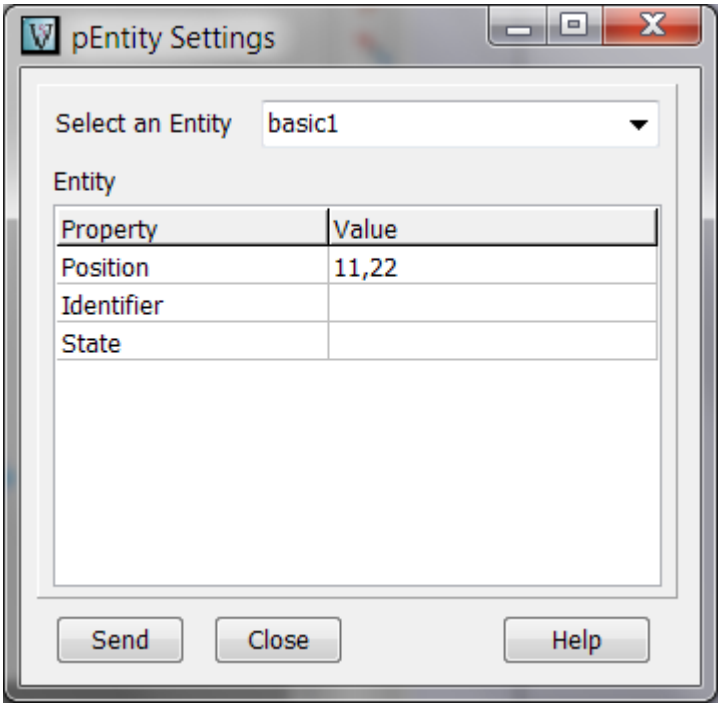
```
strGetWord(2, "11,22") -> 22
```

# Runtime Windows

When the simulation is running, HLA FedItem symbols can be selected and observed to perform various actions and monitoring:
Just select the Publishing or Subscribing FedItem and right click.



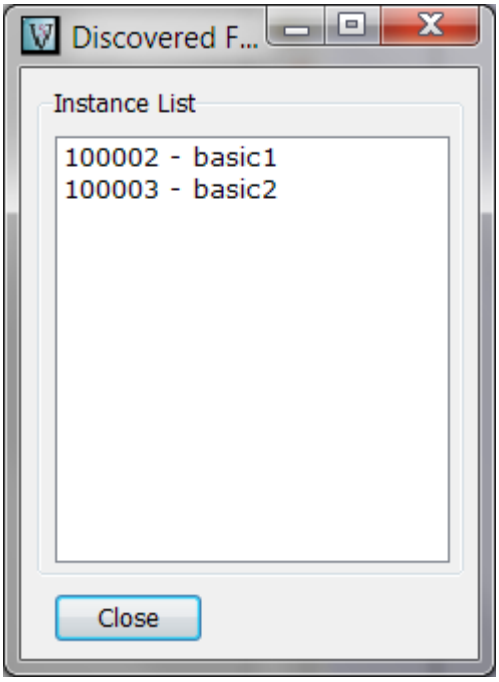

## • Settings




With this window, user can manually set the data of any Attribute/Parameter of the selected FedItem.

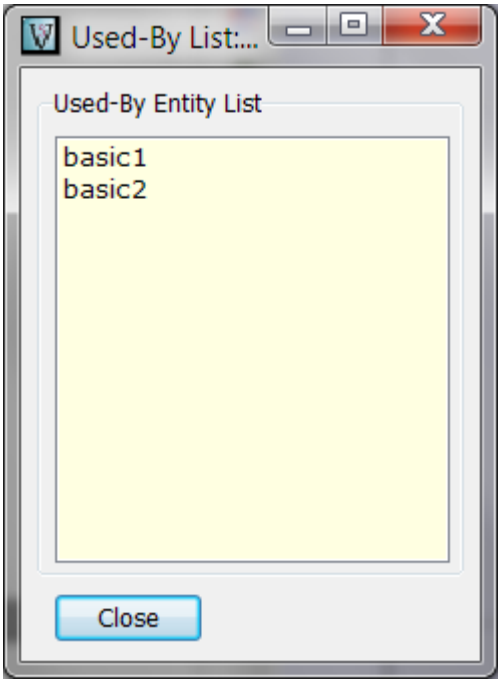
When Send button is pressed, entered data will be sent to the Simulation Engine. `setUserData()` function will receive all of them.
User can process the data string in the GUI code panel of each Attribute/Parameter.
See GUI for more explanations.

## • RTI Instances




This window lists the RTI Object handle associated with each Entity object.
This pair is used by vsTASKER to automatically retrieved the correct Entity (and then, set the `entity` pointer) for any subscribing event or to retrieve the correct `handle` for any publishing event.

## • Used-by List

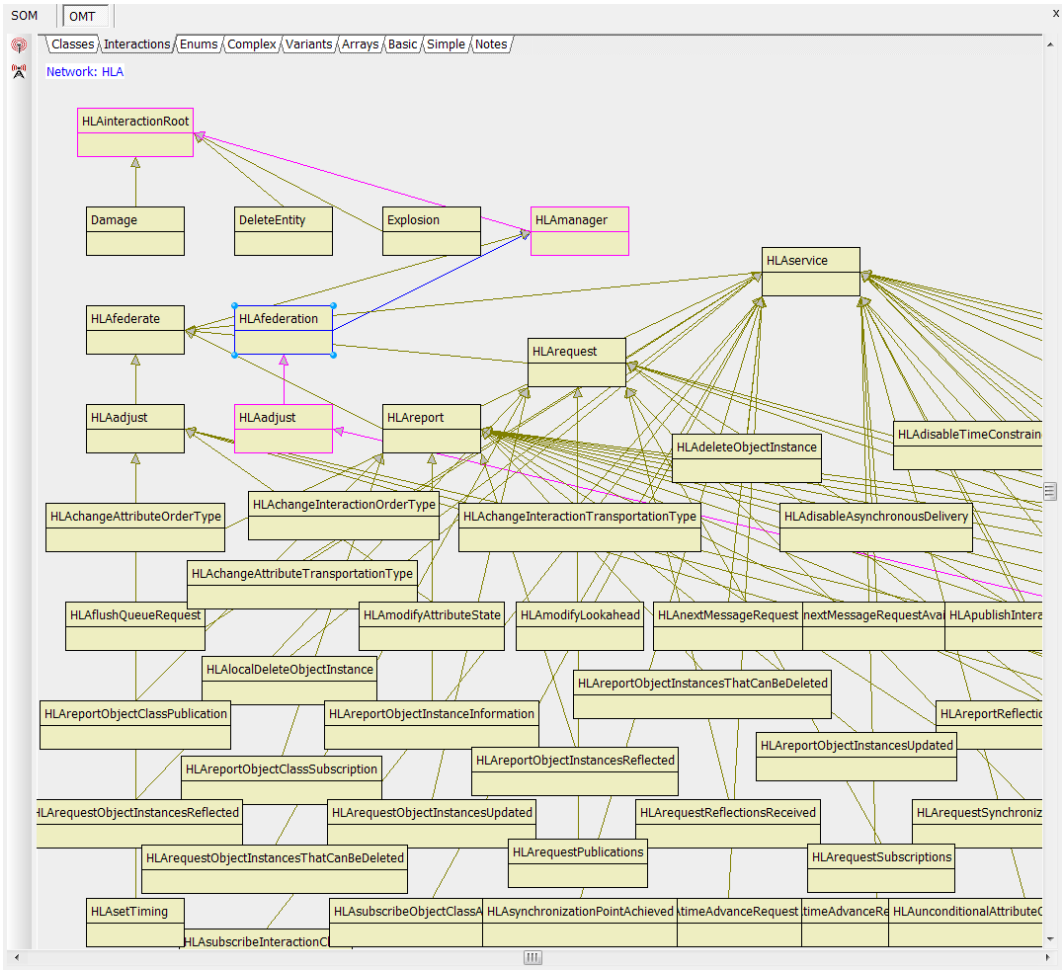



This window lists all the Entities members of the `used_by` list of the FedItem.

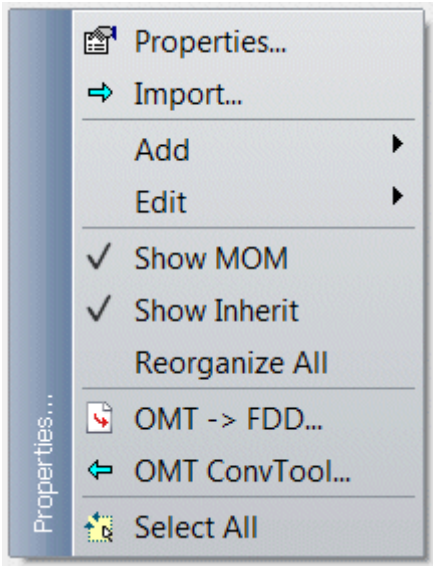
# OMT

vsTASKER offers a visual OMT editor that can be used either for viewing Federation contents, modifying some or to create Federations from scratch.
With its FDD converter, it is then possible to upgrade a Federation and update the SOM in a looping process that drastically speed up the simulation development process.

Also, the visual paradigm greatly helps debugging and understanding a HLA simulation with complex Federations.




- **Popup Menu**




**Properties**: call the property window

**Import**: *not available yet*

**Add**: list all OmtItems that can be added in the selected category.

**Edit**: list all OmtItems already defined in the selected category.

**Show MOM**: display or hide the root MOM objects user cannot modify

**ShowInherit**: display or hide the inheritance links

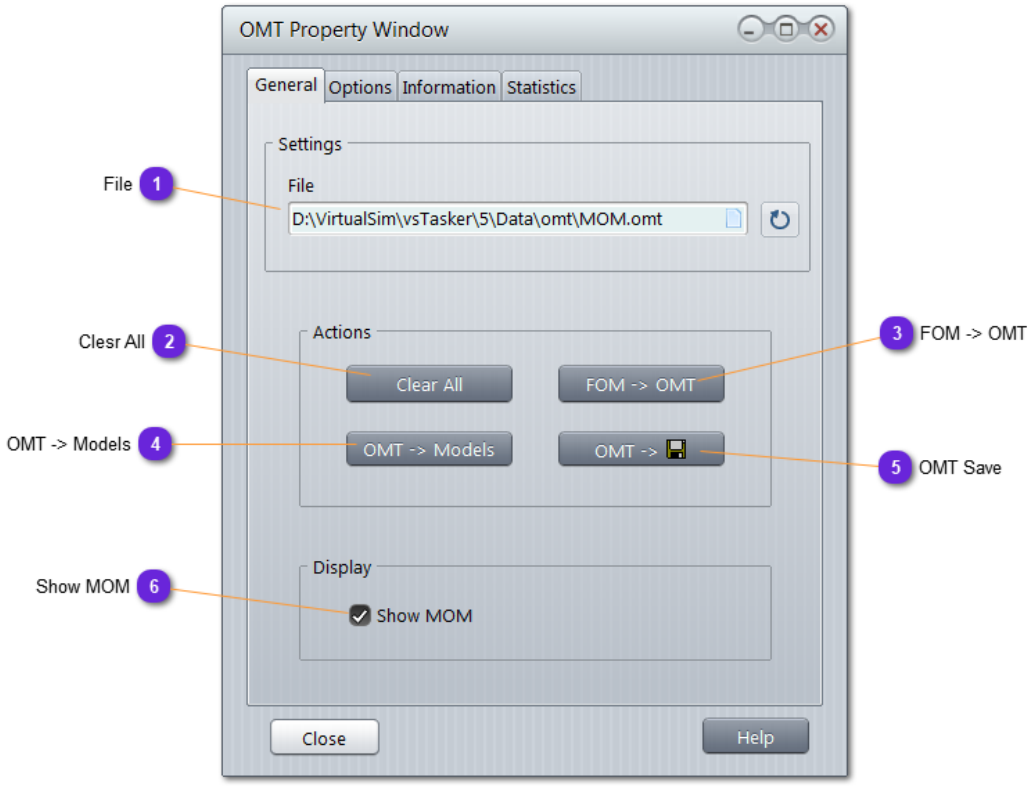
**Reorganize All**: try to organize all icons into a grid

**OMT -> FDD**: call the formatted saver

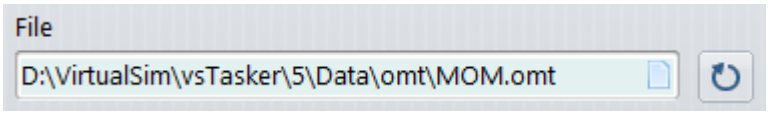
**OMT ConvTool**: call the converter tool

**Select All**: select all OmtItems in the category (useful for deletion)

# Properties




## 1 File

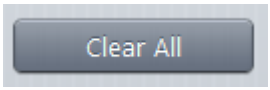



Specify here the base MOM file that describes the minimum system definition of a Federation.

- You **can** use the MOM.omt file in /omt directory for 1.3 federations.
- You **must** use the MOM.xml file in /fom directory for 1516 federations.

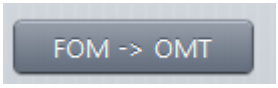
Using  , the file is reloaded, discarding all user changes in the OMT environment.

## 2 Clesr All



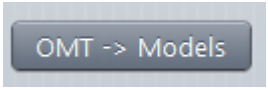

Removes all OMT data structure from the database.

## 3 FOM -> OMT




Rebuild the OMT data structure from the loaded FOM.
In case of a 1.3 Federation, types will be minimum.
1516 Federations provides maximum data information.

## 4 OMT -> Models

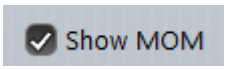



See Converter.

## 5 OMT Save


See FDD Saver.

## 6 Show MOM

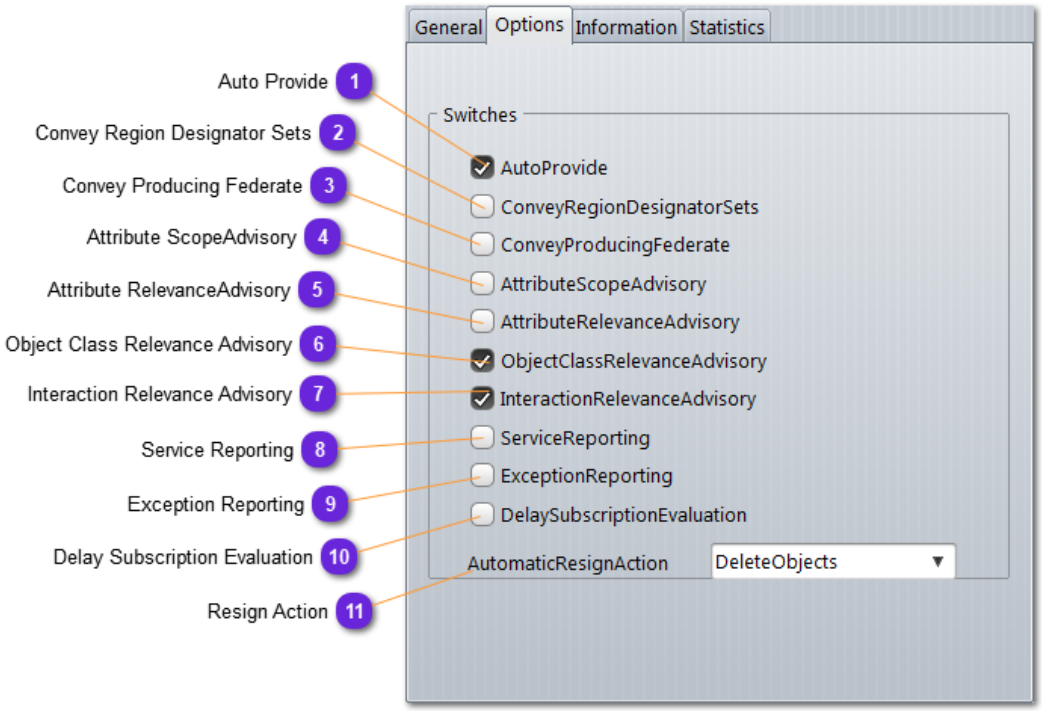



When selected, system objects (management) are displayed on diagrams.
Most of the time, they do not bring anything to the user.
When OFF, only user defined data structures are displayed for edition.

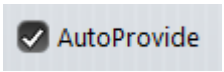
# Options

Works only for HLA1516 and Evolved.
Lists all the switches defined in the Federation. Configuration of RTI activities performed on behalf of a federate
Can be changed by the user.

*Modified settings will **not** be saved on the local database.*
*User must generate a new FDD using the Saver window to export the changes.*

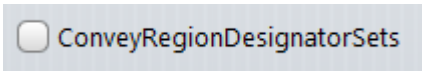



## 1 Auto Provide



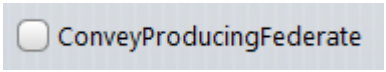

Should the RTI automatically solicit updates from instance attribute owners when an object is discovered.

## 2 Convey Region Designator Sets



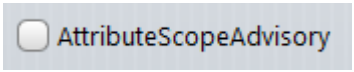

Should the RTI provide the optional Sent Region Set argument with invocations of Reflect Attribute Values and Receive Interaction.

## 3 Convey Producing Federate



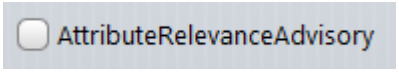

Not available in HLA 1.3 or 1516-2000. Cannot be required by federates in a mixed HLA 1.3/2000/2010 federation.

## 4 Attribute ScopeAdvisory



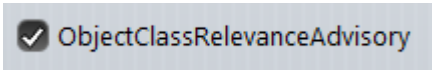

Should the RTI advise federates when attributes of an object instance come into or go out of scope.

## 5 Attribute RelevanceAdvisory



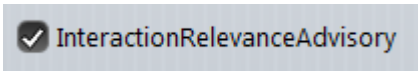

Should the RTI advise federates about whether they should provide attribute value updates for the value of an attribute of an object instance.

## 6 Object Class Relevance Advisory



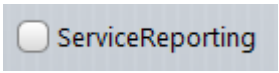

Should the RTI advise federates about whether they should register instances of an object class.

## 7 Interaction Relevance Advisory



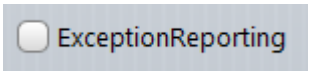

Should the RTI advise federates about whether they should send interactions of an interaction class.

## 8 Service Reporting



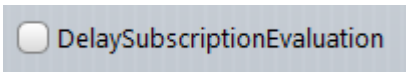

Should the RTI report service invocations using MOM.

## 9 Exception Reporting



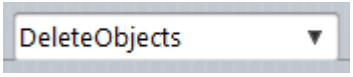

Used mainly for debugging.

## 10 Delay Subscription Evaluation




The federation execution-wide Delay Subscription Evaluation switch indicates how the RTI shall filter a TSO or RO message based on the receiving federate's known subscriptions. The setting for this switch shall come from the FDD at federation execution creation time. This switch setting shall not be modifiable during a federation execution.

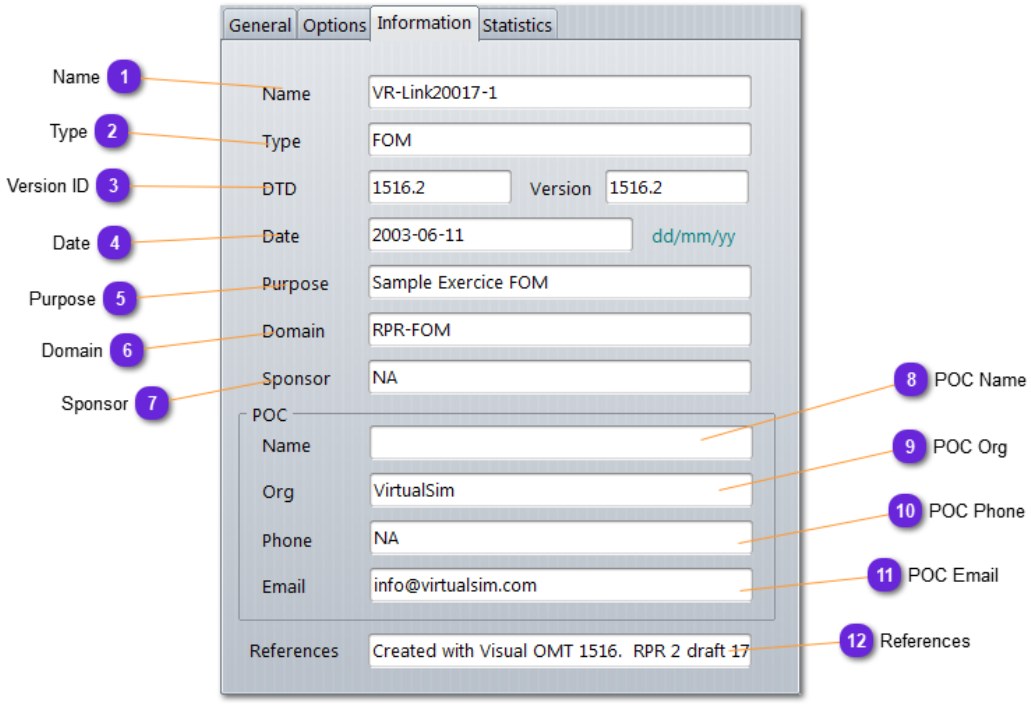
## 11 Resign Action



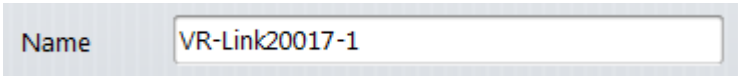

Select from the drop down list what shall be the RTI action when federate resign from the federation.

# Information

The Information panel lists the Object model identification table is done to associate important identifying information with the HLA object model.
It is important to include a minimum but sufficient degree of meta-level information in the object model description. For instance, when federation developers wish to pose detailed questions about how a federate or federation was constructed, including point-of-contact (POC) information within the HLA object model is extremely important.




## 1 Name




The name assigned to the object model

## 2 Type

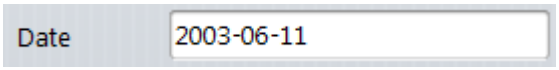

This field shall specify the type that the object model represents; valid values are

- FOM: the object model describes a federation
- SOM: the object model describes a federate

## 3 Version ID

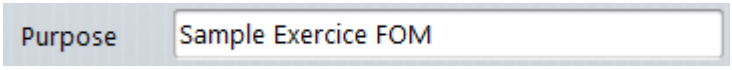

The version identification assigned to the object model

## 4 Date




This field shall specify the latest date on which this version of the object model was created or modified. The modification date shall be specified in the format YYYY-MM-DD (e.g., 1999-04-15)

## 5 Purpose




The purpose for which the federate or federation was created; may also contain a brief description of key features

## 6 Domain


This field shall specify the type or class of application to which the federate or federation applies. The following values may be used for this field, although other values are also valid:

- Analysis
- Training
- Test and Evaluation
- Engineering
- Acquisition

## 7 Sponsor


The organization that sponsored the development of the federate or federation

## 8 POC Name


The point of contact (POC) for information on the federate or federation and the associated object model; shall include an honorific (e.g., Dr., Ms., etc.) or rank, first name, and last name

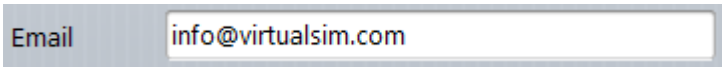
## 9 POC Org


The organization with which the POC is affiliated

## 10 POC Phone

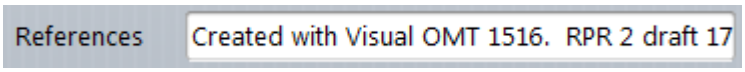

The telephone number for the POC

## 11 POC Email



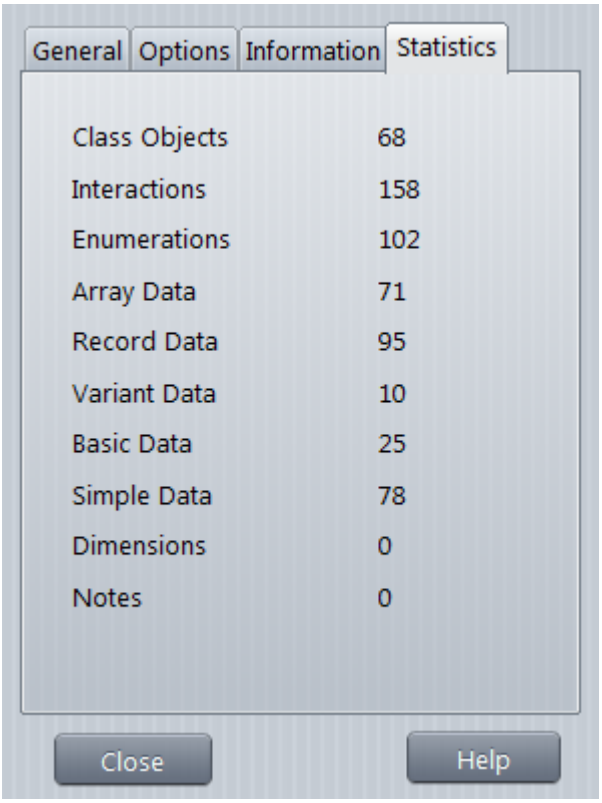

The e-mail address for the POC

## 12 References



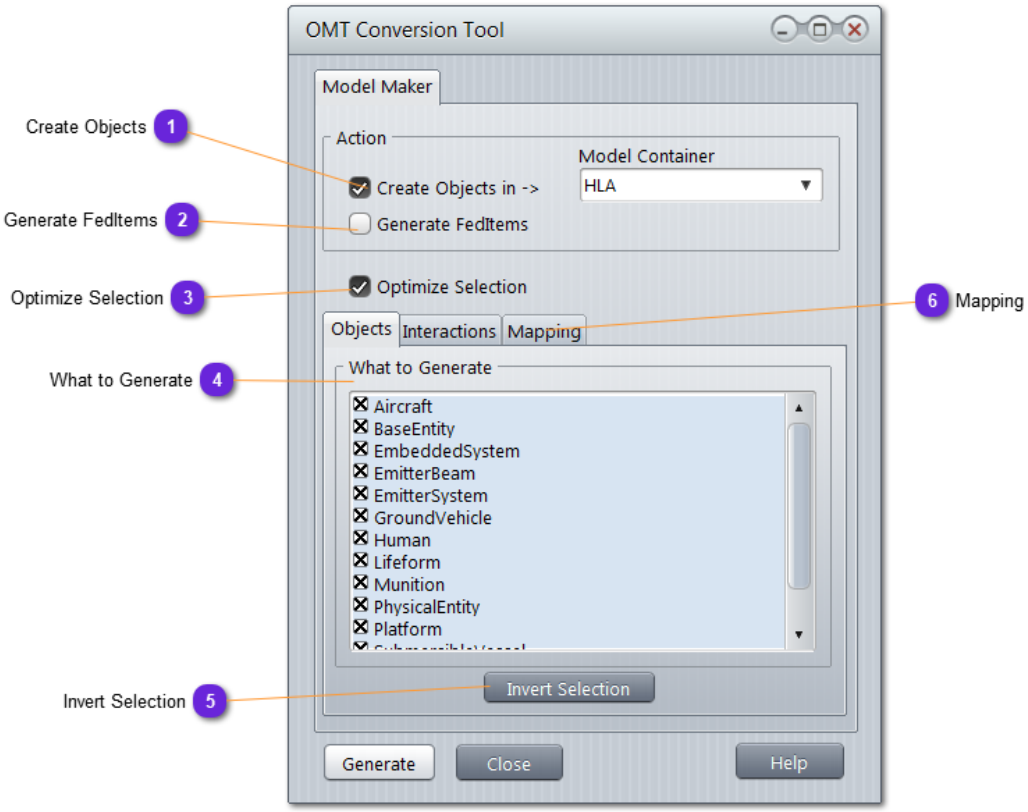

This field shall specify pointers to additional sources of information.
For example, documents that describe save/restore semantics, time management strategies, or other relevant federation policy decisions may be identified through this field.
NA shall be entered when no such references are appropriate.

# Statistics

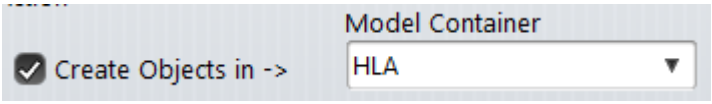



This panel summarize all imported objects and data structures from either the Federation or the OMT file.

# Converter

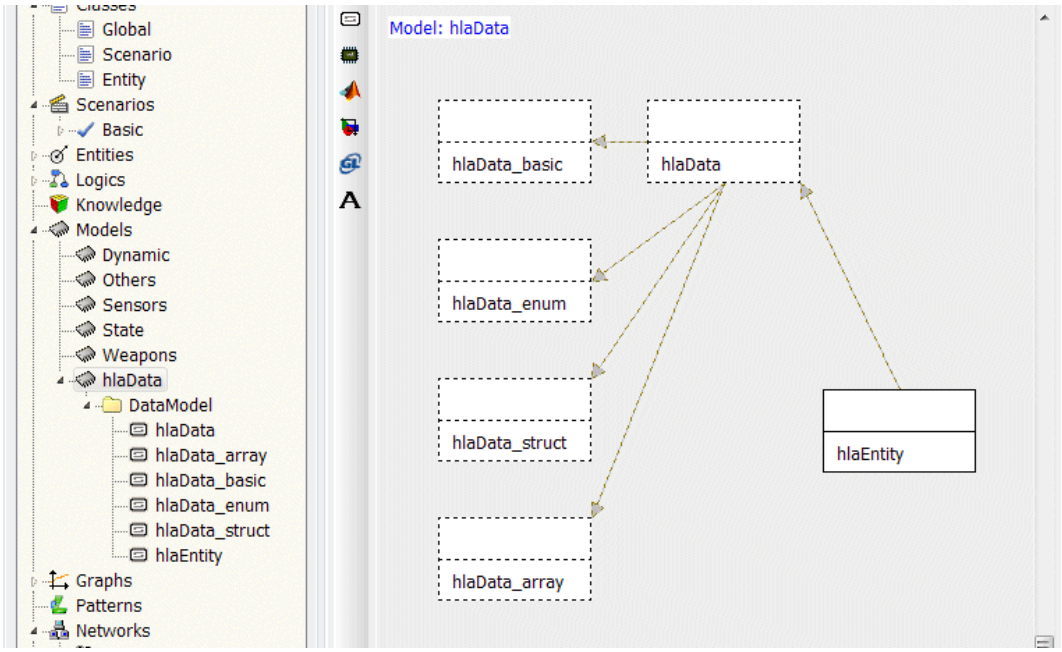

OMT Conversion Tool
Model Maker
Action
Model Container
Create Objects in ->
HLA
Generate FedItems
Optimize Selection
Objects
Interactions
Mapping
What to Generate
Aircraft
BaseEntity
EmbeddedSystem
EmitterBeam
EmitterSystem
GroundVehicle
Human
Lifeform
Munition
PhysicalEntity
Platform
Invert Selection
Generate
Close
Help
Create Objects 1
Generate FedItems 2
Optimize Selection 3
What to Generate 4
Invert Selection 5
6 Mapping

# 1 Create Objects




vsTASKER can convert all OMT data structures to C++ data representation and DataModels.
Doing so, it will become straightforward to import/export data from/to the RTI using either 1.3 or 1516 version.

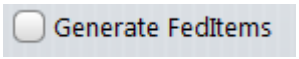
When the checkbox is ticked, all data structures will be created into the selected Model folder.
A new folder can be created by changing the name on the list-box field.




vsTASKER will use the name of the folder to automatically name the basic, `enum`, `struct` and `array` DataModels.
These structures will contains all C++ type definitions extracted from the OMT database.

Objects and Interactions will all be listed with the same name as in the OMT database, prefixed with hla.

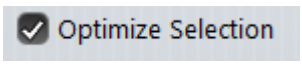
## 2 Generate FedItems




When checked, this option will force automatic generation of all OMT Objects and Interactions as FedItems (one per each) as Publisher and Subscriber.
If they already exist, confirmation for replacement will be asked.

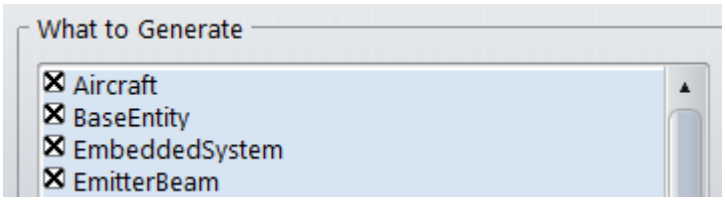
With huge federation, this option can lead to numerous (and useless) FedItems generation.
Most of the time, it is a better practice to create FedItems (in publish or subscribe mode) manually, from the SOM panel.

## 3 Optimize Selection



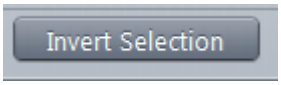

When this option is selected, vsTASKER will only select Objects and Interactions that are used (enabled) as FedItem in the SOM part.

## 4 What to Generate



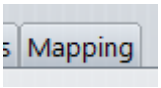

To force vsTASKER to generate Entity Catalogs based on OMT Class definitions, simply check the Objects and/or Interactions that will be used in entities.
They will be generated in Entities::Catalogs and will be using with their counterpart DataModel.

## 5 Invert Selection



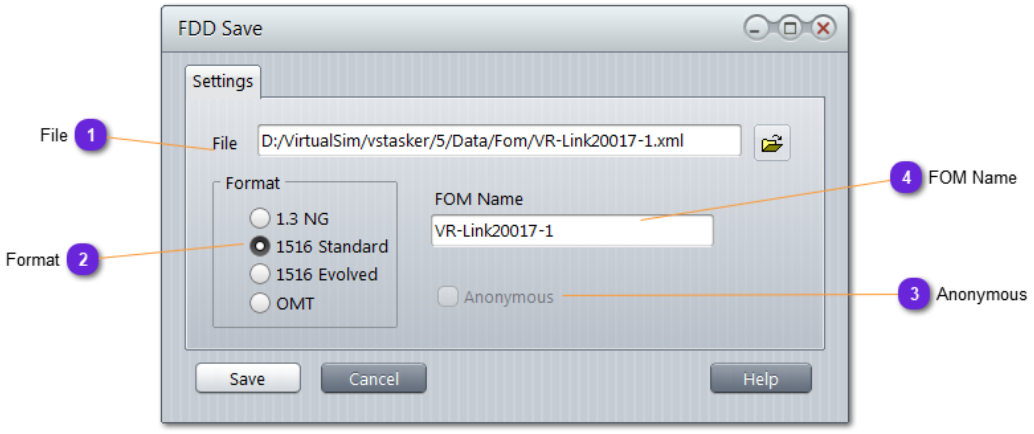

Change the selection mode of the above list.

# 6 Mapping

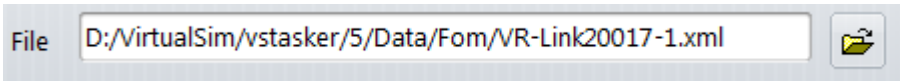



Select in this panel all objects whose corresponding DataModel must be attached to new created entity in the Catalog.
Will create a Catalog entry with all the checked object models included.

# FDD Saver

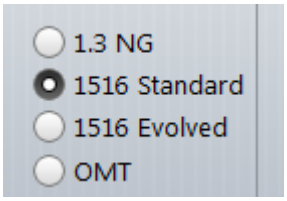



## 1 File



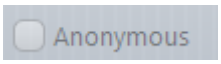

Select here the target Federation FDD file that will receive the database OMT definitions.

## 2 Format



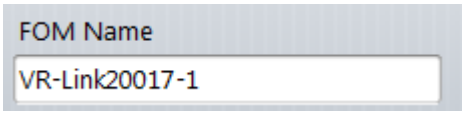

Select the data format used to define data into the FDD file.

## 3 Anonymous




*Not used.*

## 4 FOM Name

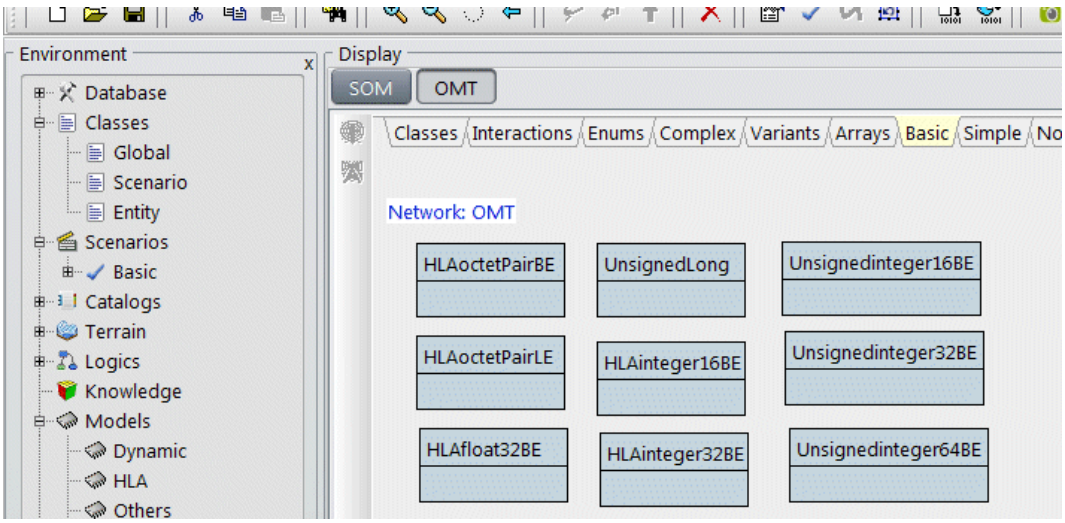



Must be specified. Will set the Federation name vsTASKER will use to connect to the RTI.

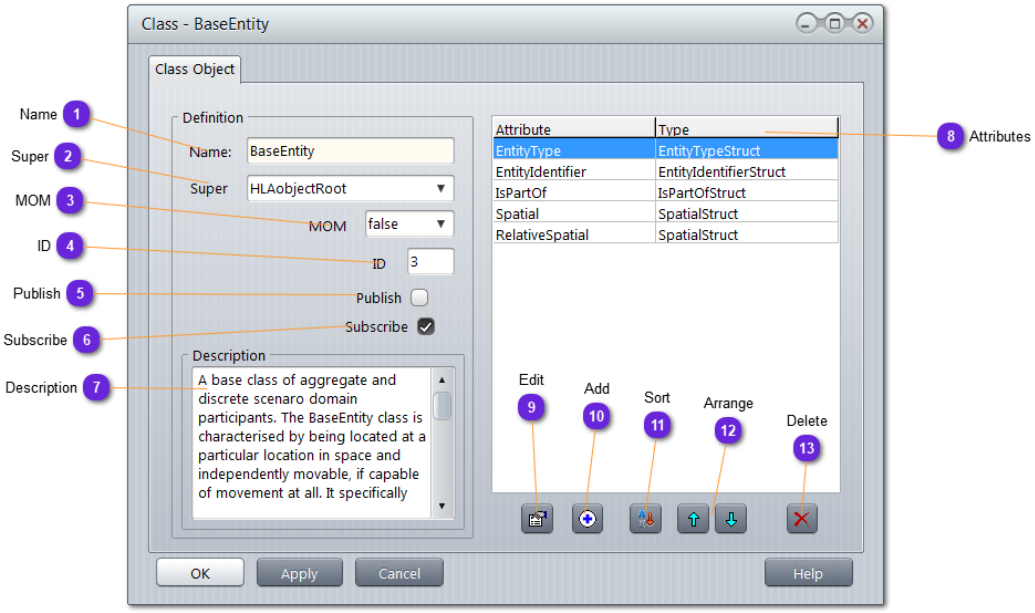
# Categories

OmtItems are divided by categories, each of them displayed with different colored symbols.
User can add as many OmtItems he needs or modify any imported one (from either an OMT definition file or generated from a Federation).

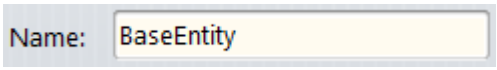
Once modified, the OMT definition must generate a new Federation file in order for the RTI to load and acknowledge the changes.
OmtItems can also be used to produce new DataModel that will be in sync with the SOM definition. They are also used to provide correct SOM to each FedItems.



# Object Class

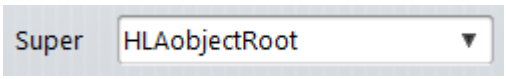



## 1 Name




Object/Interaction (class) name in the Federation

## 2 Super




Select the inherited Object/Interaction (parent).

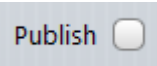
## 3 MOM


Set it to true for Management (system) object or false for user-defined one.

## 4 ID

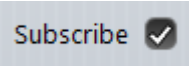

Local id of the Object/Interaction. Read-only.

## 5 Publish



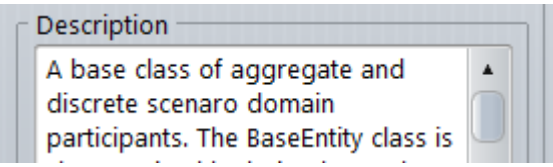

Check it if you want this Object/Interaction to be used as a publisher FedItem by the Converter.

## 6 Subscribe



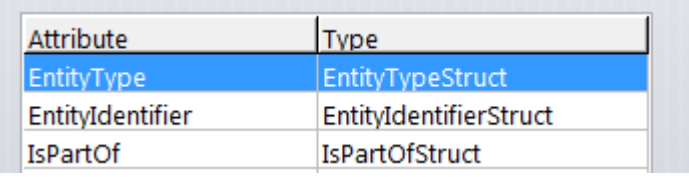

Check it if you want this Object/Interaction to be used as a subscriber FedItem by the Converter.

## 7 Description




Set the help/description of this Object/Interaction.

## 8 Attributes



| Attribute | Type |
|---|---|
| EntityType | EntityTypeStruct |
| EntityIdentifier | EntityIdentifierStruct |
| IsPartOf | IsPartOfStruct |

List of all the attributes of the Class and the associated type.
You can double click the entry for edition.

## 9 Edit

Double click the Attribute/Parameter in the list or use this button to Edit it.

## 10 Add

Use to add a new Attribute/Parameter in the List.

## 11 Sort

Sort all Attributes/Parameters in alphabetic way.

## 12 Arrange

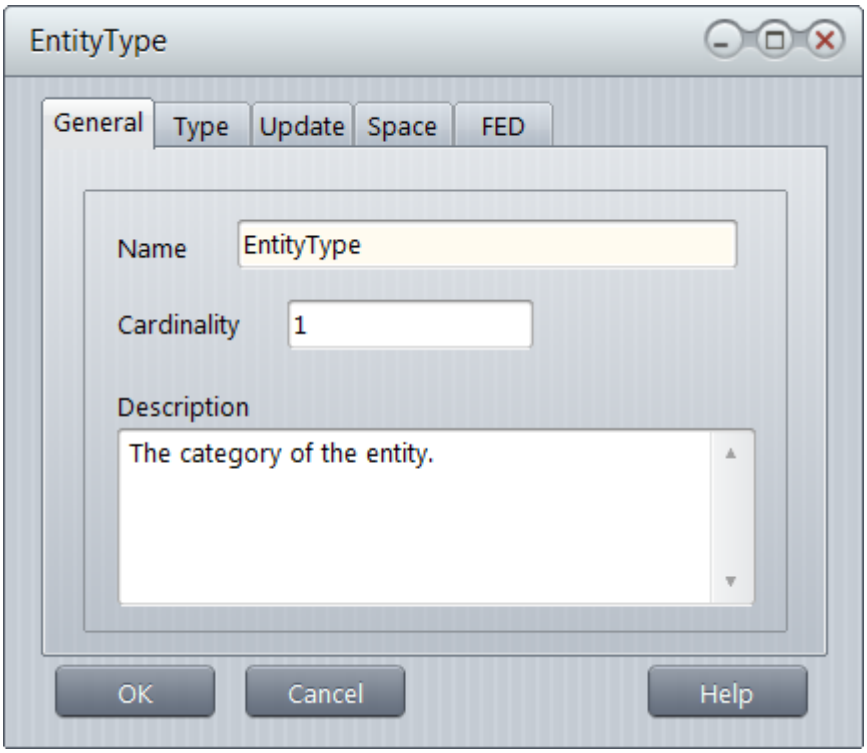
Move up or down the list the selected Attribute/Parameter.

## 13 Delete

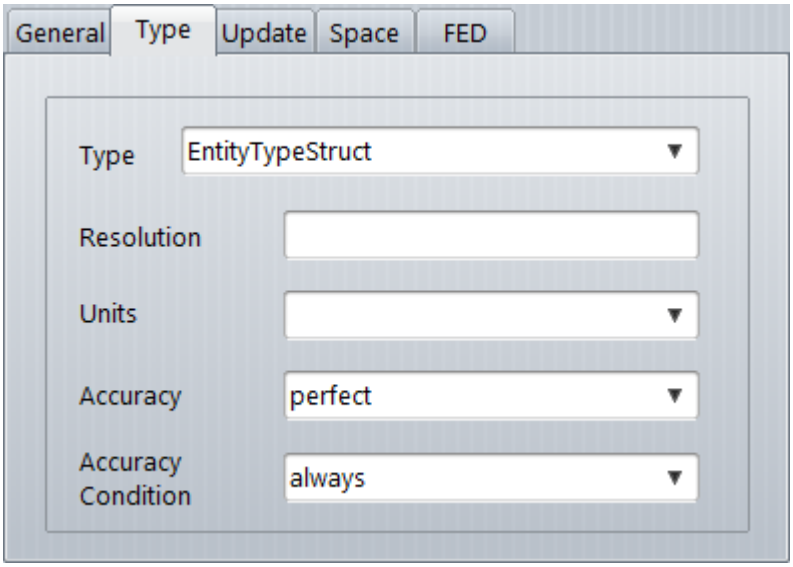
Suppress the selected Attribute/Parameter from the Object/Interaction (class).

# Object Fields

Uses to setup all requested parameters of an OMT Object Field.
Under 1516, all fields can be parameterized.
Use default values if you do not know the proper setting.

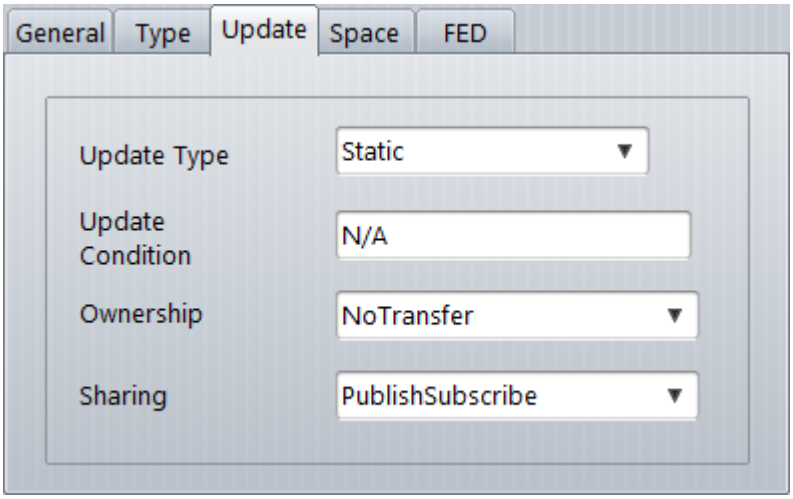



| | |
|---|---|
| **Name** | Mandatory field |
| **Cardinality** | This is used to record the size of an array or sequence. A designation of 1+ in this column shall allow for unbounded sequences. Cardinalities of multidimensional arrays shall include the sizes of every dimension listed in their normal order of precedence. A one (1) shall be entered in this column for any attribute that is composed of a single instance |
| **Description** | whatever text to describe the Attribute/Parameter |



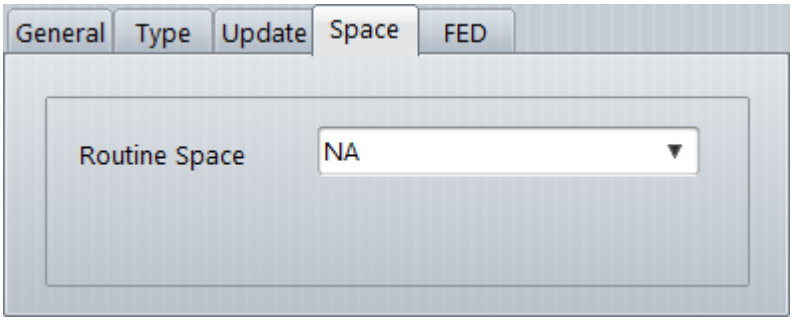

| | |
|---|---|
| **Type** | Mandatory field. Select the type from the list built based on the Enum, Complex, Variant, Array, Basic and Simple types. |
| **Resolution** | This entry may have different kinds of entries, depending upon the kind of parameter. For parameters of scalar numerical measures, the resolution column may contain a single dimensioned numeric entry for each row of the table. This value may specify the smallest resolvable value separating parameter values that can be discriminated. However, when such parameters are stored in floating point data types, their resolution so defined might vary with the magnitude of the parameter value. Hence, in these cases and others, a better sense of the resolution may be conveyed by the data type. |
| **Units** | if available in the FOM, select it from the drop down list, N/A otherwise (can be left blank) |
| **Accuracy** | This entry is intended to capture the maximum deviation of the parameter value from its intended value in the federate or federation. This is ordinary expressed as a dimensioned value, but it may also be perfect for many discrete or enumerated parameters |
| **Accuracy Condition** | This shall contain any conditions required for the given accuracy to hold in a given simulation or federation execution. It may consist of reference to a particular type of update algorithm that determines the accuracy, or it may be an unconditional always. |

### Object Fields



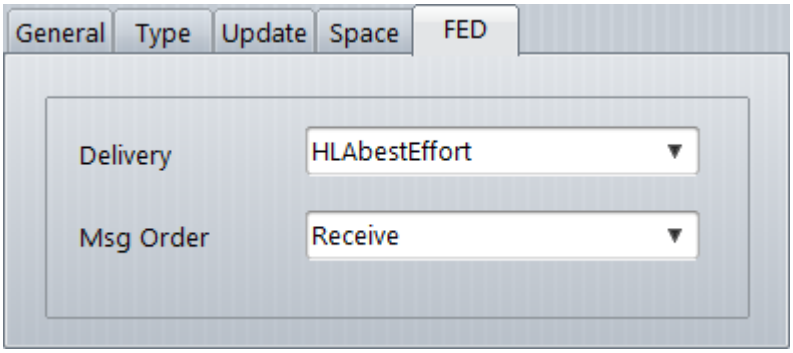

Defines the way the value must be updated by the RTI, in a Publish, Subscribe or both modes.

| | |
|---|---|
| **Update Type** | N/A, Static, Periodic, Conditional |
| **Update Condition** | Specify here condition if **Type** is set to Conditional. Mostly literal information |
| **Ownership** | Divest, Acquire, Divest-Acquire, No Transfer |
| **Sharing** | Publish, Subscribe, Publish-Subscribe, Neither |




For Data Distribution Management only.

| | |
|---|---|
| **Routine Space** | This column shall record the association of an interaction parameter with a routing space if a federation is using DDM services. It shall contain a routing space name from the routing space table. For SOMs, or for FOMs in which the federation is not using DDM services, N/A shall be entered into this column<br>Federate, Service Group, D1, D2, D3, D4, D5, D6 |



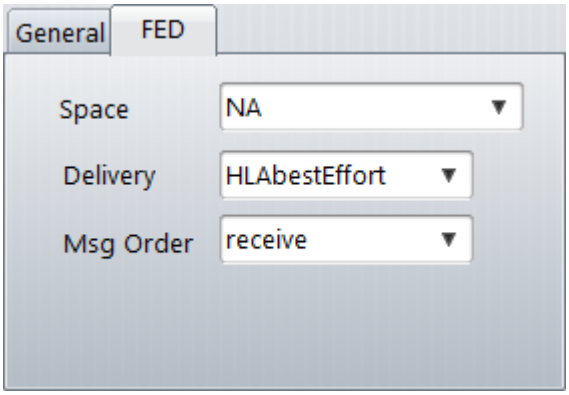

For 1.3, use FED panel only.

| | |
|---|---|
| **Delivery** | Reliable, Best Effort |
| **Msg Order** | Time Stamp, Receive |

# Interactions

See here for the general settings of the Interaction panel.

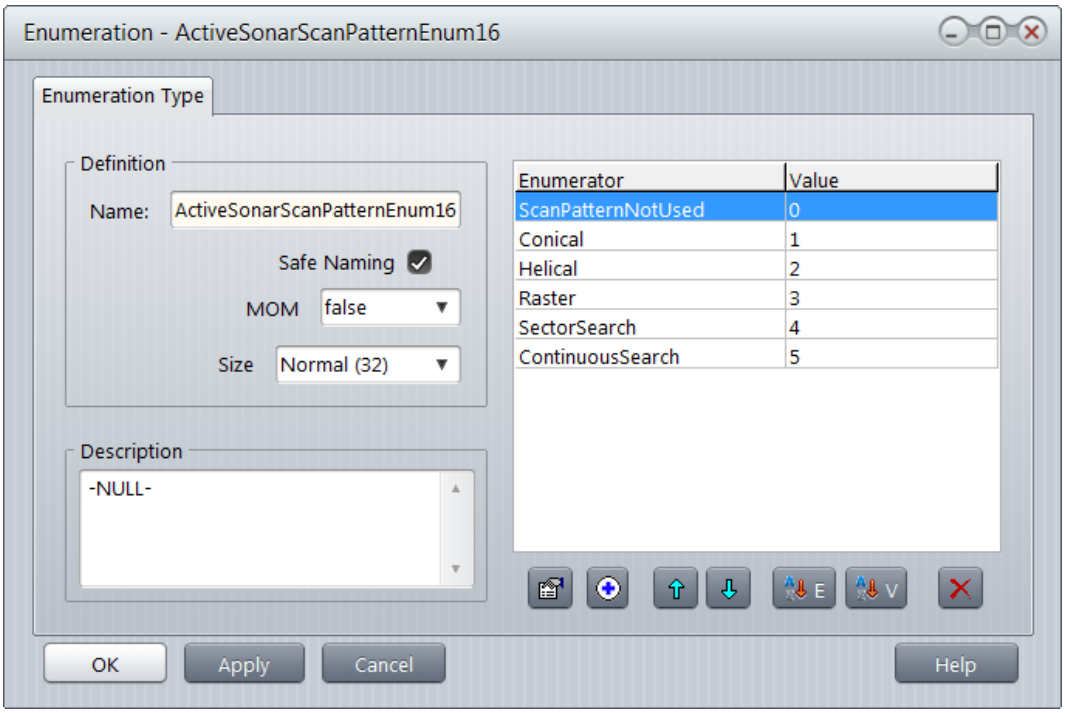
An Interaction is seen as a Complex data structure that cannot been divided in terms of Parameters by the RTI.
Thus, this panel specifies the RTI distribution mode for the Interaction seen as a whole as Parameters DDM settings are all grayed out.




| | |
|---|---|
| **Space** | Federate, Service Group, D1, D2, D3, D4, D5, D6 |
| **Delivery** | reliable, best effort |
| **Msg Order** | time stamp, receive |

# Enums

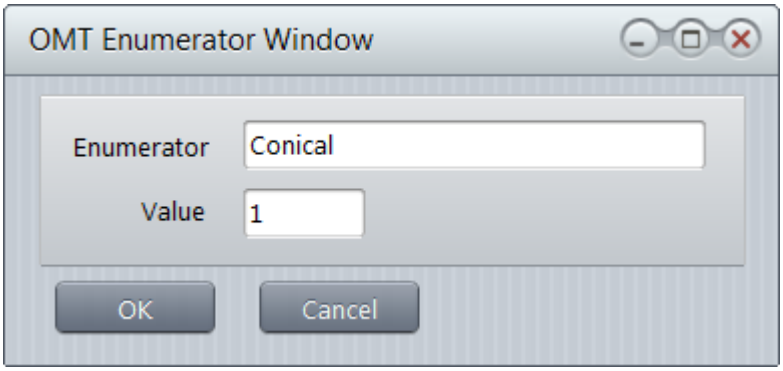
Used to define an Enumeration:




- to add a new Enumerator (entry) in the List. Double click the entry or to open and edit it (see below).
- & to change position of the selected Enumerator in the List.
- to delete the selected Enumerator.

| | |
|---|---|
| **Name** | Enumeration name in the Federation. |
| **MOM** | Set it to true for Management (system) Interaction or false for user-defined one. |
| **Safe Naming** | By default, every Enumerator (label) will be prefixed with `e#_` where `#` will be a unique number. This will insure uniqueness of the label. If unchecked, Enumerator label will be used as it. |
| **Description** | Set the help/description of this Class. |

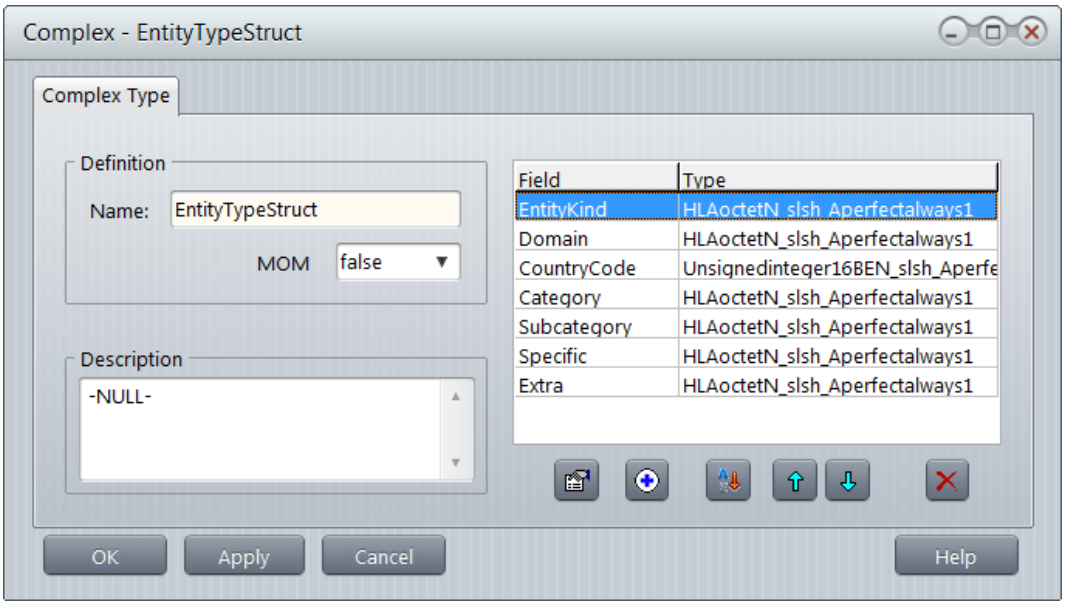
### Enums




Give for each enumerator the label name used inside the C++ `struct enum`

# Complexes

Used to define a Complex structure and its fields:




- to add a new Field in the List. Double click the Field or to open and Edit it.
- & to change position of the selected Field in the List
- to delete the selected Field.

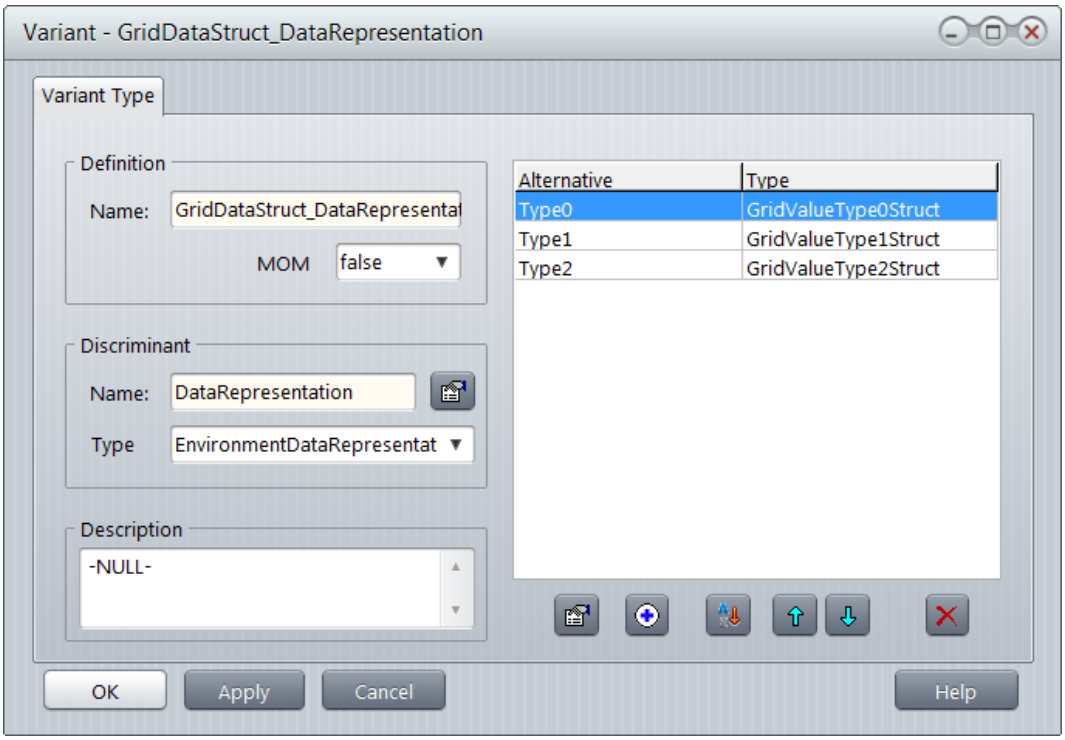
| | |
|---|---|
| **Name** | Complex name in the Federation. |
| **MOM** | Set it to true for Management (system) Interaction or false for user-defined one. |
| **Description** | Set the help/description of this Class. |

Complex type will be translated into C++ as a `typedef struct`

Each **Field** describe a Parameter like entry.

# Variants

Used to define a Variant record data structure with Alternatives:




- to add a new Alternatives in the List. Double click the Record or to open and Edit it.
- & to change position of the selected Alternatives in the List
- to delete the selected Alternatives.

## • Definition

| | |
|---|---|
| **Name** | Interaction name in the Federation. |
| **MOM** | Set it to true for Management (system) Interaction or false for user-defined one. |

## • Discriminant

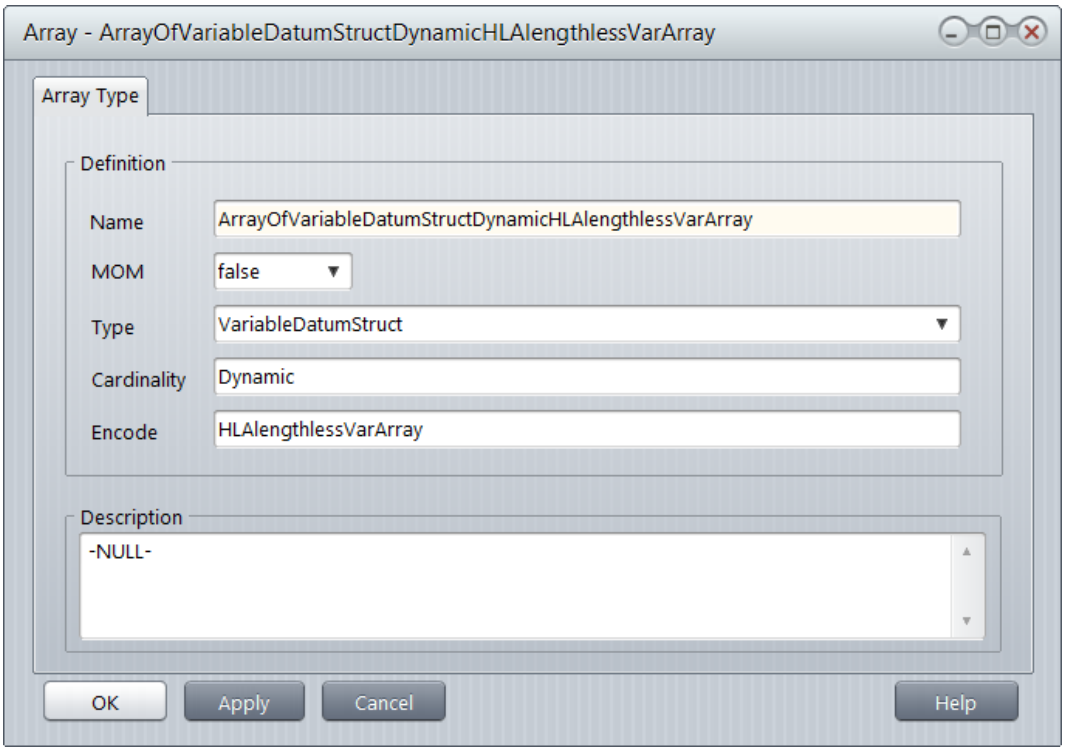
| | |
|---|---|
| **Name** | Discriminant name. |
| **Type** | Discriminant Enum type to select from the existing list. |

Variant Records will be translated into C++ as `typedef union`

Each **Alternative** describe a Parameter like entry.

# Arrays

Define an Array data type for the Federation:

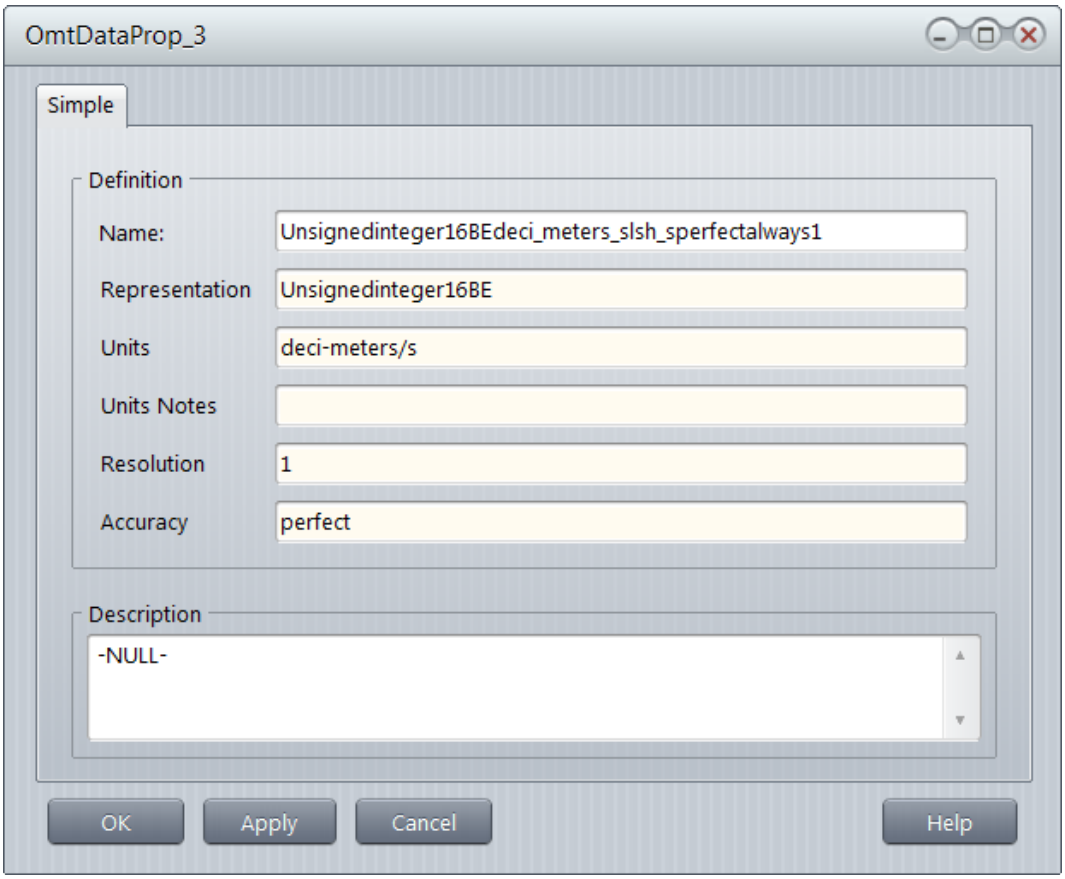



Arrays are translated into C++ as `Type* Name` or `Type Name[Cardinality]`

| | |
|---|---|
| **Name** | Name of the Array in the Federation. |
| **MOM** | Select here if this array is part of the MOM class. |
| **Type** | List of all the Complex and Basic types defined in the OMT. |
| **Cardinality** | Dynamic or Static. |
| **Encode** | Type of encoding (default HLAlengthlessVarArray). |
| **Description** | Help part for this array. |

# Simples

Definition of simple (scalar) data types to be used in the Federation:



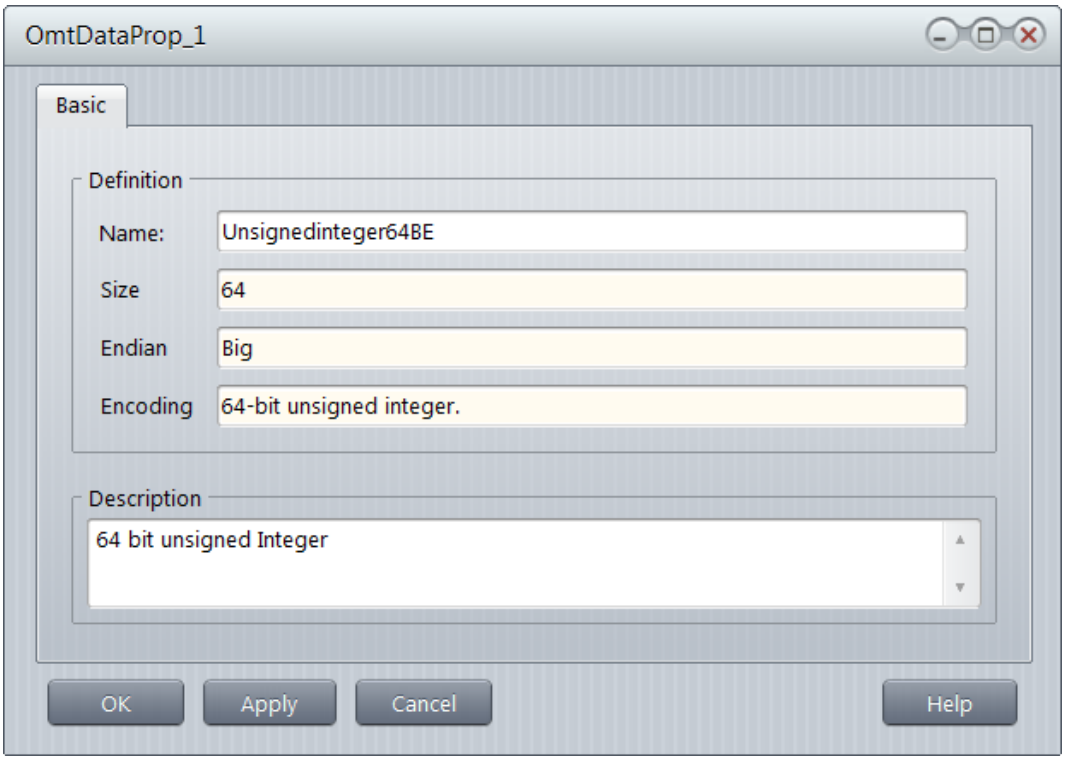

| | |
|---|---|
| **Name** | name of the Array in the Federation. |
| **Representation** | put here a basic type. |
| **Units** | The Units column shall contain the units (e.g., m, km, kg) used for each attribute whenever such units exist. |
| **Units Notes** | description of the unit, if given. |
| **Resolution** | This entry may have different kinds of entries, depending upon the kind of parameter. For parameters of scalar numerical measures, the resolution column may contain a single dimensioned numeric entry for each row of the table. This value may specify the smallest resolvable value separating parameter values that can be discriminated. However, when such parameters are stored in floating point data types, their resolution so defined might vary with the magnitude of the parameter value. Hence, in these cases and others, a better sense of the resolution may be conveyed by the data type. |

### Simples

| | |
|---|---|
| **Accuracy** | This entry is intended to capture the maximum deviation of the parameter value from its intended value in the federate or federation. This is ordinary expressed as a dimensioned value, but it may also be perfect for many discrete or enumerated parameters. |
| **Description** | Help part describing the purpose of this array. |

# Basics

Definition of basic data types to be used in the Federation:

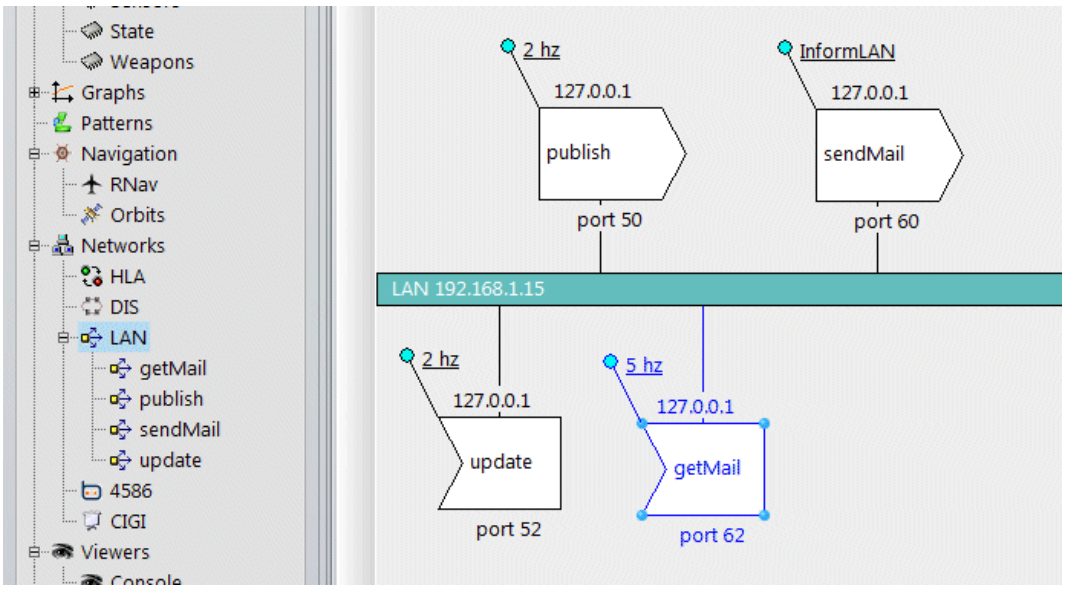



| | |
|---|---|
| **Name** | Name of the basic type as it will be used by other typed parameters of the OMT |
| **Size** | Number of bits used to encode the value |
| **Endian** | Big or Small |
| **Encoding** | Specify the packing of the data including the sign |
| **Description** | Help part describing the purpose of this basic type |

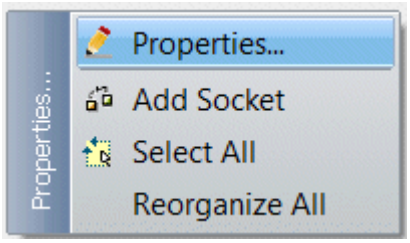
# Sockets

Sockets are a simple and easy mechanism to connect the simulation engine to a network and exchange information in real-time.
Several SockItems (represented by a symbol on the diagram) can be used, each of them connected to a real LAN socket (under a given port and a specific protocol), one for sending and one for receiving.


For each of them, the data process of the data to send or received is located inside each SockItem, making the development, change and maintenance easier.
Also, during runtime, each of the socket icon will change color according to its state (connecting, connected, cannot connect, sending data, receiving data...) allowing an quick and easy monitoring of what is happening on the network.




## • Popup Menu




**Properties**: call the LAN setup window

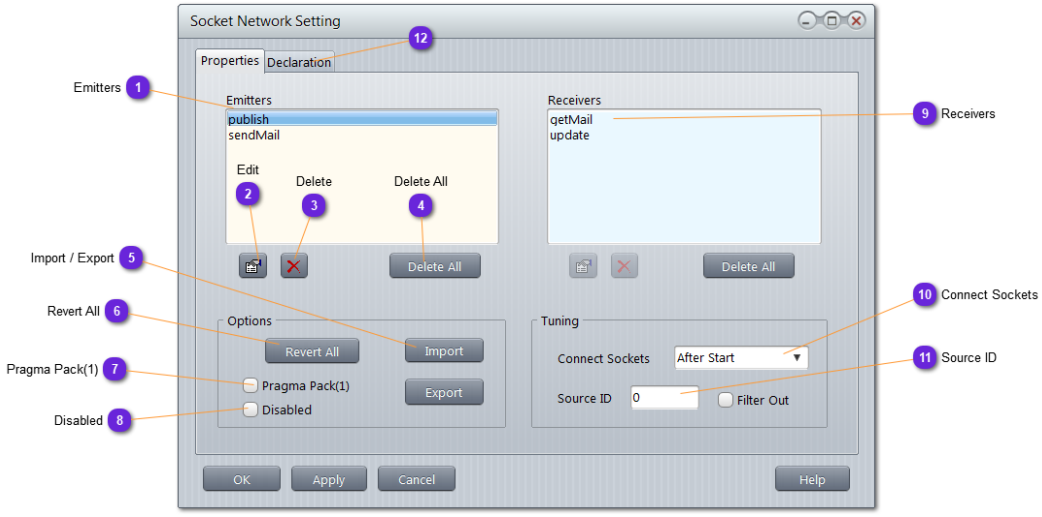
**Add Socket**: will create a SockItem as an **Emitter** or **Receiver** depending on the area created (**top** for Senders, **bottom** for Receivers)

**Select All**: select all SockItem symbols (useful for deletion)

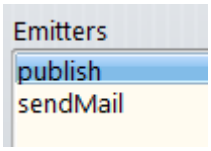
**Reorganize All**: tries to arrange all symbol in an ordered manner.

Refer to the **Tutorial document** to learn how to develop your first simulation using LAN sockets.

# Lan Setup




## 1 Emitters




List of all the SockItems in emitting mode.

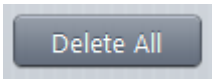
## 2 Edit

Call the selected SockItem property window.

## 3 Delete

Remove the selected SockItem of the list.

## 4 Delete All



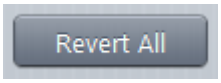

Remove all SockItems of the above list.

## 5 Import / Export

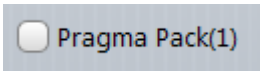

**Import** will save all the current SockItems and the LAN setting (including declarations) to /Data/Shared directory. Extension is lan.
That can be useful for reusing a specific communication package or saving the current one to be replaced by another stored one.
Use **Export** to store the actual LAN setting (including declarations and all SockItems) into a package file for later import into another database.

*Importing a new package will erase all SockItems and replace the current LAN settings.*

## 6 Revert All




## 7 Pragma Pack(1)




If checked, will add the `#pragma pack(1)` macro into the header file to force data structure packing to **1** byte so that the size of the exchanged structures with other computer will remain the same.

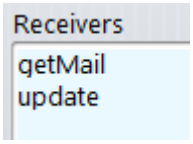
*This packing will be done on user data classes inheriting from MsgData (see Engine/vt_sockets.h). MsgData is already packed and sized 8 bytes. The first 2 bytes are for the source, the following 2 bytes are for the type and the last 4 bytes are for the (total) length of the message.*
*So, if a MyMsg class inheriting from MsgData contains a short value, the size of MyMsg will be the following:*

- *Without pragma pack(1): 8 + 4 = 12 bytes*
- *With pragma pack(1): 8 + 2 = 10 bytes*

## 8 Disabled

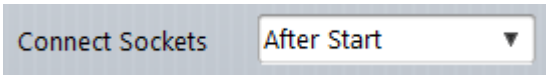

If checked, the LAN capability will be deactivated. No communication will be established. Faster than disabling all SockItems.
Useful to mute the simulation engine from the distribution network (if running without socket communication is allowed).

## 9 Receivers




List of all the SockItems in receiving mode.

## 10 Connect Sockets




Specify when the sockets must be created and connected to their port.

- After Start: when the simulation is run
- Before Start: after the simulation is loaded but before it is started.

## 11 Source ID


- If Source ID is **0**, this option is **ignored**: outgoing messages will not be stamped. If not zero, all outgoing messages will be stamped with the value (first byte of the header).
- If Filter Out is **checked**, all incoming messages with same Source ID **will be ignored but removed from the socket buffer !**

.

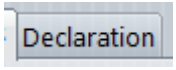
*This is useful when a socket writes to the same port as a listener, to avoid picking the sent data.*
*In order to process only messages from a specific source, add in the Socket Receive top part the following code:* `if (msg->source == whatever) ...`

**Deprecated old logic (copied here to help migrating your code) :**

- *If Filter Out is checked, only the incoming messages whose Source ID differs from the current Source ID value will be kept and processed. All with same ID will be discarded.*

• *If Filter Out is not checked, only the messages with the same Source ID will be kept and processed.*

## 12 Declaration

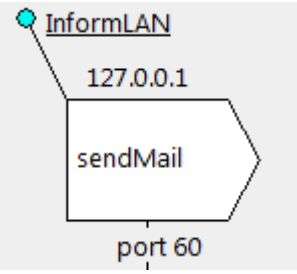



Add here all your message data classes using the following format:

```
class my_data : public MsgData
{
  public: my_data() { size = sizeof(*this); type = 123; }
};
```

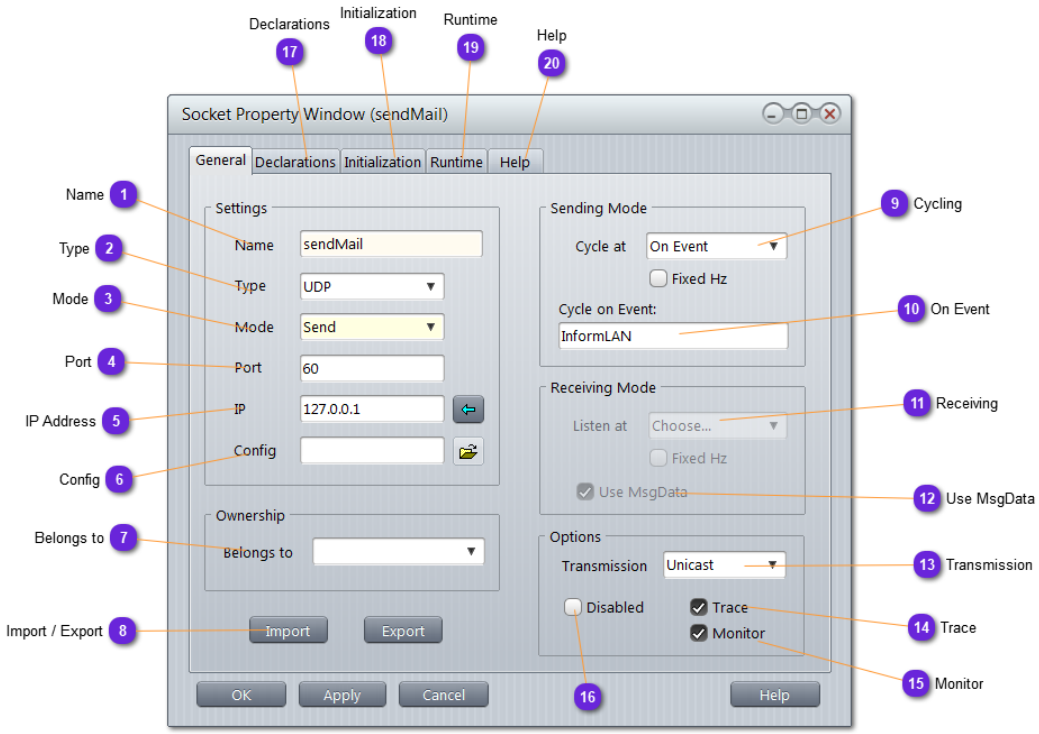
where 123 is the class type integer of your choice.
add after your own data.

# Socket Property

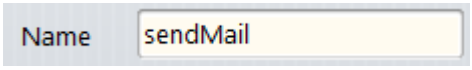



A Socket object in vsTASKER is a convenient way to send or receive data using the network card and TCP/UDP protocol.
The setting of such object remains simple and the user just has to concentrate on what to send and what to do with the received data.

*One SockItem cannot send and received at the same time. It is either a s a receiver. Different ports shall be used for now.*

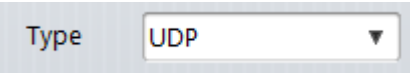



## 1 Name



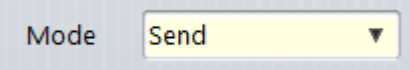

Name the SockItem. Must be unique.

## 2 Type




Select from the drop down list whether the protocol will be UDP or TCP.
UDP is a faster protocol but which is not reliable. Also, UDP can broadcast more easily than with TCP.

## 3 Mode

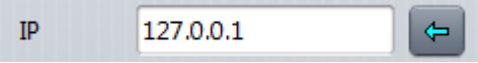



Each SockItem can either send or receive. If the mode is changed, the position of the icon on the diagram will also change.

## 4 Port


Specify the port to use. Note that there should be one port per SockItem.

## 5 IP Address




IP address which will be used to listen on a particular port.
Use  button to retrieve the current computer IP.

## 6 Config


If the setting must be done from a configuration file, so that to be adaptable from another code, without having to open vsTASKER and change the setting, specify here the configuration file that will be used by the SockItem at creating time, on the simulation engine, for setup. The file must exist and be accessible at runtime.
Content of the file shall be the following, in a text file:

```
# IP, port, cast (unicast | broadcast | multicast)
127.0.0.1, 50, unicast
```

## 7 Belongs to

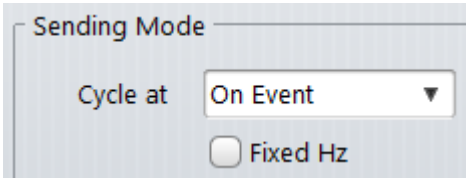

If the SockItem must use a particular entity data, select it here if it already exist in the scenario.
The variable entity will be set at runtime.
Otherwise, the following code in the Initialization panel (`RESET` phase) will do the trick:

```
entity = S:findEntity("myEntity");
```

## 8 Import / Export

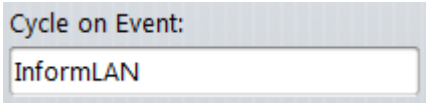

Use Import to replace the current SockItem with a previously exported one.
Use Export to save the current SockItem to a file for later retrieval or sharing.
Default location is /Data/Shared. Extension is skt.

## 9 Cycling



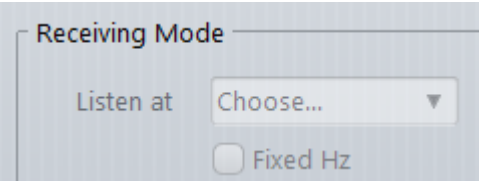

Chose the frequency at which the runtime code of the SockItem will be called. If On_Event, call happens only when the proper event is received (10).
If **Fixed Hz** is checked, the frequency will the the wall clock one and not the simulation clock.
*(only for Emitters).*

## 10 On Event



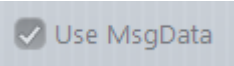

Put here all events that will trigger the runtime code for one cycle. Must be separated with comas if many.
*(Only for Emitters)*

## 11 Receiving



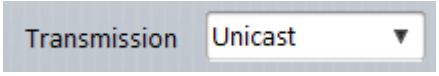

Specify here at which frequency the SockItem runtime code will be called to process the received data.
If the frequency of the receiver is lower than the one of the sender, some data will be lost as they are not pooled (on UDP). If the frequency of the receiver is higher then the one of the sender, some calls will just be skipped because of no new data available.
*(Only for Receivers)*

## 12 Use MsgData



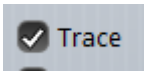

For receivers, `MsgData` can be expected as a header of the message. This is convenient for casting the message into the correct class object.
Sometime, when the sender does not inherit from `MsgData`, uncheck this option.
The runtime code will get raw data and it must handle itself the marshmalling of the data from the socket buffer.

## 13 Transmission



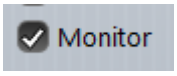

Select from the drop down list the kind of transmission the socket will operate. Bear in mind that some casting require special IP addresses.
Unicast: default, peer to peer communication. Local 127.0.0.1 or any other IP will do.
Multicast: unique emitter and multiple receiver. Multicast addresses range from 224.0.0.0 to 239.255.255.255. Receivers must register to the multicast group address. (UDP only)
Broadcast: all nodes of the subnet will see the traffic (UDP only).

## 14 Trace




When checked, the console will output all connection success failure, data entry or sending, and SockItem states.

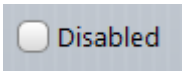
## 15 Monitor




When checked, the SockItem icon will change color according to the inner state sent by the simulation engine:

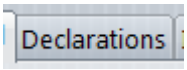
- Greyed out: disabled
- Green: socket created, no data
- Magenta: data passing through
- Red: connection failed.

## 16 Disabled




If checked, the SockItem will not be created and there will be no communication performed (nor the socket created).

## 17 Declarations




Put in this panel all **data** you want to add in the current SockItem. Public data is used to generate the Dyn-UI so, consider carefully what you put in the `Public:` data section.


*No constructor and destructor should be added here. Use instead the Initialization panel.*

## 18 Initialization

Initialization

This part is called automatically at instantiation (`INIT`), when SockItem is created using `new`, or every time the object is `RESET` and finally at destruction (`CLEAN`) using `delete`.

*It is a good practice to put in the `RESET` part whatever must be reinitialized at reset. There is no break after `INIT` to allow the `RESET` part to be processed after `INIT`, at Socket creation time.*

## 19 Runtime

Runtime

This part is called by `tic(Event*)` at the specified frequency. It is an automatic routine of the SockItem so, all data of the SockItem can be accessed here.

- **Senders**: if triggering mode is Cycle on Event, use event of type `Event*`. if Cycle at set, no argument is available. Use `scen()` to get the data you need.

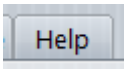
- **Receivers**: Use `msg` of type `MsgData*` and cast it according to the type of data you are expected on this socket.

  Sometimes, the number of messages are numerous and to not wait for the next cycle to read the queued ones, use `return AGAIN;` but beware of blocking the simulation if continuous messages are coming. Better put a limit in the number or `return AGAIN` allowed per cycle.

General | Declarations | Initialization | Runtime | Help

```
// If socket mode is:
// Receive: use msg of type MsgData* and cast it.
// Send   : if "Cycle on Event" set, use event of t
//        : if "Cycle at" set, no argument availabl


  if (getEvent()->isName("InformLAN")) {
     msg = *(CommandData*) getEvent()->data;
     send(&msg);
  }
```

## 20 Help




Any description of the SockItem can be put here and will be used for the automatic document generation (see Make Documents)

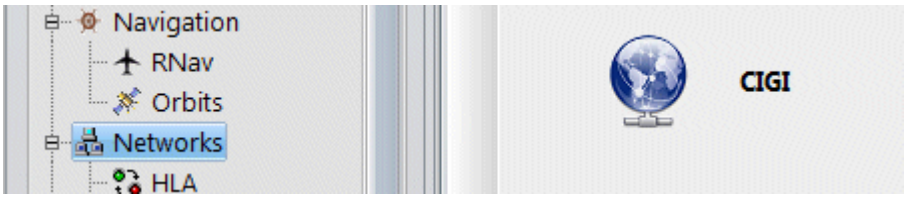
# CIGI

The Common Image Generator Interface (CIGI) is a standardized interface between a real-time simulator host and an image generator. CIGI is an open interface offered to promote commonality in the visual simulation industry. vsTASKER is using the standard library and version 3.3 interface, available at http://cigi.sourceforge.net

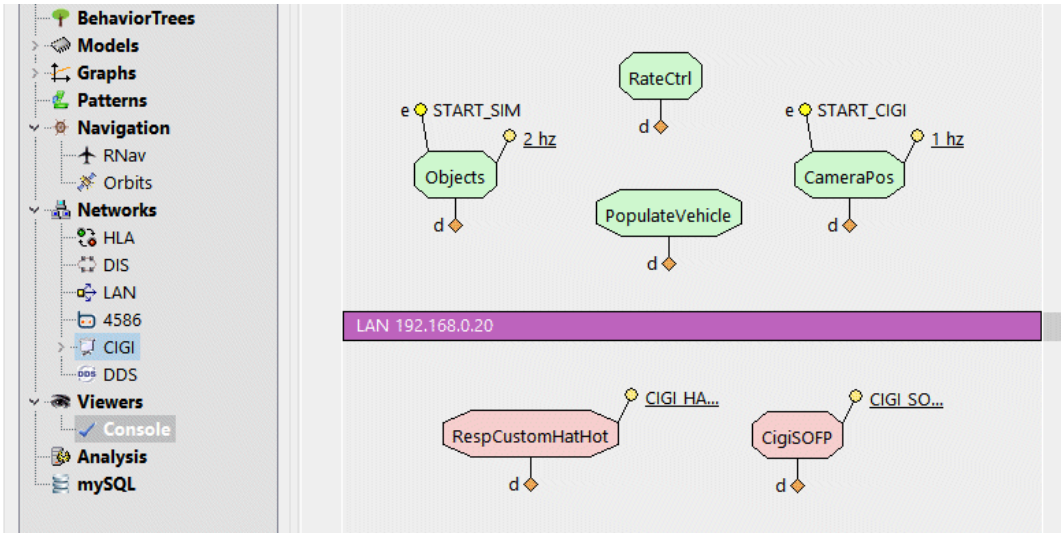


## • Packets

Select Networks, then CIGI for tuning the CIGI host.



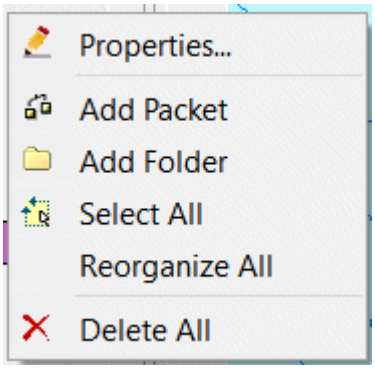

In the Environment, click on CIGI to access the data packets (Packets) definition pane




The area is made of two sections separated by the magenta line. **Upper** section is for the **Sent** packets (Host to IG) and **Lower** section is for the **Received** packets (IG to Host).

## • Popup Menu

In the CIGI Packets definition panel, right click on the background:




**Properties**: call the settings window

**Add Packet**: create a Packet that will be emitting or receiving data according to its position on the diagram (**top** for senders and **bottom** for receivers)

**Add Folder**: create a new folder to store specific Packets. See Folders to learn how to use them.

**Select All**: select all Packet symbols (useful for moving or deletion)

**Reorganize All**: tries to reorganize the Packet symbols in the diagram so that to avoid overlapping.

**Delete All**: remove all the Packets of the current view. Cannot remove folders if not empty.

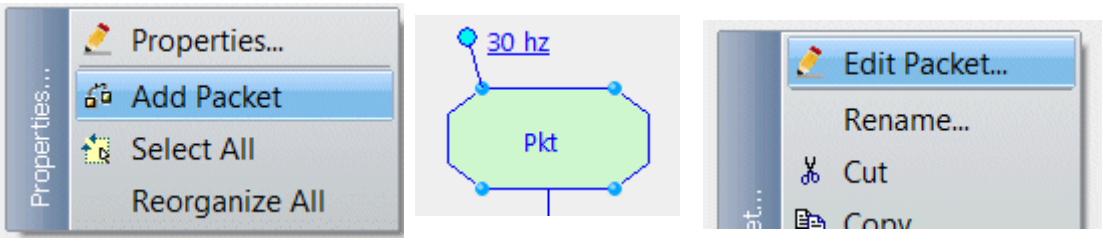
## • How to Use

First, it is mandatory to set the Host and IG IP address so that both can communicate.
In the Packet definition background, right-click and select Properties to popup the settings panel.

See here for the description of the Settings window.

CIGI obeys to fixed rules and sequences. Most of the messages from the Host to the IG do not request a response. If such is the case, because the response is asynchronous (most of the time), a mechanism must be set in place in order to treat the answer appropriately. vsTASKER provides the mechanism to process such asynchronous response without blocking the simulation.

To **Add** a Packet, right-click in the proper section: **Upper** for **Host to IG** and **Lower** section for **IG to Host** messages.




Then select it and **Edit** the Packet.

See here for the property window of the Packet.

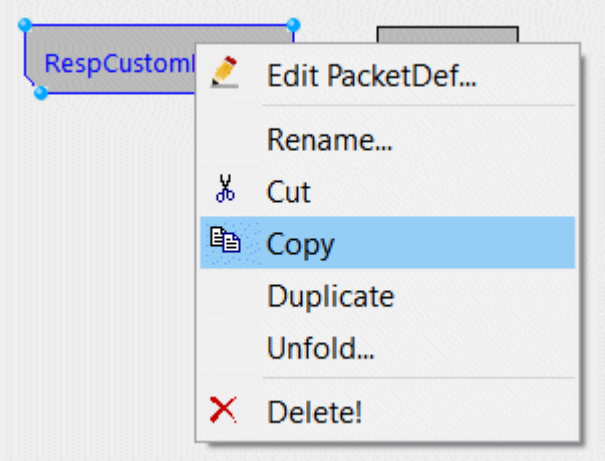
## • Definitions

A CIGI simulation may not need to use all the message packets listed in the standard. The Definition part acts like a repository for available packets. The list can be imported from a database, including proprietary packets of a vendor (if any).

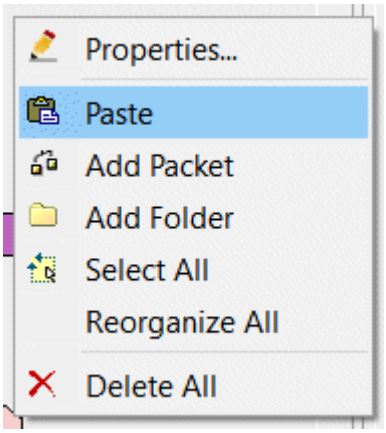
The easiest way is to import a CIGI predefined list of supported packets (Data/Shared/CIGI/default.cigi or vbs-ig.cigi)

Otherwise, create it packet by packet as explained below.

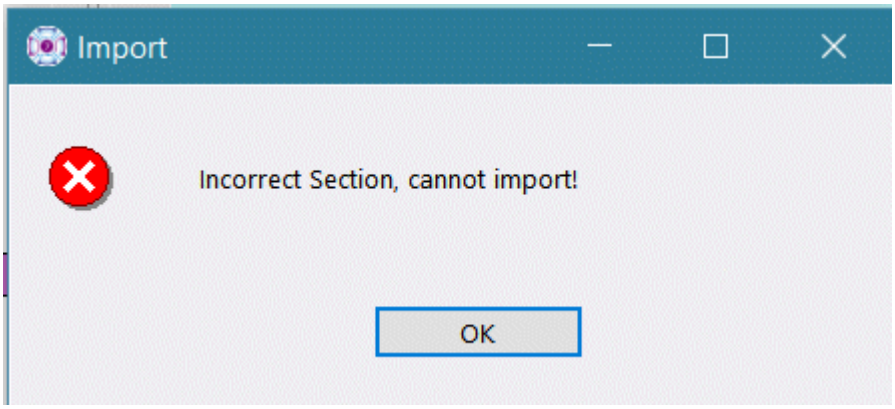
To instantiate a definition packet, select it, **Copy** it:




Then **Paste** it in the proper section of the Packets panel (and inside the proper folder if needed)



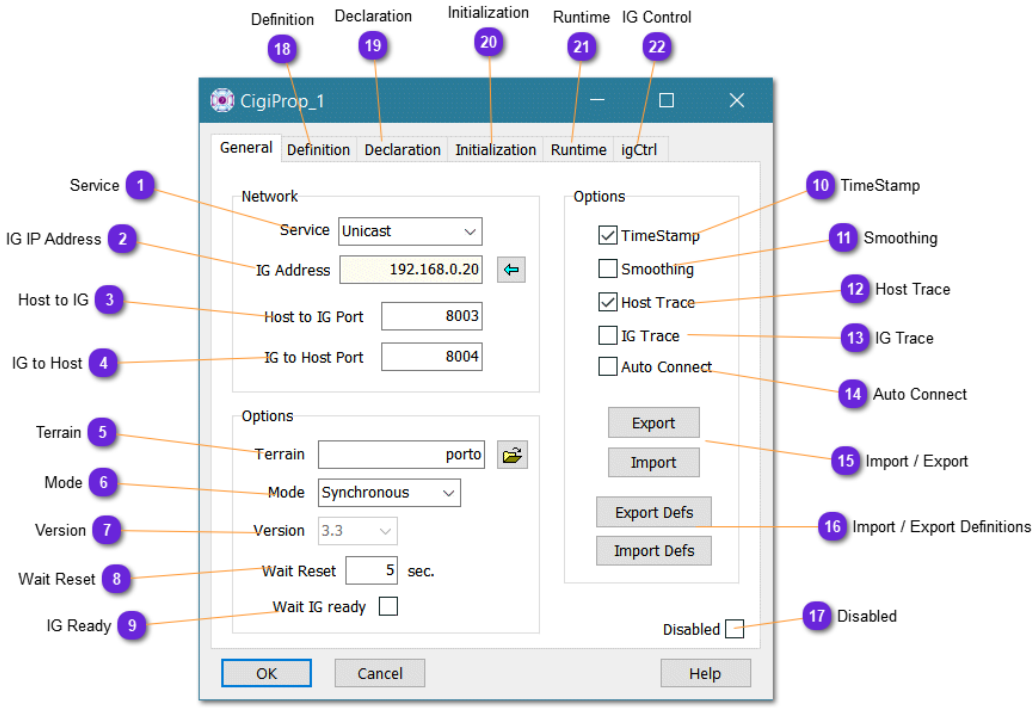

**Pay attention to the Mode** of the Packet definition. A **Sending Out To IG** mode can only be pasted into the **Send** section (top) while a **Receiving from IG** can only be pasted into the **Receive** section (bottom). Failing to do so will result in an error message:




*As the number of packets can become big, the display zone will not be enough and will end up messy.*
*It is a good practice to create folders (which can be embedded) to group similar packets together.*

Refer to the **Tutorial document** to learn how to develop your first CIGI simulation.

# Settings

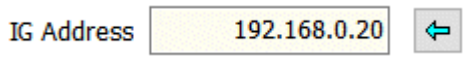



## 1 Service


Select the kind of transmission the socket must use in emission:

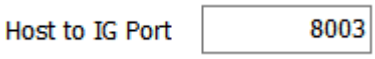
- **Unicast**
- **Multicast**

## 2 IG IP Address



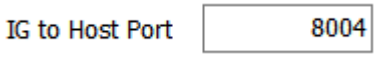

Set here the network IP address to use for sending the data packet to the IG. The ⇦ button retrieves the current computer IP, when the IG is running on the same computer as the host.

## 3 Host to IG




Specify here the port used to write data packet to the IG.

## 4 IG to Host




Specify here the port use to read data packet from the IG.

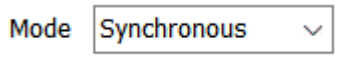
## 5 Terrain


Select from the list the database (terrain id) to send to the IG at startup for load.
The selected file must be text with the list of id and terrain name, according to the format: id=name:
**1=porto**
**2=rahmadi**
etc.

The terrain definition file must have .def extension.

Use  to load it.

## 6 Mode




CIGI supports both Asynchronous and Synchronous modes:

Asynchronous: the Host sends data to the IG at a predetermined rate. The message may occur in response to a system timer or some other event within the Host. The IG may in turn send messages to the Host containing IG status and mission function data; however, the interval between host messages is determined by the host itself.

Synchronous: the IG sends a start-of-frame (SOF) message to the Host to signal the beginning of each frame. This message, which is usually driven by a vertical sync signal from the display system, functions as a "heartbeat" that dictates the timing of data transfers between the IG and Host. The SOF message also contains mission function responses, event notifications, and other IG data.

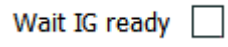
## 7 Version


Select here the version intended to use.
The code generator will use this information at best to produce an API that will support the version.

## 8 Wait Reset

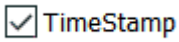

Time in second between the database/terrain **load** message and the **reset** message.

## 9 IG Ready



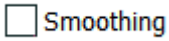

If checked, the host will wait for the IG to be ready (Control Frame), otherwise, will consider the IG ready and the simulation engine moves on.

## 10 TimeStamp



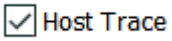

When checked, every packet is time stamped with host time.

## 11 Smoothing



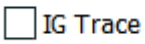

Option for the IG, if it does support entity packets extrapolation (or interpolation between two entity state packets).

## 12 Host Trace



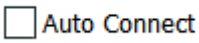

When checked, host will output on the console and cigi_out.txt file (application root directory), information about every packet sent.

## 13 IG Trace




When checked, host will output on the console and cigi_out.txt file (application root directory), information about every packet received.

## 14 Auto Connect




When checked, vsTASKER will automatically start CIGI connection at startup. Otherwise, the connection will be initiated by the simulation engine.

## 15 Import / Export

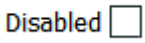

**Export** will save all the current Packets and the CIGI settings (including user code) into /Data/Shared directory. Extension for the package is cigi.
That can be useful for reusing a specific communication package or saving the current one to be replaced by another stored one.
It is a good idea to export a CIGI configuration to be shared by other future databases.


*Importing a new package will erase all current packets and replace the CIGI settings (including the code) with the package ones.*

## 16 Import / Export Definitions


**Import Defs** will save all the current Packets and the CIGI setting (including declarations) to /Data/Shared directory. Extension is cigi.
That can be useful for reusing a specific communication package or saving the current one to be replaced by another stored one.
Use **Export** to store the actual CIGI setting (including declarations and all Packets) into a package file for later import into another database.

## 17 Disabled

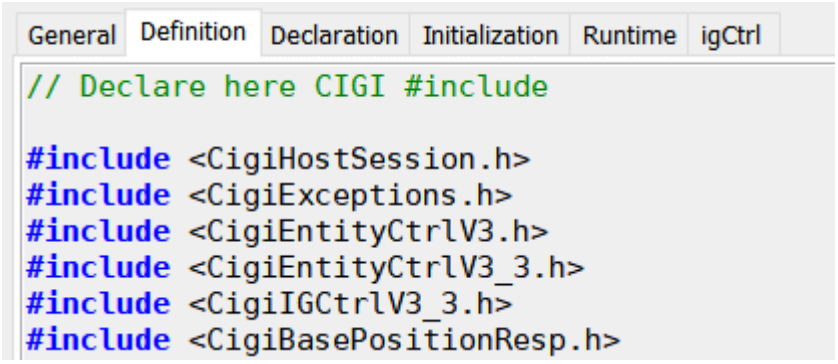



Check this to stop CIGI and mute all communication both ways.

## 18 Definition

Definition

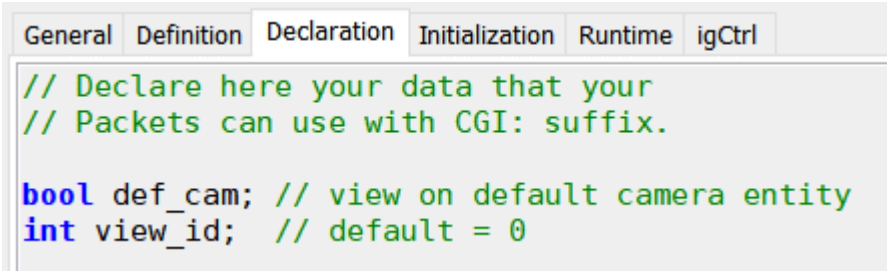
Add here all the packets includes needed for the compilation:




## 19 Declaration

Declaration

List here all the CIGI global data to be used by any Packet at runtime.




## 20 Initialization

Initialization

Initialize your CIGI global data here but not only. Use the normal runtime events to perform any specific action you need.
See System Phases/Events chapter in the **Developer Guide**.

## 21 Runtime

Runtime

Will be called at every cycle. Put the code you want here.

# 22 IG Control

igCtrl

Use whatever method of the `CigiIGCtrl` class to change the way the message header is set every time a bench of messages must be send.

```
igCtrl.SetSmoothingEn(true);
```

# Design Mode

Designing a scenario in 2D using a map is great and easy but does not offer all the benefits of a 3D positioning. Line of Sight, coverage, features not displayed on the map might become a problem once the simulation is running and the IG is used.

To solve this issue, vsTASKER offers the CIGI at design time : IG can be turned ON and entities can be seen positioned as they will be at startup. Using 2 screens is the recommenced setting for such mode as vsTASKER GUI map (one screen) will be used for creating and repositioning entities while the second screen will be used for the IG (with another mouse or preferably a joystick to move the camera or switch the view from one entity to one another).


*This mode need a **gamepad** in order to control the camera of the IG (zoom orientation) and other operations.*

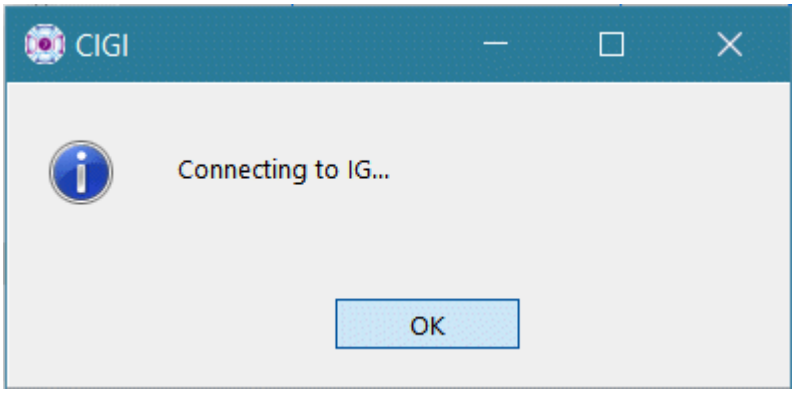

## • How to use

Start the IG then setup the CIGI configuration (ports, IP and terrain database) on vsTASKER so that the GUI can communicate (both ways) with the IG.

All the entities in the scenario must have the CigiEntity component attached and setup with the proper Type so that to be visible on the IG, the same way as during runtime.

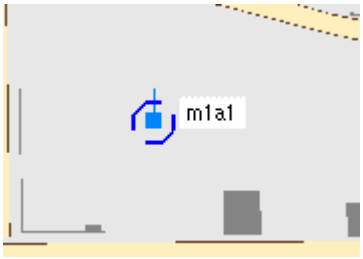
Use the following toolbar button to activate the mode (or from the menu **Engine::Cigi Design Mode**):

vsTASKER will then connect to the IG and wait for the terrain to be loaded and the IG to return to the standby state.




Then all defined entities will be sent.

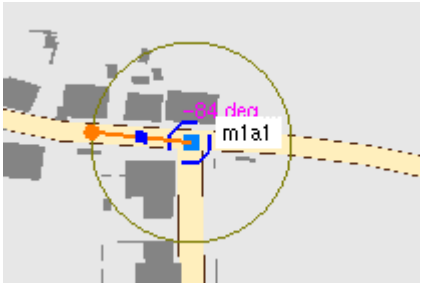
Now click on one entity of the map:




The IG will move the camera to focus on the selected m1a1.
Use the gamepad to turn around (left thumb stick) and zoom in and out using the triggers (left and right).

### Design Mode

Now change the orientation of the m1a1 on the GUI and drag it somewhere else using the mouse:

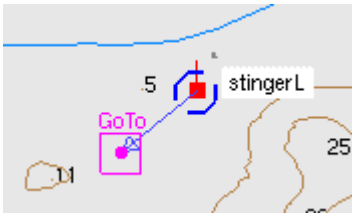


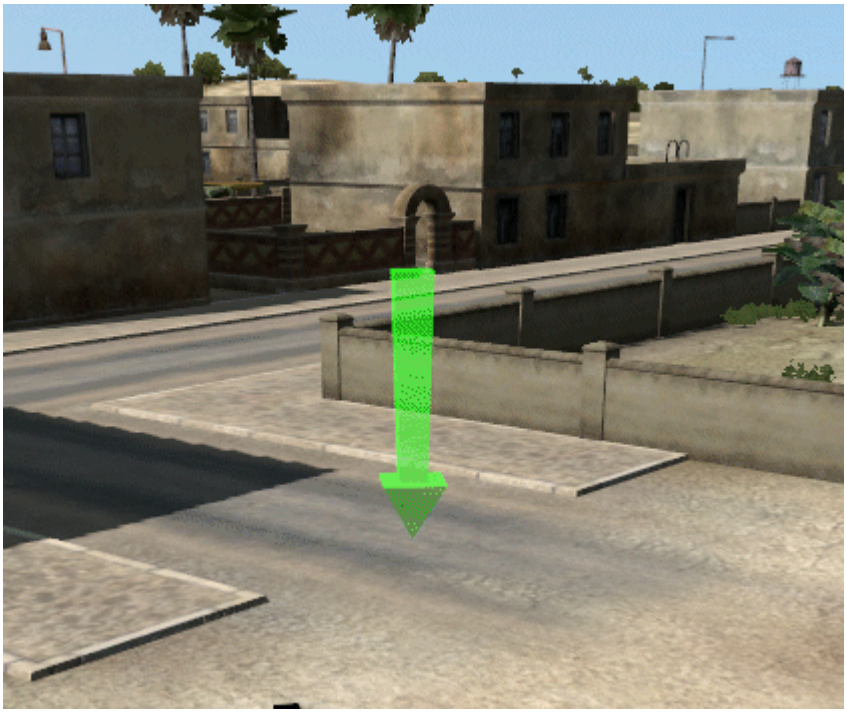

See that the entity has moved to its new position with its new orientation. Using the IG, it is much easier to check the visibility from a position.

If the entity has a plan, each waypoint will be represented using a green arrow:

On the map, selecting the waypoint and moving it on the map will be reflected on the IG view. It is then possible to accurately position it at best.

### Design Mode

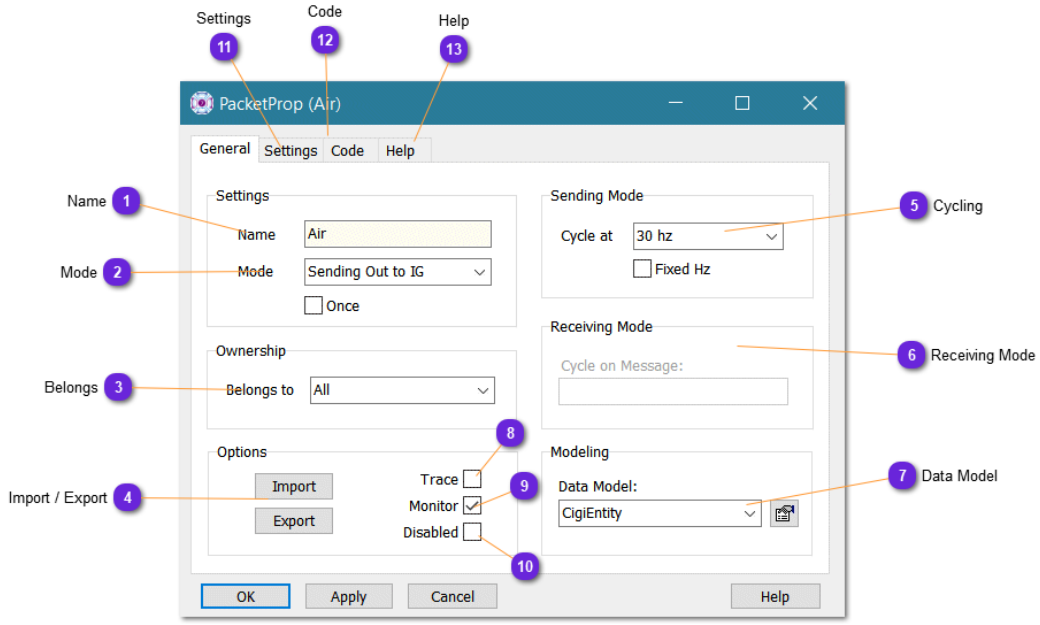


To stop the communication with the IG, just unclick the toolbar button (this will be done automatically as soon as the simulation engine is launched).


*The CIGI Design mode will see more and more functionalities added with coming versions, as this edition can be a good alternative to the embedded 3D OSG viewer, since it can directly use the target IG.*

# Packet Properties




## 1 Name

Name Air

Name of the Packet. Must be unique.

## 2 Mode

Mode Sending Out to IG

Each Packet can either send or receive. If the mode is changed, the section on the diagram the Packet belongs will also change.

## 3 Belongs

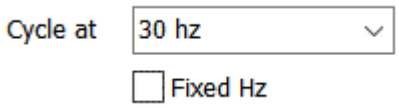
Belongs to All

If the Packet must use a particular entity data, select it here if it already exist in the scenario.
The variable entity will be set at runtime.
Otherwise, the following code in the Initialization panel (`RESET` phase) will do the trick: `entity = S:findEntity("myEntity");`

## 4 Import / Export

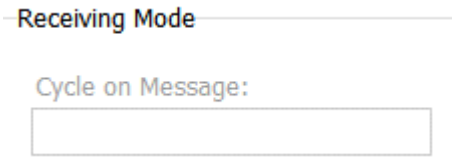

Use **Import** to replace the current Packet with a previously exported one.
Use **Export** to save the current Packet to a file for later retrieval or sharing.
Default location is /Data/Shared. Extension is .pkt

## 5 Cycling




Chose the frequency at which the runtime code of the SockItem will be called. If On_Event, call happens only when the proper event is received (10).
If **Fixed Hz** is checked, the frequency will the the wall clock one and not the simulation clock.
*(Only for Senders).*

## 6 Receiving Mode



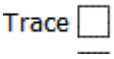

Put here the message stamp sent by the IG and that triggers the runtime code of this Packet.
i.e: `CIGI_POSITION_RESP_PACKET_ID_V3`
*(Only for Receivers)*

## 7 Data Model


If a specific Data Model must be used jointly with the Packet, it is a good idea to specify it here.
**The purpose of such entry is mainly informative**: the user knows that a specific Data Model is used by the Packet and must be attached (and setup) to the scenario Player or any Entity using the Packet.
Retrieving the selected Data Model can be done this way:

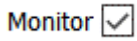
```
if (strEqual(getDataModel(),"CigiEntity")) ...
```

## 8 Trace




When checked, the console will output all data received or sent by the Packet. Tracing must be enabled globally, at the Settings level.

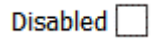
## 9 Monitor




When checked, the Packet icon will change color according to the simulation engine information:

- Greyed out: disabled
- Black: socket created, no data
- Magenta: data passing through (in or out)

## 10 Disabled

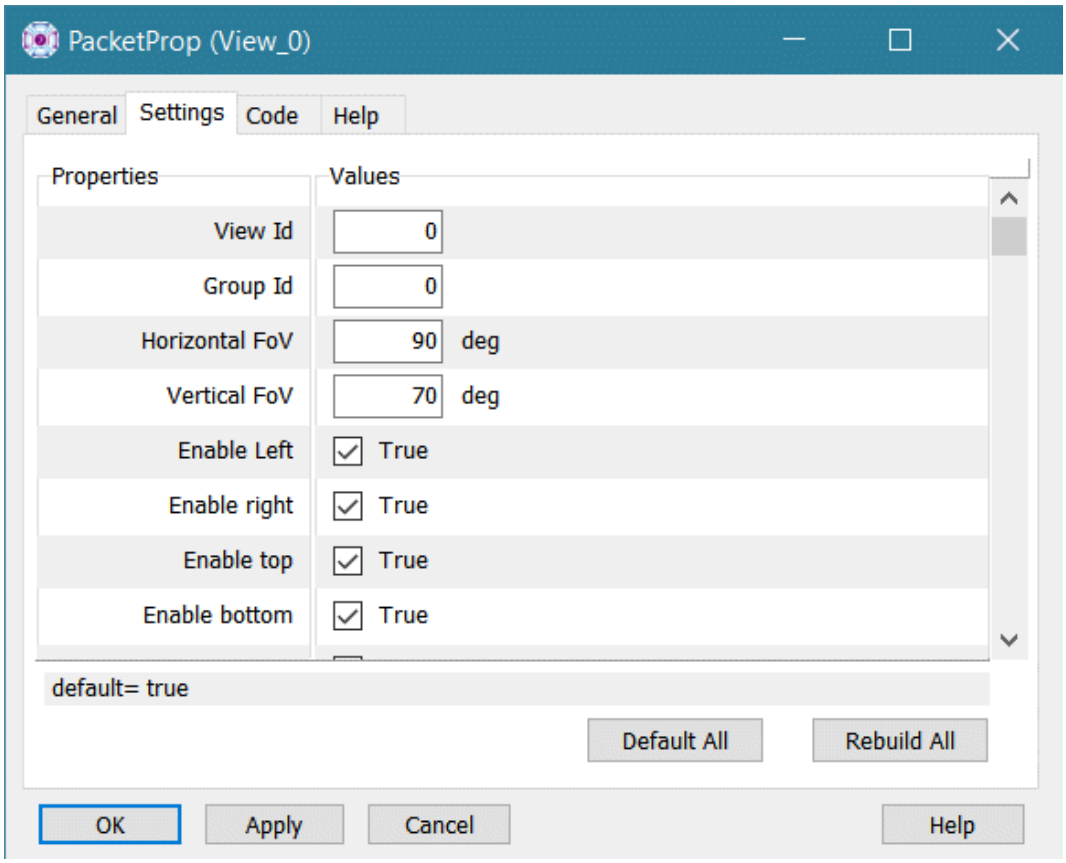



If checked, the Packet will not be created and there will be no communication performed.

# 11 Settings

Settings

This dynamic interface is the translation of the public user data defined in Code::Declaration panel (see here) with `//&&` tags
It's purpose is to provide user an easy way to setup the data of the Packet.




**Default All**: Replace all values with the default ones found in DEF[] for each entries.
**Rebuild All**: Reparse the **Code::Definitions** part to sync the Interface. This is not done automatically as the rebuild process delete the current interface before reparsing and defaulting all the values.

*In vsTASKER, a CIGI Packet object exists as one instance. It is a message producer. It can issue one message at a time (View Definition for example) or many (Entity Control) in sequence or according to some conditions for network optimization.*

## 12 Code

Code

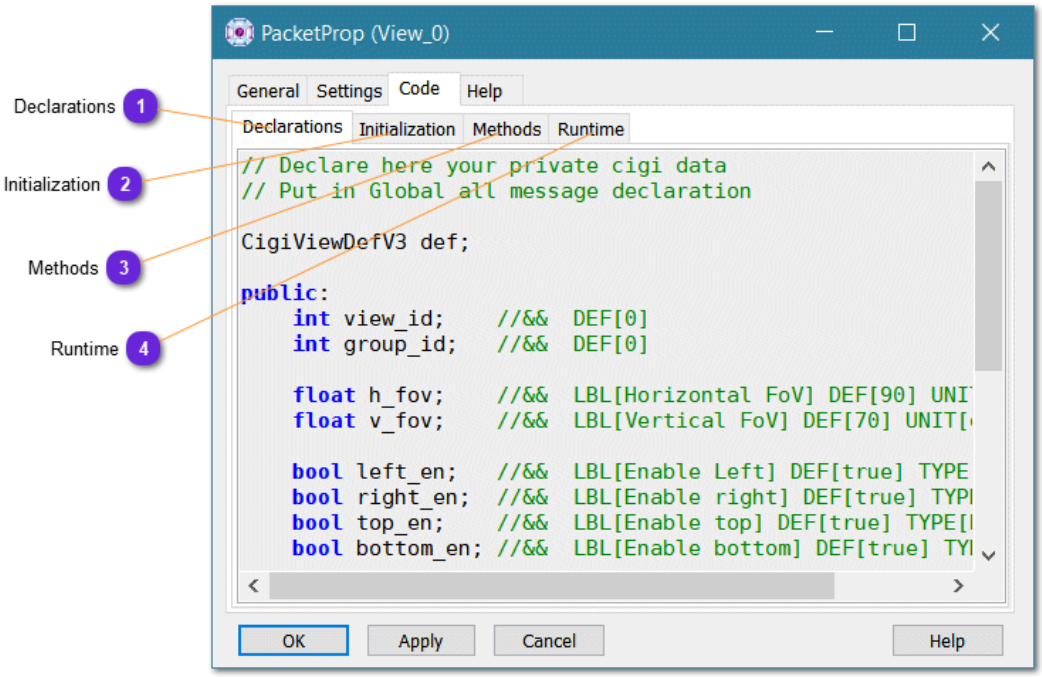
See here

## 13 Help

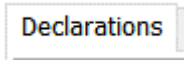
Help

Any description of the Packet can be put here and will be used for the automatic document generation (see Make Documents)

# Packet Code




## 1 Declarations




Put in this panel all **data** you want to add in the current Packet. Public data is used to generate the Dyn-UI so, consider carefully what you put in the `Public:` data section. It is also good to define all the methods that will be called from any other objects (logics, components, etc.) since the object will be instanciated once and easily accessible:

```
// Packet name: View_0
// Method defined in View_0 : mySetup()
// how to call it:

   cici.View_0->mySetup();
```


*Consider the Packet as a C++ Class which will instanciate only one object. No constructor and destructor should be added here. Use instead the Initialization panel.*

# 2 Initialization

Initialization

This part is called automatically at instantiation (INIT), when Packet is created using new, or every time the object is RESET and finally at destruction (CLEAN) using delete.

*It is a good practice to put in the RESET part whatever must be reinitialized at reset. There is no break after INIT to allow the RESET part to be processed after INIT, at Socket creation time.*

# 3 Methods

Methods

Put here all the Packet methods declared in the Declaration section.

# 4 Runtime

Runtime

This part is called by `tic(Event*)`. It is an automatic routine of the Packet so, all data of the Packet can be accessed here.

- **Senders**: if triggering mode is Cycle on Event, use event of type `Event*`. if Cycle at set, no argument is available. Use `scen()` to get the data you need.

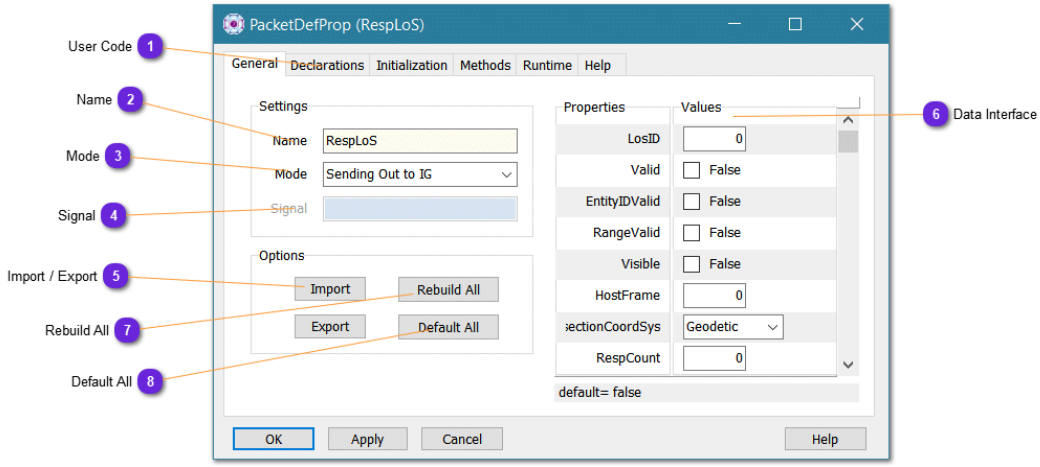
i.e: to send a request for position,
do:

```
      if (entity) {
        CigiPositionReqV3 positionReq;
        positionReq.SetObjectID(entity->getId());

positionReq.SetObjectClass(CigiBasePositionReq::Entity);

positionReq.SetUpdateMode(CigiBasePositionReq::OneShot);

positionReq.SetCoordSys(CigiBasePositionReq::Geodetic);
        // Enqueue the position request
        *mgr->transmitter << positionReq;
      }
```

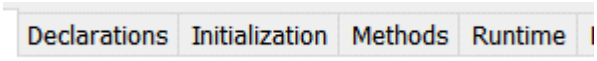
- **Receivers**: Use `getEvent()->data` of type `MsgData*` and cast it according to the type of data you are expected on this packet:
  i.e: for a message type: `CIGI_POSITION_RESP_PACKET_ID_V3`
  do:

```
      response = (CigiPositionRespV3 *) getEvent()->data;
      printf("CigiPositionResp received\n"
            "\tObject id:      %d\n"
            "\tArtPart id:     %d\n", response-
>GetObjectID(), response->GetArtPartID());
```

# Packet Definition




## 1 User Code

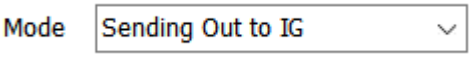



See here.

## 2 Name


Name of the Packet definition. Must be unique.

## 3 Mode




Each Packet can either send or receive. This mode setting will condition the section to be dropped by copy/paste.

## 4 Signal


Available only for **Receiving from IG** mode.
Put here the IG signal which will trigger the Packet.
For i.e: `CIGI_HAT_HOT_RESP_PACKET_ID_V3_2`

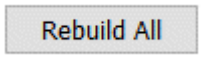
## 5 Import / Export


Import the Packet definition to a single file or import one packet from a file. This is helpful to exchange Packet Definition from one database to one another as the copy/paste might not work. Also good for exchanging definitions by email or across network.
Default location is /Data/Shared. Extension is .pdef

## 6 Data Interface

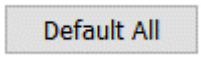

This dynamic interface is the translation of the public user data defined in Declaration panel with `//&&` tags
It's purpose is to provide user an easy way to setup the data of the Packet.

## 7 Rebuild All




Reparse the **Code::Definitions** part to sync the Interface. This is not done automatically as the rebuild process delete the current interface before reparsing and defaulting all the values.

## 8 Default All




Replace all values with the default ones found in DEF[] for each entries.

# Viewers

A Viewer is the shell of the simulation engine that will contain the main loop. The Viewer will create the RTC main node, initialize the simulation, control it and display the output.
When the Viewer targets a specific environment or a graphic engine, it will manage the frequency requirements of the simulation engine as well as the inner requirements of the environment.
A Viewer can be seen as a ready-to-go package, customizable, and provided to the end user as a first integration sample.

When the code is generated from the database, everything is translated into C++ classes and methods, including the user-code. These classes are linked against vsTasker core libraries to produce an executable. For that, a container is requested. This is what we call a Viewer. A Viewer is (and contains) the main of the simulation engine.

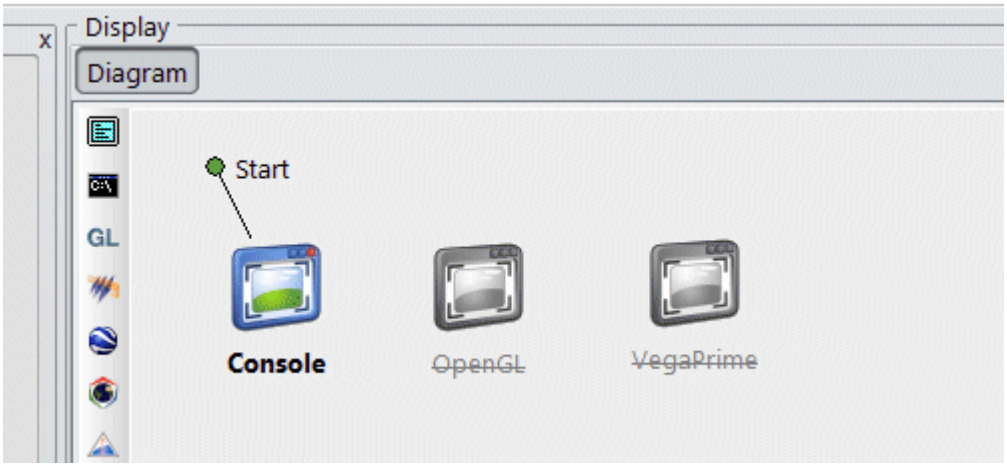
By default, the **Console** is used. The vt_console.cpp is located into runtime/console. It contains the main() function and the call to the entry node (the runtime controller `vt_rtc`).

Several Viewers are provided to the user. They are all modifiable as source code is available in runtime directory. The proposed Viewers must be seen as samples or proof of concept. The user is encouraged to rename and enhance these viewers to target their own needs.

Only one viewer can be active at a time.
The code generator will use the viewer setting to add and link with proper libraries. It is the responsibility of the user to make sure that the selected viewer will support the user-code of the database (Logics, Routines, Reactions, Classes...)
For example, if some logic objects got OpenGL calls (to draw line, explosions, symbols, etc.) during the simulation, it will not be possible to activate and link with a Console viewer; only the OpenGL viewer will work.

But from a Console viewer, it is possible to chose the OpenGL viewer and have a real-time replicated OpenGL window, refreshed at 30hz, similar to the one used for the GUI map (because both items are using the same code, DLL for the GUI, LIB for the simulation engine).

### Viewers




Above are three Viewers added to the Database but only one (Console) is active.
It is mandatory to have at least one Viewer activated.

*A Viewer is not necessarily a graphic engine. It can also embed vsTASKER simulation engine into a Qt application or silently, into another simulation engine. It must be seen as a wrapper.*

## • Environment

Add a User Viewer. See below.

Add a Console viewer

Add an OpenGL viewer

Add a VegaPrime viewer

Add a GoogleEarth viewer

Add an OpenSceneGraph viewer

Add an OsgEarth viewer

Add a Titan viewer

Add an STK viewer

 Add a Delta3D viewer

 Add a GL-Studio viewer

 Add a Triton module viewer

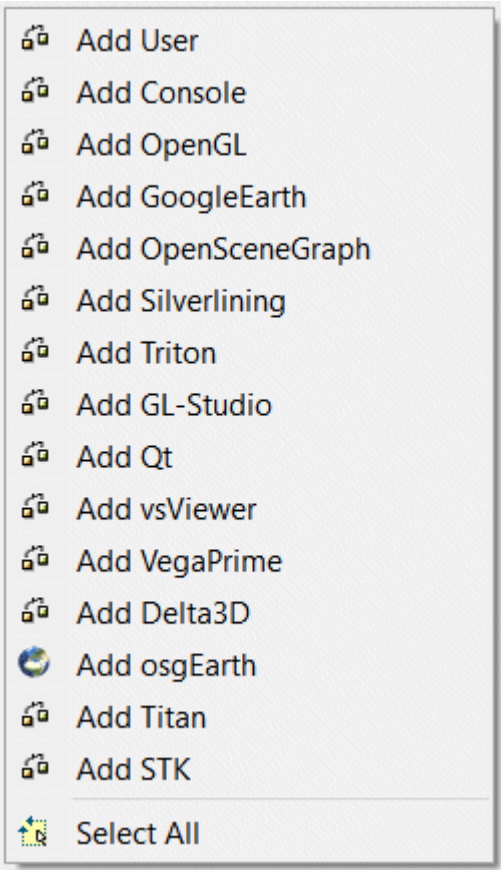
 Add a SilverLining module viewer

 Add a Qt viewer

 Export for vsViewer

## • Popup Menu

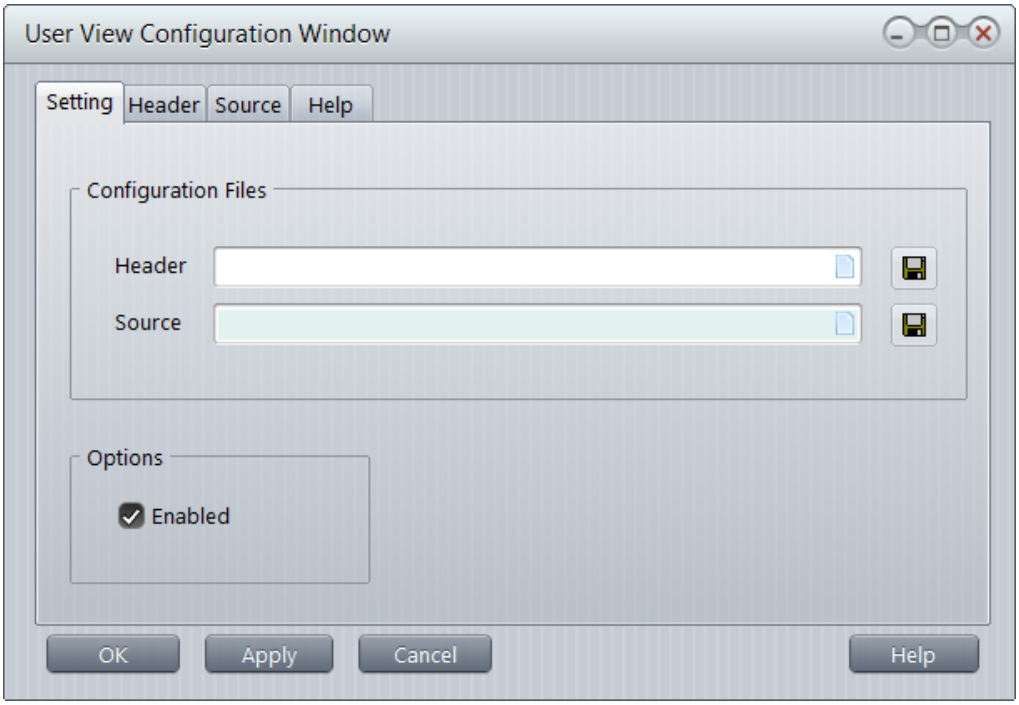
Right click from the diagram bring the popup menu that replaces the vertical toolbar for viewer creation:




## • Creating a User Viewer

Each viewer code is provided as C++ code in the runtime/ directory. So the user can change and modify any of them. It is recommended to duplicate the files before making any change to avoid losing them when installing a new product update.

To integrate vsTASKER with a special viewer or even a dedicated shell defined by a pair C++/H files, just select button on the toolbar and drop it into the diagram area.
Open it and select the **Header** and **Source** code of your viewer into the appropriate fields:



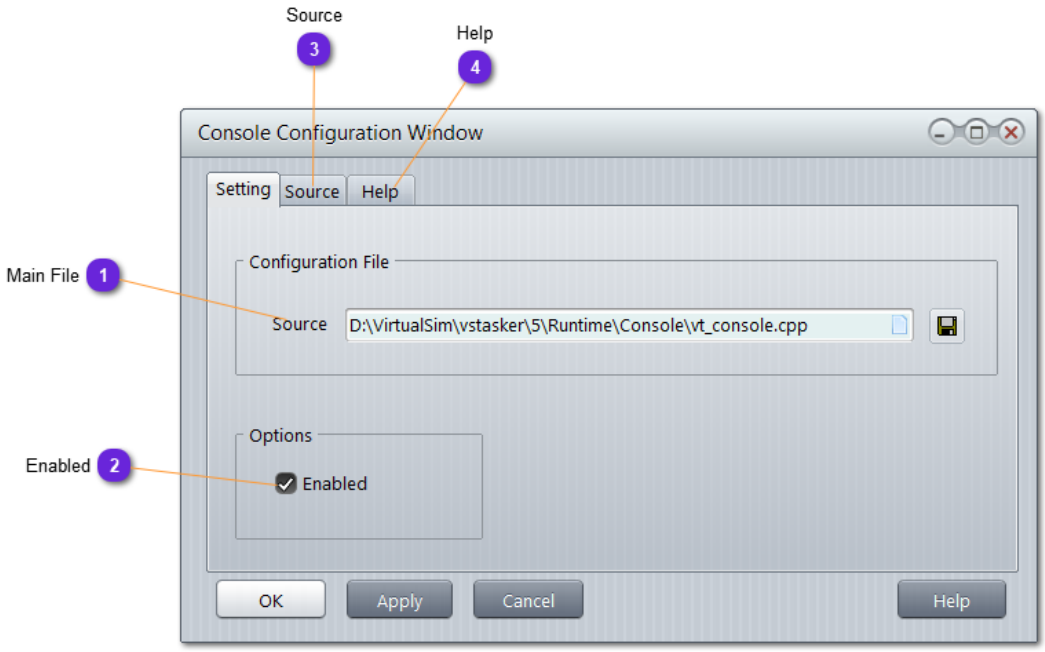

To create your own viewer, it is a good idea to learn from the given examples. Start with the vt_console_man.cpp basic viewer that shows how to initialize the vsTASKER sim engine (`vt_rtc`) and advance time step by step. Normally, you should call your own application from within the runtime loop. If your application already has a runtime loop (OpenGL, VegaPrime, game engine loop, etc.) you must initialize `vt_rtc` before entering the loop then call the `tic()` function from within the loop, including treating all events. You can see vt_opengl.cpp as an example to start.

# Console

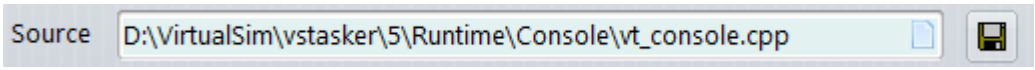
Console viewer is the default viewer. It is the smallest shell for the simulation engine.
Only text print (from the sim) can be displayed on the console.

Two viewers are provided in runtime/console directory:

vt_console.cpp: is the simplest main that define, initialize and call the simulation engine runtime function.
vt_console_man.cpp: integrates a loop sample on how to use the tic() call to advance the RTC time.

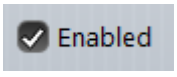



## 1 Main File



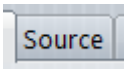

Specifies the C++ wrapper used to build the console application. The default provided one can be found in runtime/console/vt_console.cpp.
Once loaded, the file can be modified in the Source panel.

## 2 Enabled




One viewer must be at least enabled for the simulation to compile. Enabling one viewer disable all the other one.

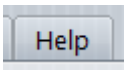
## 3 Source




Source code can be modified into this window or directly from Visual Studio from the solution (or any other text editor).
If code is modified, it will be automatically saved when OK button is depressed.
It is a good idea to save the modified code under another file to keep the original one unchanged. Use  button for that.
Refer to "Understanding Runtime" in the Tutorial document of vsTASKER.


*Do not forget that all provided source code in vsTASKER environment is subject to change from one revision to one another. So, it is a good practice to change the name of any cpp wrapper if any change must be applied on it.*

## 4 Help



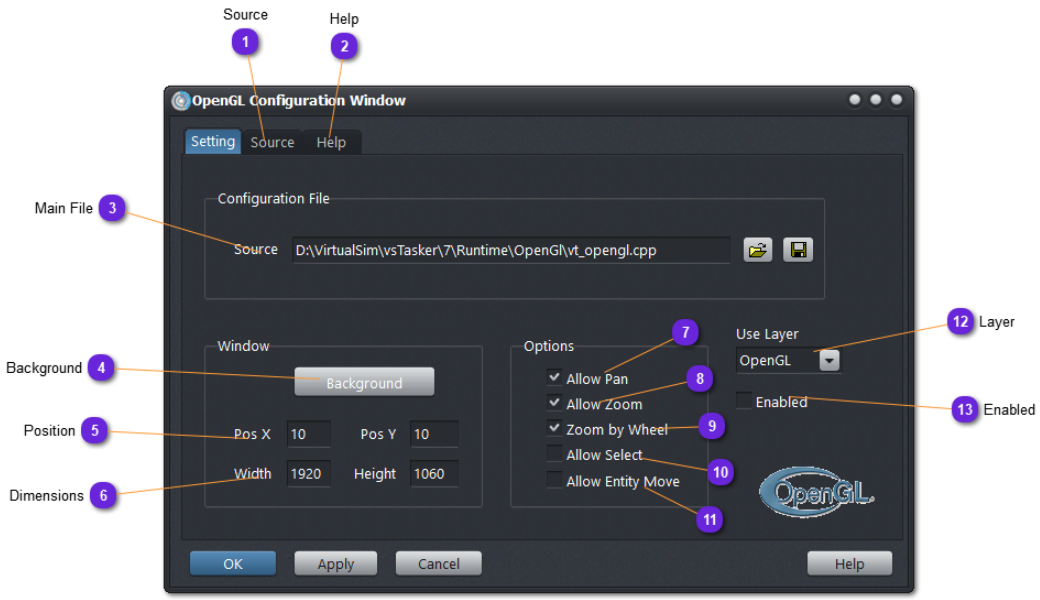

Put here all text explanations or documentation content needed to maintain this Viewer.
The *Help* content is saved but is not inserted into the generated code.
It will be used for the automatic document generation (see Make Documents)

# OpenGL

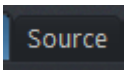
OpenGL viewer is a module available in /Runtime/OpenGL under vt_opengl.cpp name.
It creates and display an windows whose size can be setup either using the Viewer property window.

The viewer will be linked against all plugin libraries selected in Database::Plugins::Sim.
Appropriate code will be automatically generated.

If user want to change representations on the 2D window, code can be modified in plugins/ directory.
If the DLL part is modified, the change will be shown in vsTASKER GUI map display.
If the LIB part is modified, the change will appear in the OpenGL Sim engine associated (viewer) display.



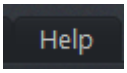

## 1 Source




Source code can be modified into this window or directly from Visual Studio from the solution (or any other text editor).
If code is modified, it will be automatically saved when OK button is depressed.
It is a good idea to save the modified code under another file to keep the original one unchanged. Use button for that.
Refer to "Understanding Runtime" in the Tutorial document of vsTASKER.

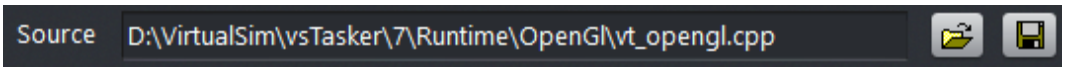
*Do not forget that all provided source code in vsTASKER environment is subject to change from one revision to one another. So, it is a good practice to change the name of any cpp wrapper if any change must be applied on it.*

## 2 Help



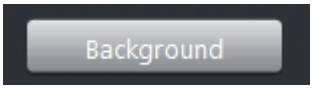

Put here all text explanations or documentation content needed to maintain this Viewer.
The *Help* content is saved but is not inserted into the generated code.
It will be used for the automatic document generation (see Make Documents)

## 3 Main File



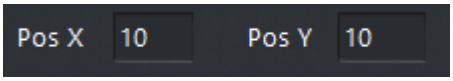

Wrapper main file used to display a 2D OpenGL display as an output of the simulation engine.
/runtime/opengl/vt_opengl.cpp is the default wrapper file.

## 4 Background



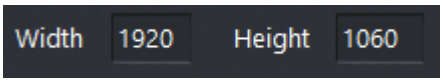

Set the OpenGL window background color. Default is white.

## 5 Position




Specify here the position on the desktop of the upper-left corner of the window. Desktop (0,0) is the upper-left corner of the screen.

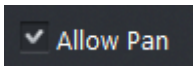
## 6 Dimensions




Specify the dimension of the OpenGL window, in pixels.
The window will be positioned top left of the desktop.


*It is not possible yet to resize the OpenGL once created.*

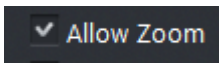
## 7 Allow Pan




If **enabled**, the mouse **can pan** the displayed map/terrain on the window.
If **disabled**, the map center is **fixed** on the window.

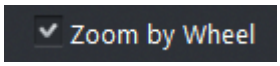

*There is no scroll bar for the displayed map on the OpenGL window, so, panning can only be available through the mouse action.*
*Maintain down the **right mouse button** or combining **CTRL key** and **left mouse button**.*

## 8 Allow Zoom



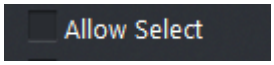

If checked, zooming on the map is allowed.

## 9 Zoom by Wheel



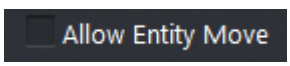

If **enabled**, the **mouse wheel** is used to control the zoom.
If **disabled**, the zoom is done using the **mouse motion** by combining the **CTRL** key and **mouse right key**.

## 10 Allow Select



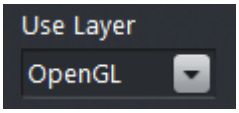

If **enabled**, the mouse left button can be used to **select entities or whatever** can be selected on the map, like during design time.
If **disabled**, **nothing** can be selected.

## 11 Allow Entity Move



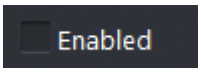

If checked, mouse click can select any displayed entity. Repositioning the selected entity is possible (maintain the left mouse button down while dragging).

## 12 Layer




Select which layer to use to create the OpenGL window.
**OpenGL**: default, pure OpenGL main window
**Glut**: main using glut32 (3.7.6 version) enhanced library to support mouse wheel. Useful when menus are requested.

## 13 Enabled




One viewer must be at least enabled for the simulation to compile. Enabling one viewer disable all the other one.
When unchecked, all specific viewer code (and third party software API) are not generated and do not impact the makefile.

# GoogleEarth

This Viewer is requested to use Google-Earth (GE) application to be used as a 3D viewer.

## • Setting up

In order to use GE as a 3D viewer for vsTASKER simulation engine, an old version of GE (v7 and before) must be used because the COM+ capability has been deprecated by Google.
Also, an Apache server must run in order to command GE.

**Installing Apache**

Go into /runtime/geath/3rd party directory.
Run appserv-win32-2.6.0.exe file and install only Apache.
Give whatever server name and email you want.
Finish installation but do not run yet Apache server.

Once installed, go to C:/AppServ/Apache2.2/conf and edit file httpd.conf
Search for `DocumentRoot`.
Replace with the following:

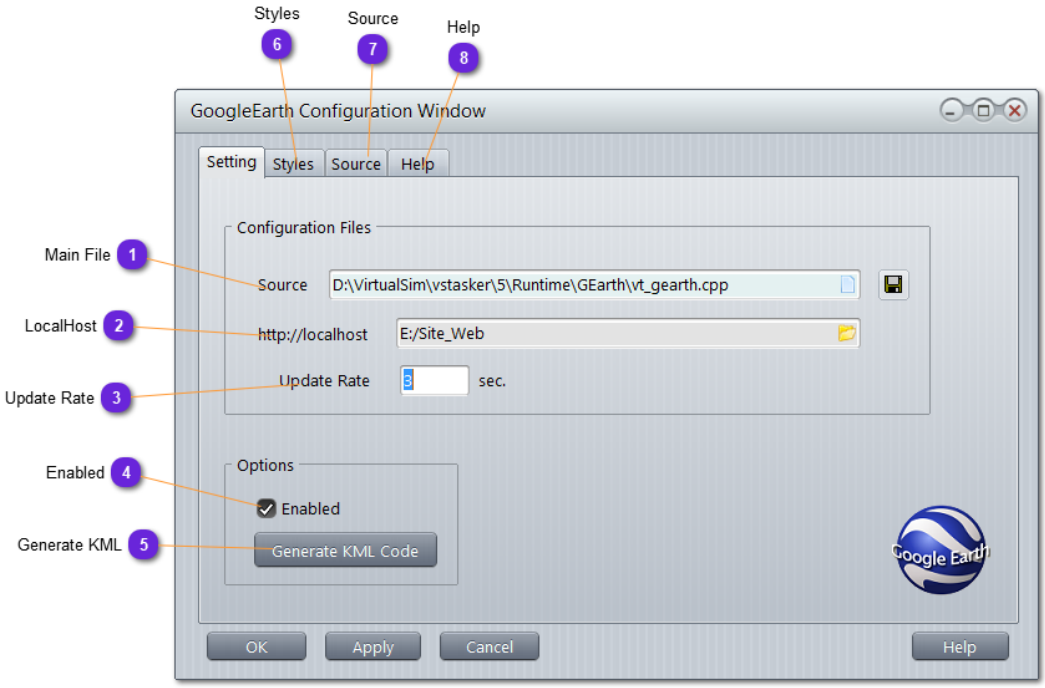
```
DocumentRoot "D:/VirtualSim/LocalHost"
```

Create the directory D:/VirtualSim/LocalHost
Use this directory in the below window setting, for the field http://localhost
Start Apache in C:/AppServer/Apache2.2/bin/httpd.exe

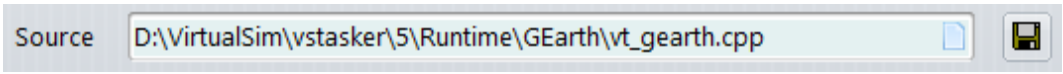
**Installing GoogleEarth**

Go into /runtime/geath/3rd party directory.
Run GoogleEarthWin_7.exe file and install.
Close GE.

Now, you can try to run the samples provided in /Data/Db/GEarth directory.

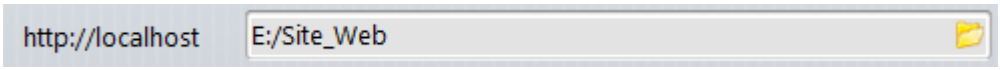



## 1 Main File



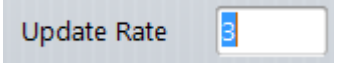

Wrapper main file used to communicate with GE application (using COM API) as an output of the simulation engine.
/runtime/gearth/vt_gearth.cpp is the default wrapper file.

## 2 LocalHost



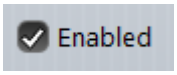

Directory where the sim engine will drop files GE must read as a stream of command.
Correspond to the `DocumentRoot` value of the Apache server (configuration file, see above).
Suggested value: D:/VirtualSim/LocalHost

## 3 Update Rate



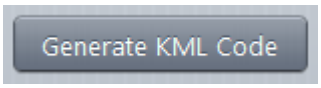

Frequency at which the data file sent to GE will be produced. 1hz seems the minimum for GE.

## 4 Enabled



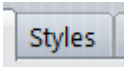

One viewer must be at least enabled for the simulation to compile. Enabling one viewer disable all the other one.
When unchecked, all specific viewer code (and third party software API) are not generated and do not impact the makefile.

## 5 Generate KML



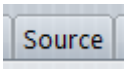

Force some files to be generated in localhost for testing purposes. These files are normally sent to GE in a streaming link.

## 6 Styles




Lists the graphical objects that will be displayed on GE as Icons for Entities and Points (feature).

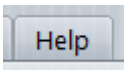
## 7 Source




Source code can be modified into this window or directly from Visual Studio from the solution (or any other text editor).
If code is modified, it will be automatically saved when OK button is depressed.
It is a good idea to save the modified code under another file to keep the original one unchanged. Use  button for that.
Refer to "Understanding Runtime" in the Tutorial document of vsTASKER.


*Do not forget that all provided source code in vsTASKER environment is subject to change from one revision to one another. So, it is a good practice to change the name of any cpp wrapper if any change must be applied on it.*

## 8 Help




Put here all text explanations or documentation content needed to maintain this Viewer.
The *Help* content is saved but is not inserted into the generated code.
It will be used for the automatic document generation (see Make Documents)

# OpenSceneGraph

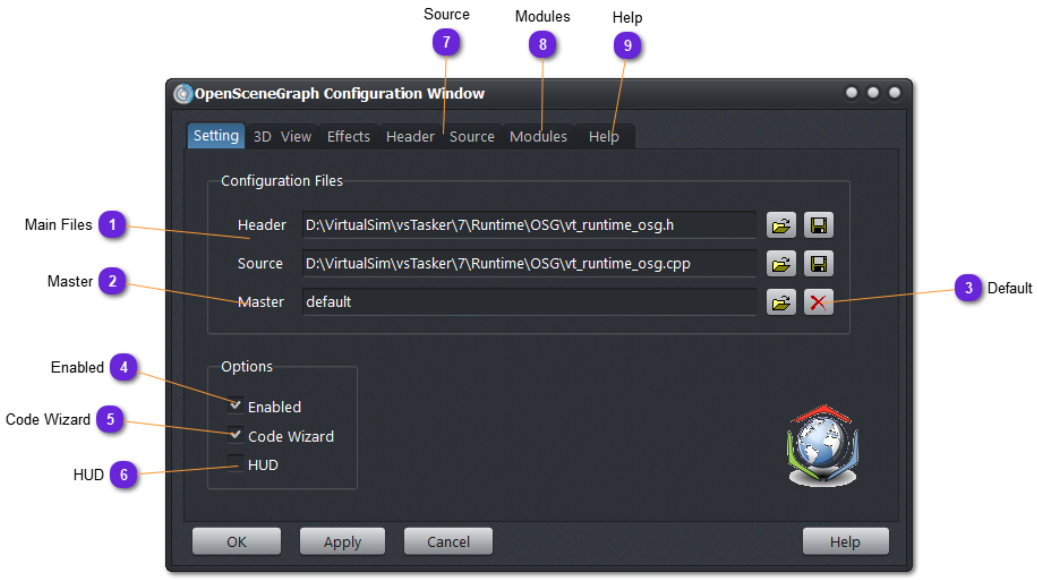
OpenSceneGraph (OSG) viewer facilitates the integration of vsTASKER simulation engine with OSG 3D viewer.
The Viewer will combine OSG library and output visual with runtime engine, outputting a unique (and possibly standalone) application.
You can get OSG onto www.openscengraph.com website.

*This viewer can also be used with vsVIEWER and Qt, as both can support an OSG thread.*

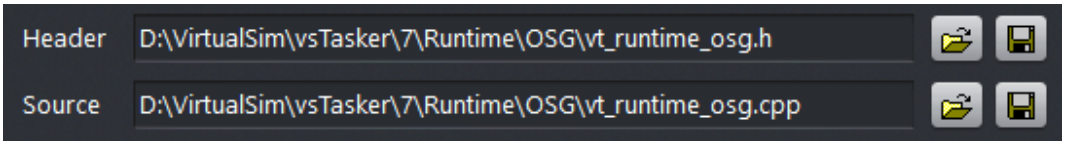
Out of the box, vsTASKER is distributed with the version **3.4.0** of OSG.
OSG DLL are located in /bin/vc[100/140]/osgPlugins.
OSG libraries are located in /lib/vc[100/140]/osg.

*The OSG setting is done in the user modifiable /src/OSG/osg_setup.cpp*
*This file is compiled with the project when needed.*

Make sure that the `%path%` contains a link to OSG dll directory to make the executable run correctly (sim engine will not run otherwise).

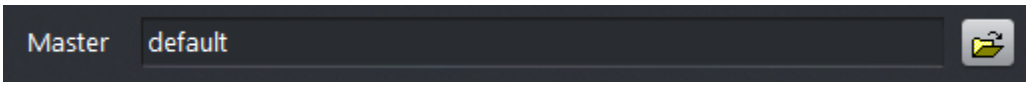


## 1 Main Files




Wrapper main files (header and source) embedding the simulation engine and managing the OSG main window and loop.
/runtime/osg/vt_runtime_osg.h/cpp are the default wrapper files.

## 2 Master

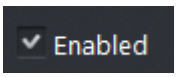



By default, the simulation engine will get the same terrain database (file) as defined in **Terrain::3D Editor::OpenSceneGraph** window. But often, the terrain that should be displayed on vsTASKER GUI 3D view does not need all the details of the rendered scene. In such case, the GUI and SIM database are different in size and details but should keep the same scale, origin and projection definitions.

## 3 Default

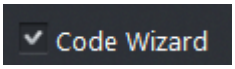
Clear the overwriting terrain database file and revert to default one (defined in **Terrain::3D Editor::OpenSceneGraph**)

## 4 Enabled




One viewer must be at least enabled for the simulation to compile. Enabling one viewer disable all the other one.
When unchecked, all specific viewer code (and third party software API) are not generated and do not impact the makefile.

## 5 Code Wizard

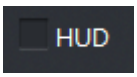



When this option is checked, the source code of vsTASKER will be processed by the wizard in order to include all the necessary OSG code to make entities displayed automatically on the OSG terrain view. The specific code will be added between special tags that shall not be removed. Once inserted, it will no more be changed and user can amend it.
If unchecked, the wizard will remove all wizard added code, including whatever user has inserted between the tags!

*Using the wizard has now been deprecated since version 7.1. It can still work but we strongly suggest to use the Model Components, as explained in the Tutorial.*
*Note that both methods cannot be mixed! To convert a database to use Model Components, first, uncheck the Code Wizard, generate the code then proceed as indicated in the Tutorial.*

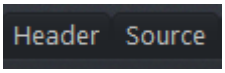

*Wizard only runs at code generation. To add or remove it, check or uncheck then generate the code.*

## 6 HUD




When checked, display some textual data on 2D projection.
The object used is /src/osg/osg_hud.cpp
The function called at each frame is `drawImplementation()`.
Either the 2D OpenGL code is entered here, either it is put into the HUD panel of the HMI property window (in such case, do not disable the HMI to allow code generation).

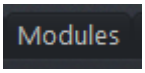
## 7 Source




Header and Source code can be modified into this window or directly from Visual Studio from the solution (or any other text editor).
If code is modified, it will be automatically saved when OK button is depressed.
It is a good idea to save the modified code under another file to keep the original one unchanged. Use  button for that.
Refer to "Understanding Runtime" in the Tutorial document of vsTASKER.

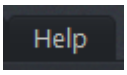

*Do not forget that all provided source code in vsTASKER environment is subject to change from one revision to one another. So, it is a good practice to change the name of any cpp wrapper if any change must be applied on it.*

## 8 Modules



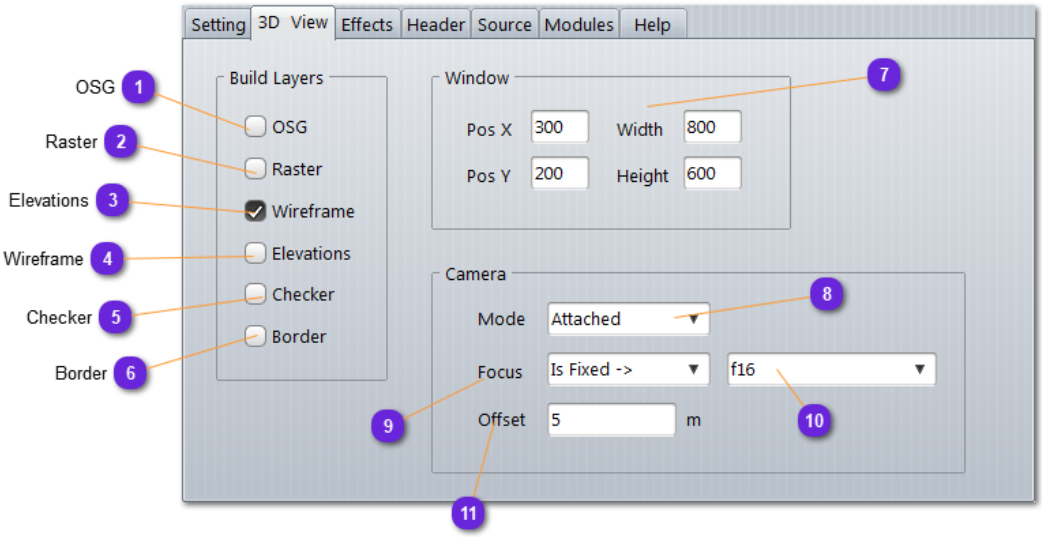

Add here all user defined modules needed for the OSG viewer to compile and link.
All the built-in modules provided in *runtime/OSG/src* are already included into the makefile by vsTASKER, so no need to include them here.
Add only new OSG modules that expand the viewer capabilities.

## 9 Help




Put here all text explanations or documentation content needed to maintain this Viewer.
The *Help* content is saved but is not inserted into the generated code.
It will be used for the automatic document generation (see Make Documents)

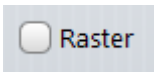
# 3D View




## 1 OSG

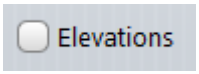

When checked, the terrain database specified in the 3D terrain setting (see here) will be loaded automatically into the scene (and offset if specified in the Setting window)

## 2 Raster



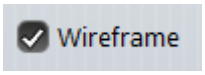

When checked, raster maps (tiled or not) will be created into the OSG scene, in place of the terrain.
Raster tiles will be flat.

## 3 Elevations



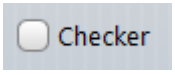

When checked, the elevation data (if loaded) are reconstructed into OSG scene.

## 4 Wireframe



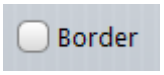

When checked, the terrain wireframe (if loaded) will be reconstructed into the OSG scene.

## 5 Checker



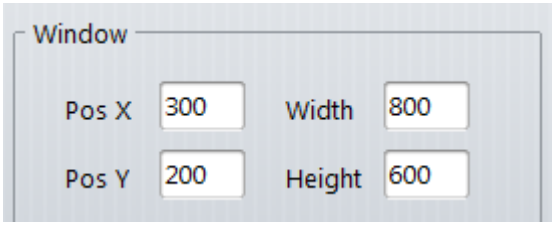

When checked, a checker surface with black and white squares, will be automatically constructed into the OSG scene the replace the missing terrain.

## 6 Border



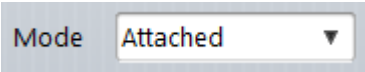

When checked, only the border of the terrain, as a line, will be added into the OSG scene to materialize the terrain surface.

## 7 Window




Position of the OSG window (top left corner) on the desktop (Pos X, Pos Y). Width and height in pixels of the OSG window.

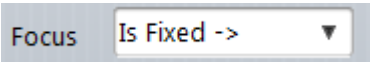
## 8 Mode




Select here the camera manipulator to use in the OSG scene:

- **Free**: the camera will be looking at the center of the OSG terrain scene. Mouse can move the camera using any of the three mouse button (while dragging).
- **Cockpit**: the camera will be put at the position of the focused entity (see below), clamped on a virtual seat, moving with the entity (yaw, pitch and roll). The position of the camera will be offset by the Eye Offset values in PtfBody (if defined).
- **Behind**: the camera will be put behind the position of the focused entity, and will remain fixed, looking at the entity.
- **Top**: the camera will be put top of the position of the focused entity, and will remain fixed, looking at the entity.
- **Attached**: the camera will be free to rotate around the focused entity. Mouse can move the camera using any of the three mouse button (while dragging).
- **Below**: the camera will be put below the position of the focused entity, and will remain fixed, looking at the entity
- **Gimbal**: the camera will be mounted to a gimbal attached below the entity. The gimbal will be used controllable (and so the camera)

## 9 Focus




Select the entity focus mode to use by the camera:
On Select: the camera will focus on any mouse selected entity in vsTASKER GUI (or using any other method, as long as an entity is selected)
Is Fixed: the entity will need to be specified before start. See below.

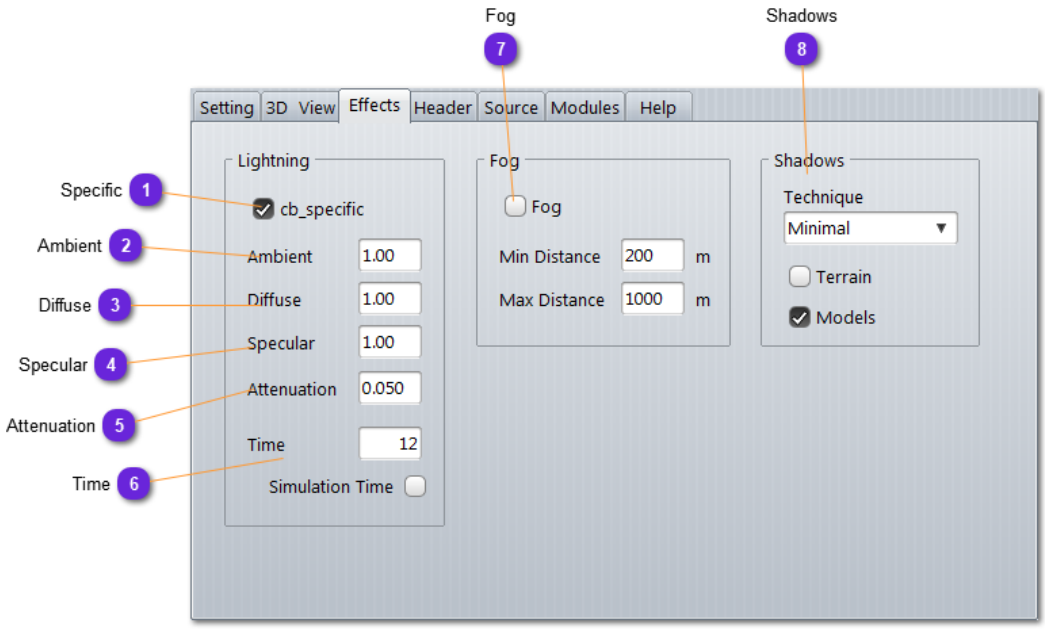
## 10 Focused Entity


Select from the drop down list the entity that will be automatically focused by the camera.

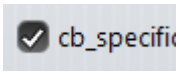
## 11 Offset


This value works for **Behind**, **Top** and **Attached** modes (8). Specify the distance at which the camera will be put at start.

# Effects

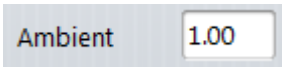



## 1 Specific




Select this option to change the values of the light.
Time can also be specified here.

## 2 Ambient

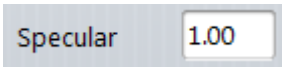



Ambient illumination is light that's been scattered so much by the environment that its direction is impossible to determine - it seems to come from all directions. Backlighting in a room has a large ambient component, since most of the light that reaches your eye has first bounced off many surfaces. A spotlight outdoors has a tiny ambient component; most of the light travels in the same direction, and since you're outdoors, very little of the light reaches your eye after bouncing off other objects. When ambient light strikes a surface, it's scattered equally in all directions.
Value ranges from 0 to 1.

## 3 Diffuse


The diffuse component is the light that comes from one direction, so it's brighter if it comes squarely down on a surface than if it barely glances off the surface. Once it hits a surface, however, it's scattered equally in all directions, so it appears equally bright, no matter where the eye is located. Any light coming from a particular position or direction probably has a diffuse component.
Value ranges from 0 to 1.

## 4 Specular

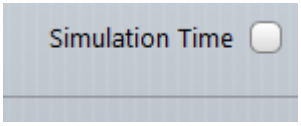



Specular light is the white highlight reflection seen on smooth, shinny objects. Specular light is dependent on the direction of the light, the surface normal and the viewer location.
Value ranges from 0 to 1.

## 5 Attenuation

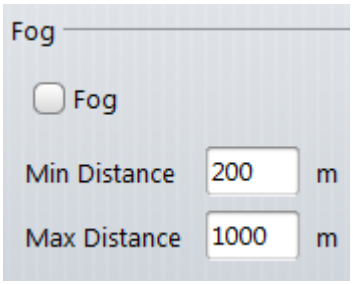

Set the linear attenuation of the light.
Value ranges from 0 to 1.

## 6 Time



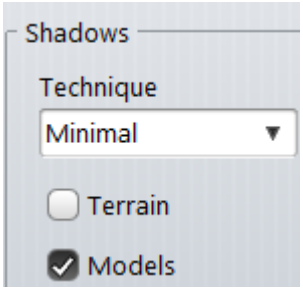

Time of the simulation, between 0h and 23h. This will simulate the obfuscation of the light according to the time of day.
Sunset and Sunrise are programmed to be at 6. Maximum daylight is at 12.

## 7 Fog




If checked, fog will be simulated.
For now, the only parameters that can be set are the minimum distance at which the fog starts and the maximum distance that can be seen into the fog. i.e: above, the fog will start at 500 meters from the camera and nothing beyond 1000 meters will be visible.

## 8 Shadows




Select the shadow **Technique** that must be used for the shadowing of the scene:

- **None**: no shadow
- **Minimal**: simple and fast shadow
- **Standard**: normal shadowing with neat borders
- **Soft**: normal shadowing with soft borders (blurred). More realistic but CPU intensive.

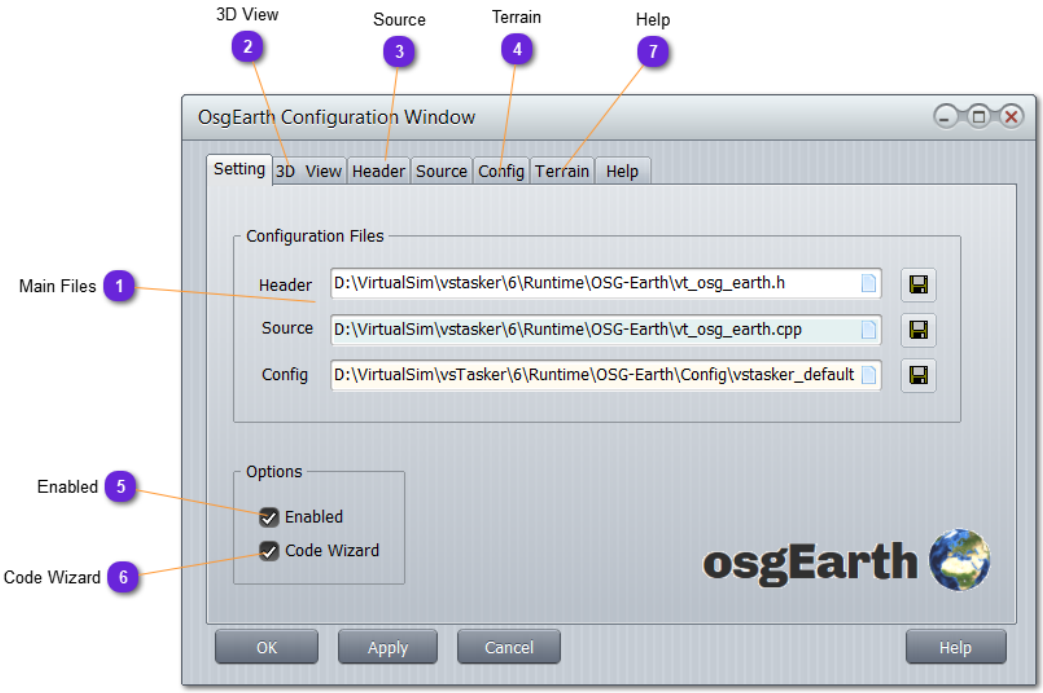
If Terrain is checked, the scene will cast shadows (any objects belonging to the terrain scene node). Use this option if you need hills and trees to cast shadows on the terrain.
If Models is checked, entity models will cast shadows on the terrain.

# OsgEarth

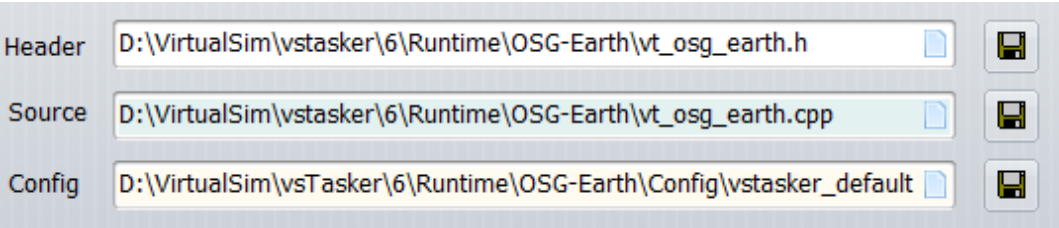
OsgEarth (OSG based) viewer facilitates the integration of vsTASKER simulation engine with a on demand terrain 3D viewer.
The Viewer will combine OsgEarth library and output visual with runtime engine, outputting a unique (and possibly standalone) application.
You can get OsgEarth onto osgearth.org website.

Out of the box, vsTASKER is distributed with the version **2.8** of OsgEarth.
OsgEarth dll are located in /bin/vc[90/100/110/120]/osgPlugins
OsgEarth libs are located in /lib/vc[90/100/110/120]/osg/osgEarth

Make sure that the `%path%` contains a link to OsgEarth dll directory to make the executable run correctly (will not appear otherwise).



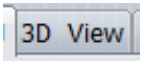
## 1 Main Files




Wrapper main file (header & source) embedding the simulation engine and managing the OsgEarth main window and loop.
/runtime/osg-earth/vt_osg_earth.h/cpp are the default wrapper files.

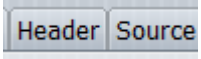
The Config XML .earth file is mandatory for the earth enfine to run and embed all the drivers needed to query data on the Internet (or from the cache).
vsTASKER uses some predefined ones located in /runtime/osg-earth/config

## 2 3D View




See here

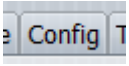
## 3 Source




Header file, Source code and Config XML can be modified into this window or directly from Visual Studio from the solution (or any other text editor).
If code is modified, it will be automatically saved when OK button is depressed.
It is a good idea to save the modified code under another file to keep the original one unchanged. Use  button for that.
Refer to "Understanding Runtime" in the Tutorial document of vsTASKER.

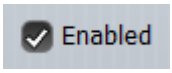

*Do not forget that all provided source code in vsTASKER environment is subject to change from one revision to one another. So, it is a good practice to change the name of any cpp wrapper if any change must be applied on it.*

## 4 Terrain



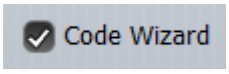

See here

## 5 Enabled




One viewer must be at least enabled for the simulation to compile. Enabling one viewer disable all the other one.
When unchecked, all specific viewer code (and third party software API) are not generated and do not impact the makefile.

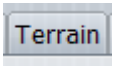
## 6 Code Wizard




When this option is checked, the source code of vsTASKER will be processed by the wizard in order to include all the necessary OsgEarth code to make entities displayed automatically on the OsgEarth terrain view. The code will be added between special tags that shall not be removed. Once inserted, it will no more be changed and user can amend it.
If unchecked, the wizard will remove all wizard added code.

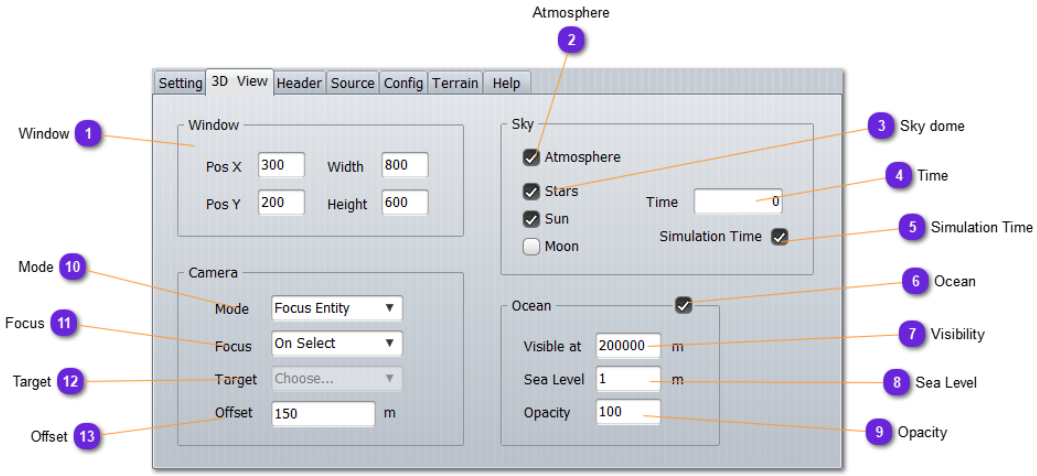

*Wizard only runs at code generation. To add or remove it, check or uncheck then generate the code.*

## 7 Help

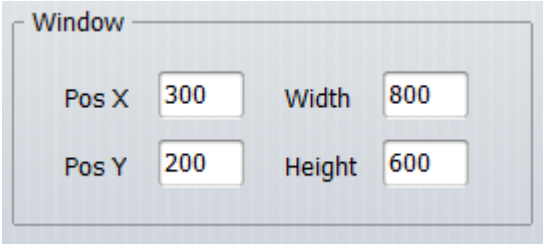



Put here all text explanations or documentation content needed to maintain this Viewer.
The *Help* content is saved but is not inserted into the generated code.
It will be used for the automatic document generation (see Make Documents)

# 3D View




## 1 Window

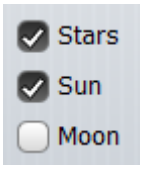



Position of the OsgEarth window (top left corner) on the desktop (Pos X, Pos Y).
Width and height in pixels of the resulting window.

## 2 Atmosphere


Check this option to enable the Atmospheric driver.
It must be included into the .earth file with the <sky /> tag.

## 3 Sky dome




Populate the sky dome with stars, the sun and/or the moon.
Works only if Atmosphere is enabled.

## 4 Time


Enter here the fixed UTC time (`time_t`) at which the earth shall be setup for day/light illumination and sun/moon position.

## 5 Simulation Time


If checked, the simulation time (Julian date) will be taken for the earth day/light illumination and sun/moon position on the sky.

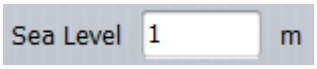
## 6 Ocean


Check this button is you want ocean to be visible on the earth.
The driver must be included into the .earth file with the <ocean /> tag.

## 7 Visibility

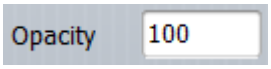

Enter here the maximum distance at which the ocean (texture/surface) will be visible from the camera.

## 8 Sea Level



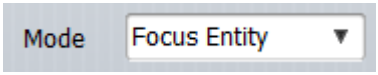

Specify here the tide effect above zero terrain altitude.

## 9 Opacity




Value for the transparency of the ocean texture. 0 for maximum transparency. Any value above increase the opacity.

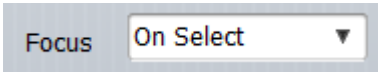
## 10 Mode




Specify here how the camera in osgEarth should react at a mouse selection on the vsTASKER GUI.

- **Free**: the camera will never react to an entity selection
- **Focus** Entity: whenever an entity will be selected (mouse of from code), the camera will move to it (and tether it).

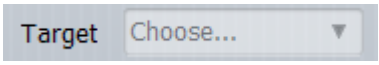
## 11 Focus




In case of automatic focusing, choice is given to either let the camera go from one selected entity to the other, or to select a predefined one (must exist):

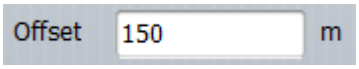
- **On Select**: will be on Free mode until one entity is selected. Tether it.
- **Is Fixed** ->: will only tether one predefined entity and remain locked on it.

## 12 Target



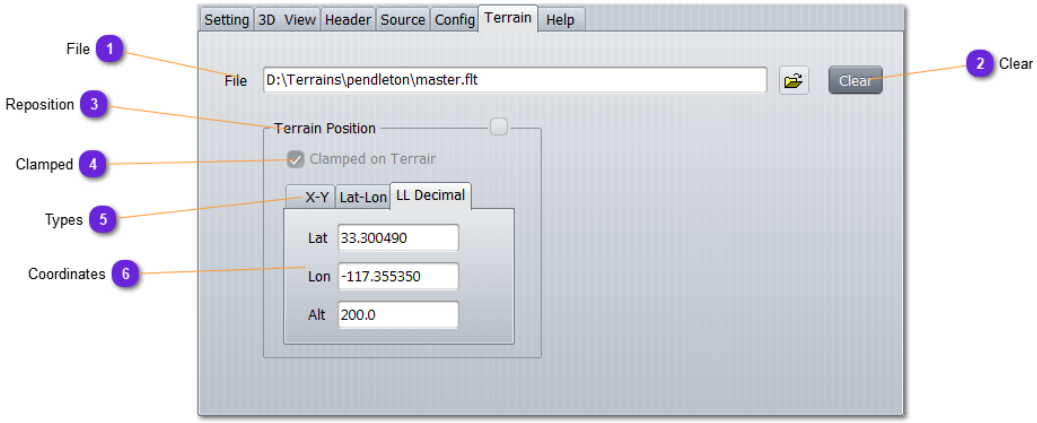

In case the **Focus** option is Fixed, select here from the drop down list the entity (must exist) the camera will tether automatically at startup.

## 13 Offset




Distance of the camera to the entity (node), in meters.
It will always be possible to zoom in or out.

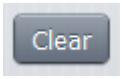
# Terrain Import




## 1 File


Use the open icon to select the file that will be imported into osgEarth. Supported terrains are OSG, IVE and OpenFlight. They must be geodetic although, not mandatory.
Some terrains cannot be loaded without explanation. They will not be visible. If so, try go get a geodetic projection for it.

## 2 Clear

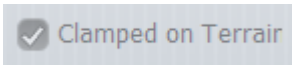



Suppress the file and reset the coordinates.

## 3 Reposition


*Not available for the moment*

## 4 Clamped

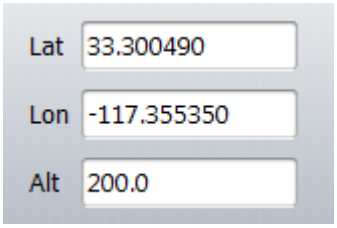



If selected, the terrain altitude will not be considered and the imported terrain altitude will be clamped on the below osgEarth terrain surface.
If not selected, the given altitude will be used for the lower-left corner of the terrain (that might then float above earth surface).

## 5 Types


Select here the format of the terrain coordinates, in case of repositioning.

## 6 Coordinates

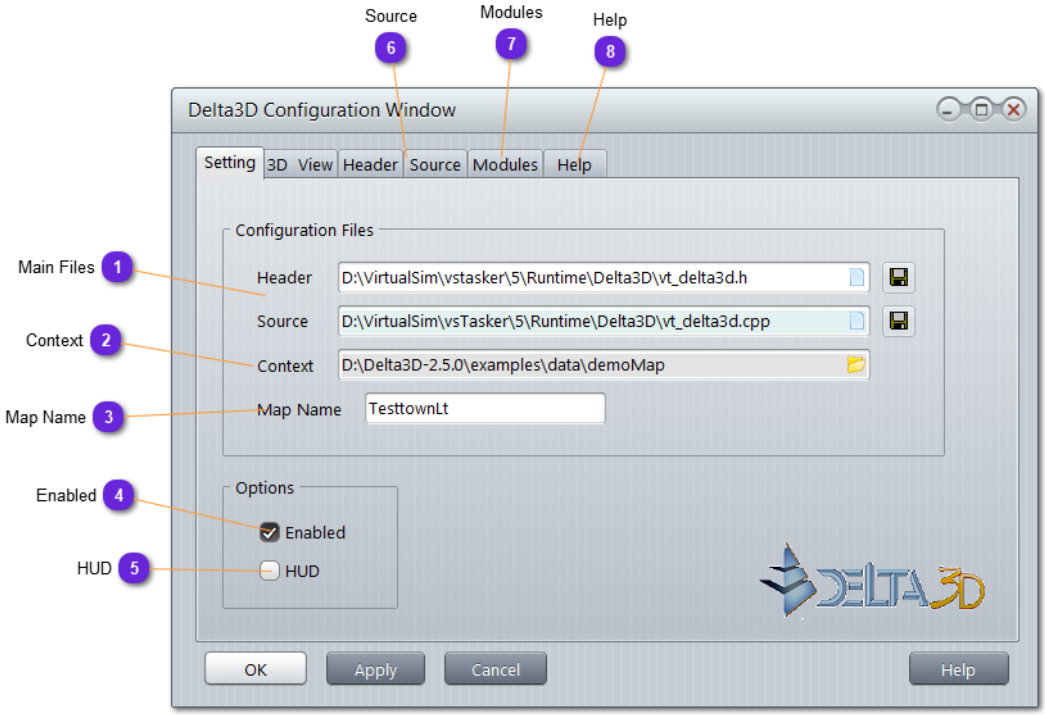



Enter here the coordinates of the lower-left corner of the imported terrain. Some terrain lower-left corner is (0,0) and then, offsetting is easy. Some are already georeferenced and do not need reposition.

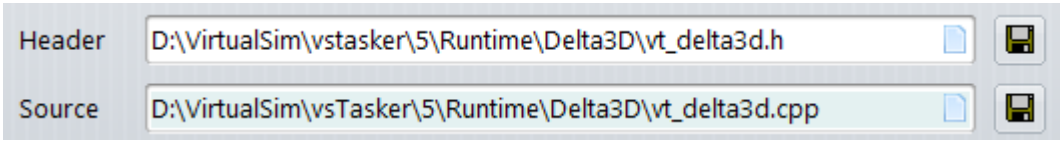
# Delta3D

This viewer used the Delta3D game engine, available for download at http://delta3d.org
Delta3d is a game and simulation engine appropriate for a wide variety of simulation and entertainment applications. It uses best-of-breed open source technologies to create a fully integrated game engine and with content creation tools.

*vsTASKER supports the version 2.7 of Delta3D that is compatible with OSG 3.1.5. If a newer version must be used, the viewer code will have to be updated. Compatibility problems might arise with the different OSG version between the version 3.1.5 of vsTASKER 5 and the one of Delta3D 2.8*

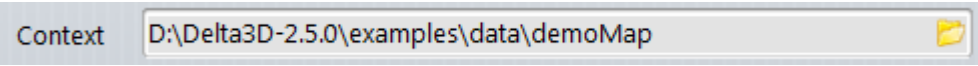



## 1 Main Files



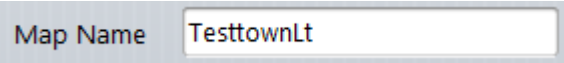

Wrapper main files (header and source) embedding the simulation engine and managing the Delta3D main window and loop.
/runtime/delta3d/vt_delta3d.h/cpp are the default wrapper files.

## 2 Context



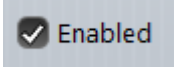

Directory where the map files and all textures, models particles (...) will be found.

## 3 Map Name



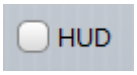

Terrain name (xml format) found under maps directory in the Context.

## 4 Enabled



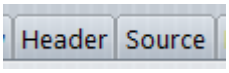

One viewer must be at least enabled for the simulation to compile. Enabling one viewer disable all the other one.
When unchecked, all specific viewer code (and third party software API) are not generated and do not impact the makefile.

## 5 HUD




When checked, display some textual data on 2D projection.
The object used is /src/osg/osg_hud.cpp
The function called at each frame is `drawImplementation()`.
Either the 2D OpenGL code is entered here, either it is put into the HUD panel of the HMI property window (in such case, do not disable the HMI to allow code generation).

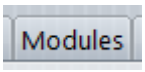
## 6 Source




Header and Source code can be modified into this window or directly from Visual Studio from the solution (or any other text editor).
If code is modified, it will be automatically saved when OK button is depressed.
It is a good idea to save the modified code under another file to keep the original one unchanged. Use  button for that.
Refer to "Understanding Runtime" in the Tutorial document of vsTASKER.

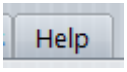

*Do not forget that all provided source code in vsTASKER environment is subject to change from one revision to one another. So, it is a good practice to change the name of any cpp wrapper if any change must be applied on it.*

## 7 Modules



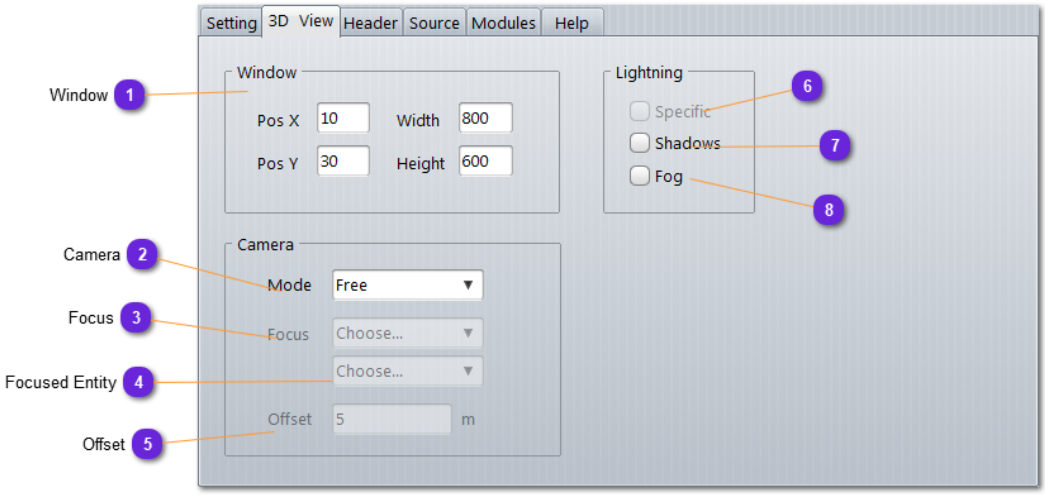

Add here all user defined modules needed for the Delta3D viewer to compile and link.
All the built-in modules provided in *runtime/OSG/src* are already included into the makefile by vsTASKER, so no need to include them here.
Add only new Delta3D modules that expand the viewer capabilities.

## 8 Help

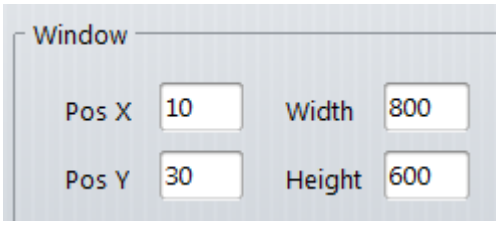



Put here all text explanations or documentation content needed to maintain this Viewer.
The *Help* content is saved but is not inserted into the generated code.
It will be used for the automatic document generation (see Make Documents)

# 3D View

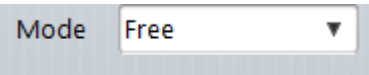



## 1 Window




Position of the OSG window (top left corner) on the desktop (Pos X, Pos Y).
Width and height in pixels of the OSG window.

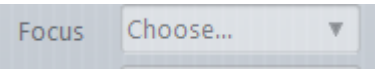
## 2 Camera




Select here the camera manipulator to use in the scene:

- **Free**: the camera will be looking at the center of the terrain scene. Mouse can move the camera using any of the three mouse button (while dragging).
- **Cockpit**: the camera will be put at the position of the focused entity (see below), clamped on a virtual seat, moving with the entity (yaw, pitch and roll). The position of the camera will be offset by the Eye Offset values in PtfBody (if defined).
- **Behind**: the camera will be put behind the position of the focused entity, and will remain fixed, looking at the entity.
- **Top**: the camera will be put top of the position of the focused entity, and will remain fixed, looking at the entity.
- **Attached**: the camera will be free to rotate around the focused entity. Mouse can move the camera using any of the three mouse button (while dragging).

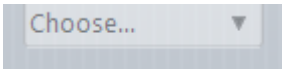
## 3 Focus




Select the entity focus mode to use by the camera:

- **On Select**: the camera will focus on any mouse selected entity in vsTASKER GUI (or using any other method, as long as an entity is selected)
- **Is Fixed**: the entity will need to be specified before start. See below.

## 4 Focused Entity

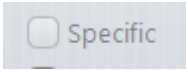



Select from the drop down list the entity that will be automatically focused by the camera.

## 5 Offset


This value works for **Behind**, **Top** and **Attached** modes (8). Specify the distance at which the camera will be put at start.

## 6 Specific




When checked, the Ambient light is turned on (sun light).
When unchecked, only defined lights will illuminate the scene.

## 7 Shadows


When checked, the shadow (normal technique) is applied on the scene (terrain and objects).

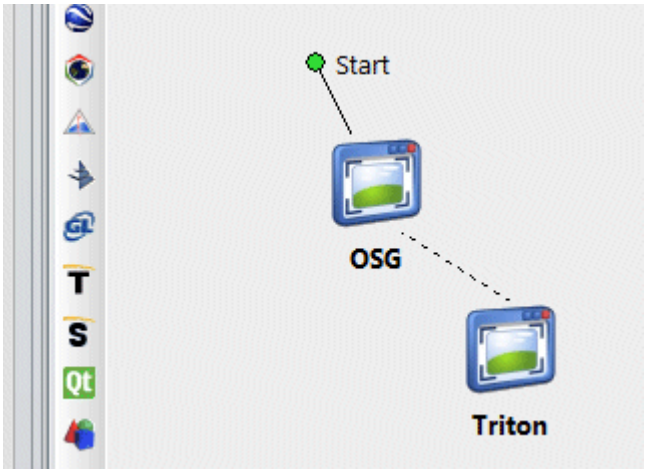
## 8 Fog


When checked, fog is enabled on the scene (default setting).

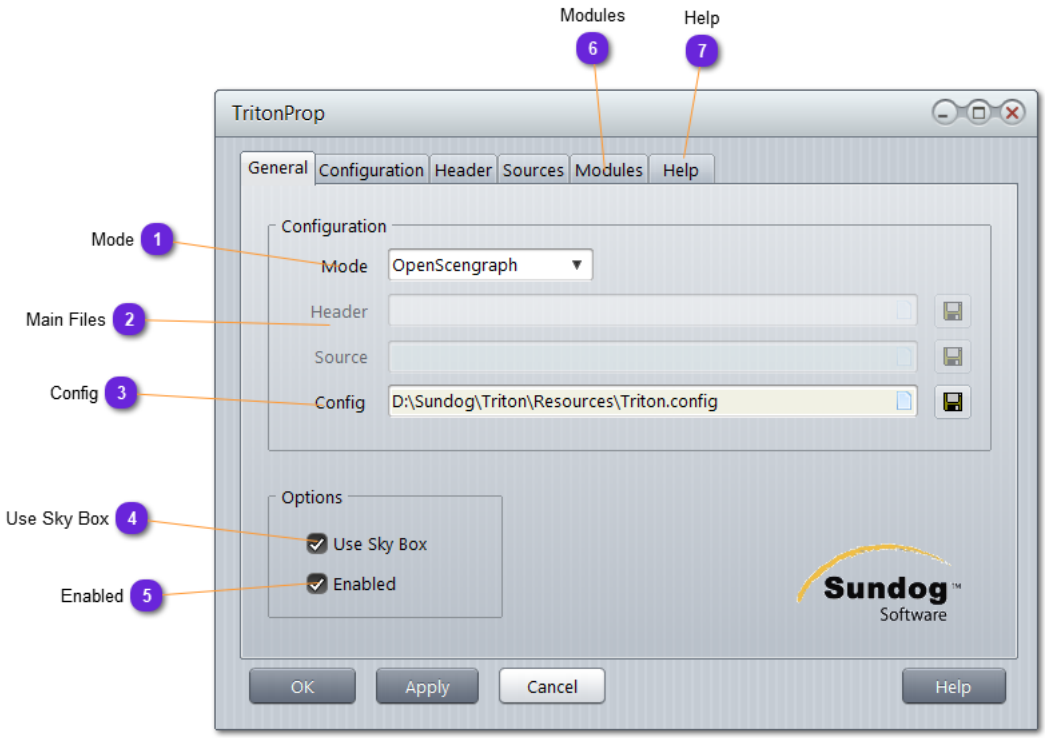
# Triton

Triton provides realistic visuals of waves for any given wind conditions or a specified Beaufort scale, and takes advantage of general-purpose GPU computing on systems using OpenGL. Triton also features the ability to seamlessly integrate with geocentric coordinate systems in addition to flat coordinate systems. The surface of Triton's sea may be configured to conform to a WGS84 ellipsoid, or a spherical model of the Earth.
You can get Triton at www.sundog-software.com

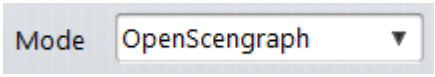
vsTASKER allows Triton to extent OSG scene by adding water surface to the terrain (but not necessarily), as Triton needs an IG to connect to.




*Triton software and license must be requested from a proper vendor (Sundog-Software), as vsTASKER does not provide them nor behave as a reseller or does support line for the product.*




## 1 Mode




Select the IG Triton connects to (currently, only one choice: OSG)

## 2 Main Files

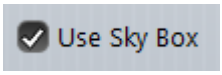

*Not available for OSG mode.*

## 3 Config


Config file used by Triton for setup.
Content will be displayed in Configuration panel.

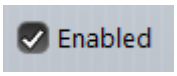

*Duplicate/backup original file to have something to revert back if you change the parameter values and save them using*  .

## 4 Use Sky Box



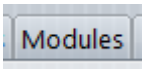

If checked, Triton displays a sky box (fix raster image with sun and according lightning) on the scene.
Uncheck it if you do not need it or if SilverLining is used.

## 5 Enabled



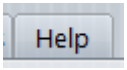

When checked, Triton `#define` will be added into the code to enable all the Triton related code in vt_runtime_osg.cpp
vsTASKER produces all needed classes and code in generated files.
When unchecked, all specific viewer code (and third party software API) are not generated and do not impact the makefile.

## 6 Modules



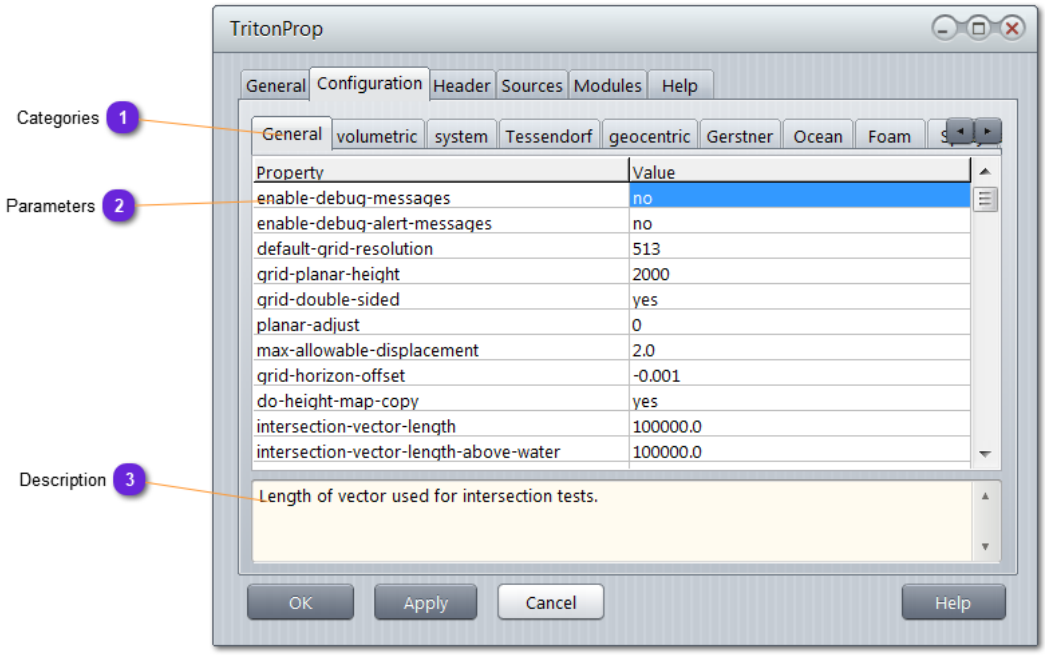

Put here all your modules that expand the Triton functionality.

## 7 Help

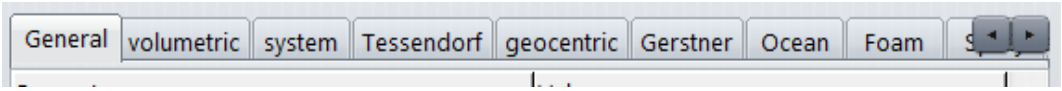



Put here all text explanations or documentation content needed to maintain this Viewer.
The *Help* content is saved but is not inserted into the generated code.
It will be used for the automatic document generation (see Make Documents)

# Configuration




## 1 Categories

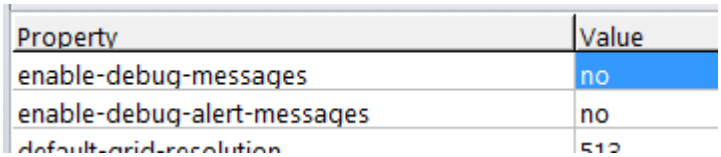



vsTASKER extract from the triton.config file all the categories listed between ##### according to the following pattern:

```
##########################################
Settings for system compatibility
##########################################
```

If also reduces the title to a pertinent world (above, system is elected)

## 2 Parameters

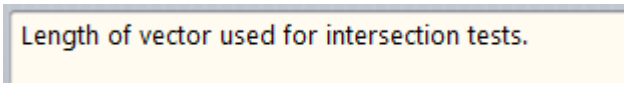



For each category, vsTASKER lists all parameters and their value, according to the following pattern:

```
# Explicitly disable the use of CUDA 4.0 and CUFFT to conduct
FFT's on NVidia systems.
disable-cuda = no
```

Property column displays the parameter name (left of =) while Value displays the right = part.

## 3 Description

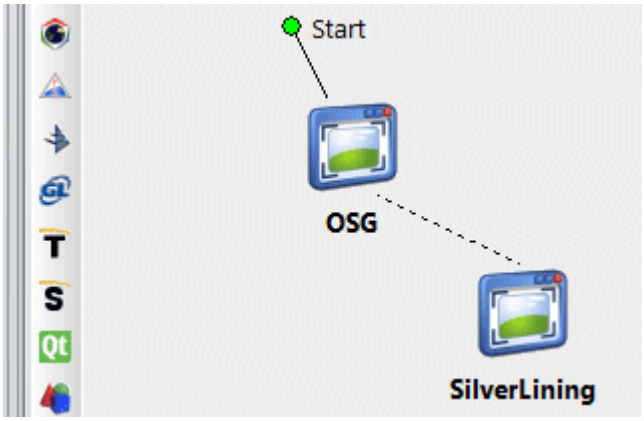



When mouse gets over a line of the Parameter table (2), displays the description found into the file, above the listed parameter.

# SilverLining

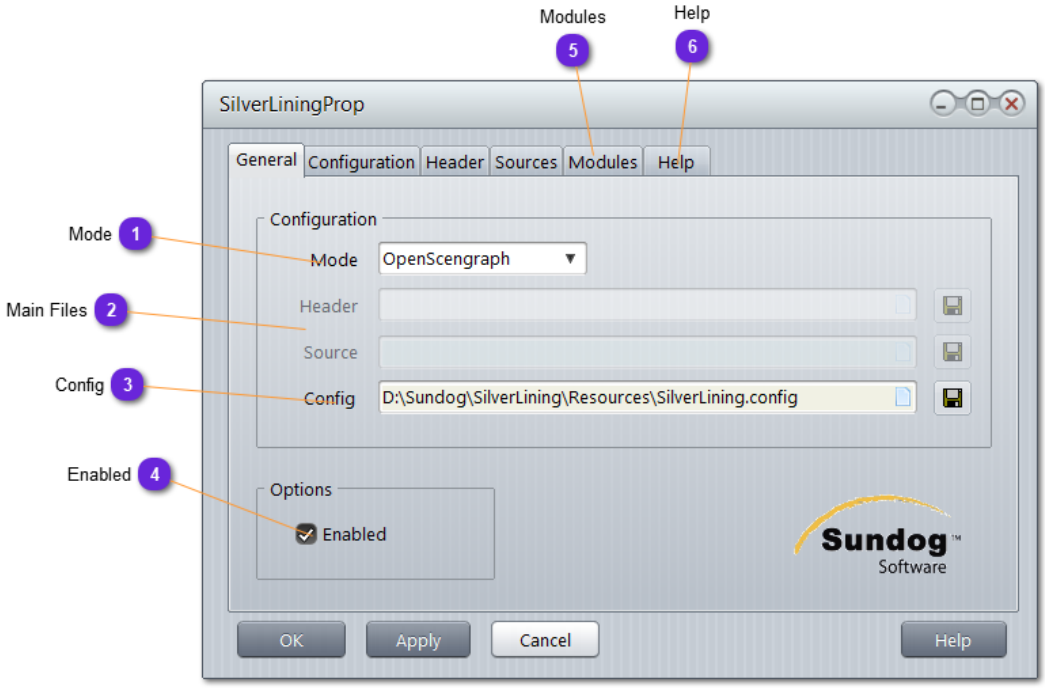
SilverLining is a class library for real-time visual simulation of the atmosphere, clouds, sky, precipitation, sun, moon, stars, and planets. SilverLining allows you to create highly realistic, high-performance outdoor scenes with 3D clouds for any location at any time of day.
You can get SilverLining at [www.sundog-software.com](http://www.sundog-software.com)

vsTASKER allows SilverLining to extent OSG scene by adding clouds to the sky. SilverLining needs an IG to connect to.

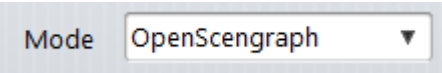




*SilverLining software and license must be requested from a proper vendor (Sundog-Software), as vsTASKER does not provide them nor behave as a reseller or does support line for the product.*

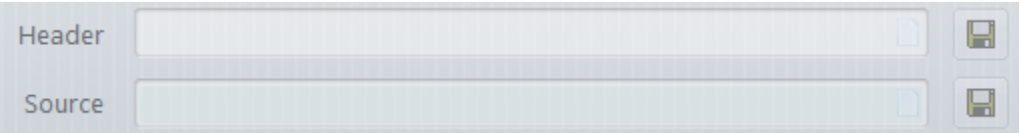


## 1 Mode



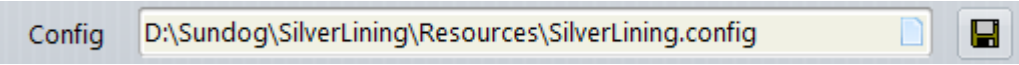

Select the IG SilverLining connects to (currently, only one choice: OSG)

## 2 Main Files




*Not available for OSG mode.*

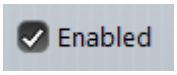
## 3 Config




Config file used by SilverLining for setup.
Content will be displayed in Configuration panel.

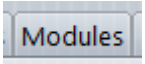
*Duplicate/backup original file to have something to revert back if you change the parameter values and save them using .*

## 4 Enabled



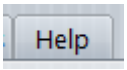

When checked, SilverLining `#define` will be added into the code to enable all the SilverLining related code in vt_runtime_osg.cpp
vsTASKER produces all needed classes and code in generated files.
When unchecked, all specific viewer code (and third party software API) are not generated and do not impact the makefile.

## 5 Modules



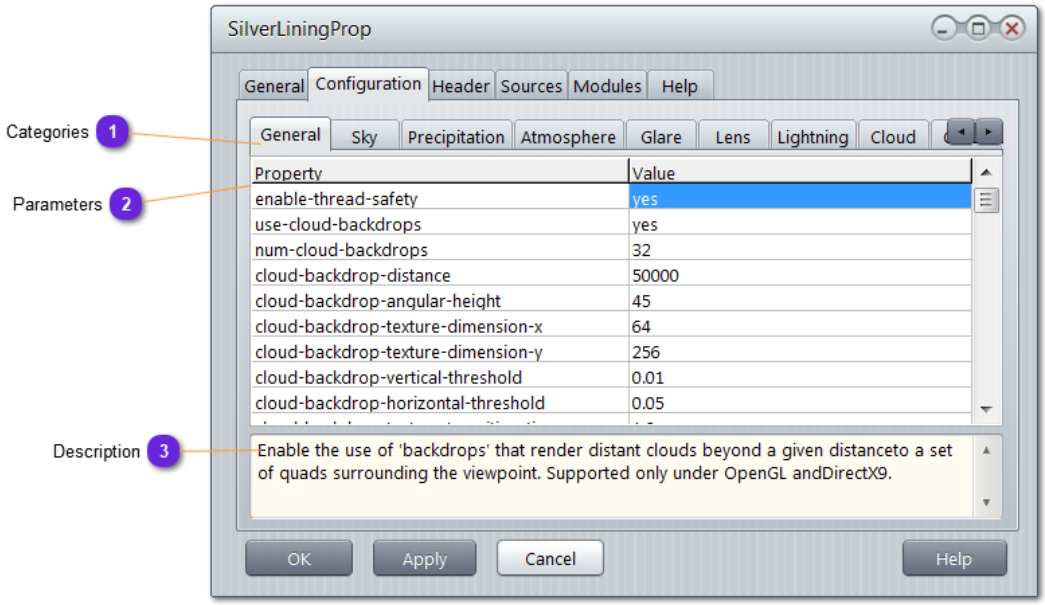

Put here all your modules that expand the Triton functionality.

# 6 Help

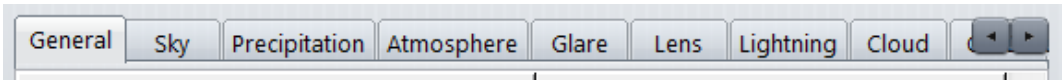



Put here all text explanations or documentation content needed to maintain this Viewer.
The *Help* content is saved but is not inserted into the generated code.
It will be used for the automatic document generation (see Make Documents)

# Configuration




## 1 Categories

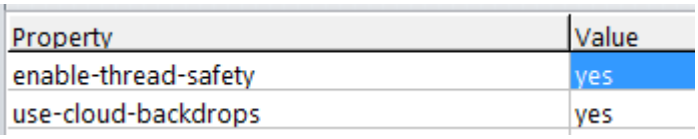



vsTASKER extract from the silverlining.config file all the categories listed between ##### according to the following pattern:

```
##########################################
# Sky Settings
##########################################
```

If also reduces the title to a pertinent world (above, system is elected)

## 2 Parameters

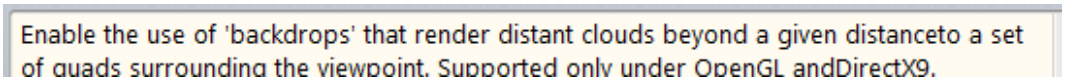



For each category, vsTASKER lists all parameters and their value, according to the following pattern:

```
# Scale the luminance of our twilight table values
twilight-scale = 1.0
```

Property column displays the parameter name (left of =) while Value displays the right = part.

## 3 Description




When mouse gets over a line of the Parameter table (2), displays the description found into the file, above the listed parameter.

# VegaPrime

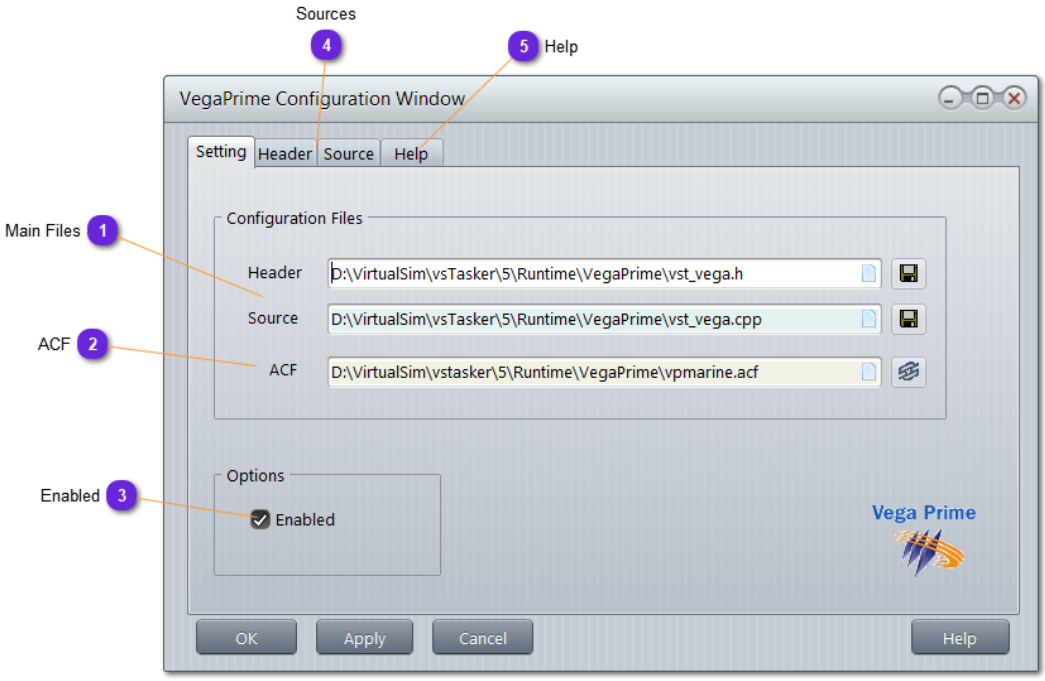
VegaPrime viewer facilitates the integration of vsTASKER simulation engine with Presagis VegaPrime IG.
The Viewer will combine VegaPrime library with vsTASKER runtime libraries, outputting a unique (and possibly standalone) application.

VegaPrime viewer is a module available in runtime/vegaprime under vst_vega.cpp name.
vsTASKER will generate a VegaPrime application by embedding the simulation RTC into the VegaPrime main runtime loop.

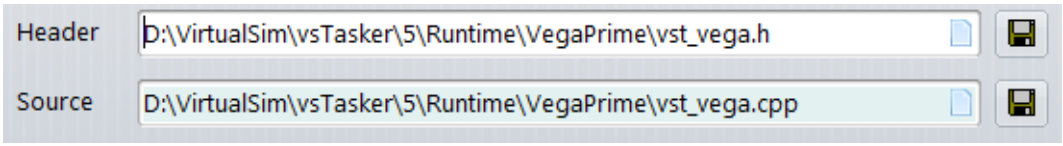

*VegaPrime software and license must be requested from a proper vendor (Presagis), as vsTASKER does not provide them nor behave as a reseller or does support line for the product.*
*Supported version goes up to v3.*

Refer to the **Tutorial document** to learn how to develop your first VegaPrime simulation.

# Properties

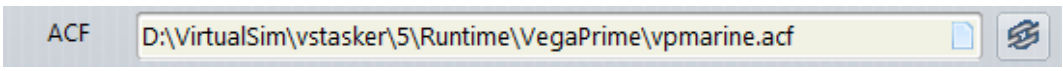



## 1 Main Files



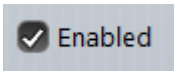

Wrapper main files (header and source) embedding the simulation engine and managing the VegaPrime main window and loop.
/runtime/vegaprime/vst_vega.h/cpp are the default wrapper files.

## 2 ACF



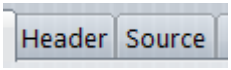

Specify here the LynxPrime.exe definition file that will be given to VegaPrime engine at load.
The acf file contains the scene, all settings, parameters, objects, effects, sounds and everything needed by the IG to run.
See VegaPrime documentation and how to use LynxPrime utility.

## 3 Enabled




One viewer must be at least enabled for the simulation to compile. Enabling one viewer disable all the other one.

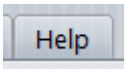
## 4 Sources




Header and Source code can be modified into this window or directly from Visual Studio from the solution (or any other text editor).
If code is modified, it will be automatically saved when OK button is depressed.
It is a good idea to save the modified code under another file to keep the original one unchanged. Use  button for that.
Refer to "Understanding Runtime" in the Tutorial document of vsTASKER.


*Do not forget that all provided source code in vsTASKER environment is subject to change from one revision to one another. So, it is a good practice to change the name of any cpp wrapper if any change must be applied on it.*

## 5 Help




Put here all text explanations or documentation content needed to maintain this Viewer.
The *Help* content is saved but is not inserted into the generated code.
It will be used for the automatic document generation (see Make Documents)

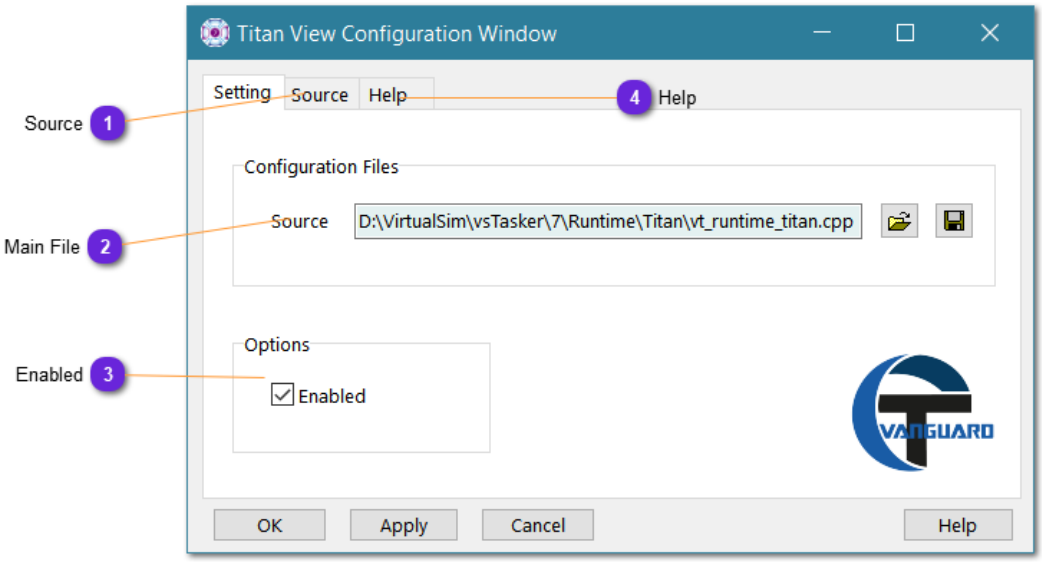
# Titan

Titan viewer is a module available in runtime/titan under vt_runtime_titan.cpp name. vsTASKER will generate a DLL by embedding the simulation RTC into a DLL to be loaded by Titan at startup.


*Titan Vanguard software and license must be requested from a proper vendor (Calytrix), as vsTASKER does not provide them nor behave as a reseller or does support line for the product.*

Refer to the **Tutorial document** to learn how to develop your first Titan simulation.

# Properties

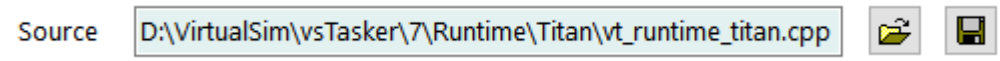



## 1 Source


Source code can be modified into this window or directly from Visual Studio from the solution (or any other text editor).
If code is modified, it will be automatically saved when OK button is depressed.
It is a good idea to save the modified code under another file to keep the original one unchanged. Use  button for that.
Refer to "Understanding Runtime" in the Tutorial document of vsTASKER.

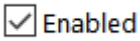

*Do not forget that all provided source code in vsTASKER environment is subject to change from one revision to one another. So, it is a good practice to change the name of any cpp wrapper if any change must be applied on it.*

## 2 Main File



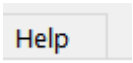

Specifies the C++ wrapper used to build the Titan DLL. The default provided one can be found in runtime/titan/vt_runtime_titan.cpp.
Once loaded, the file can be modified in the Source panel.

## 3 Enabled




One viewer must be at least enabled for the simulation to compile. Enabling one viewer disable all the other one.

## 4 Help




Put here all text explanations or documentation content needed to maintain this Viewer.
The *Help* content is saved but is not inserted into the generated code.
It will be used for the automatic document generation (see Make Documents)

# STK

Most of AGI's customers are using STK for space, aircraft, UAV and intelligence mission design and analysis.
Integration with vsTASKER facilitates designing what-if scenarios as this capability is difficult to define in STK scenarios where everything is precalculated.

## • How it works

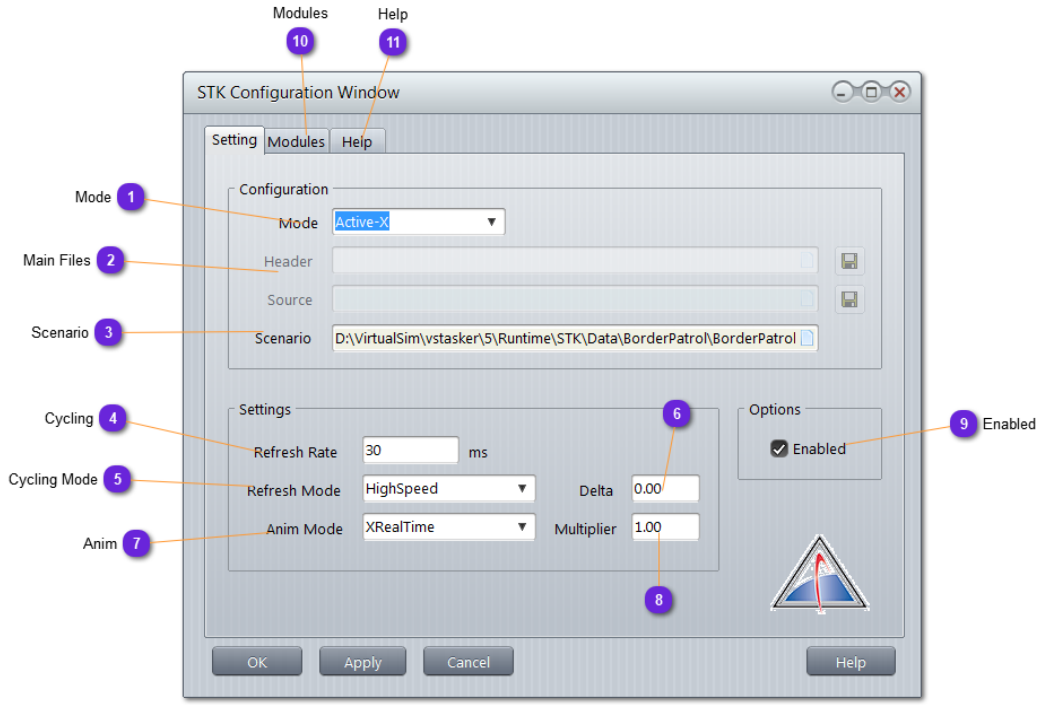
vsTASKER can connect to STK using the embedded STK viewer.
Three modes are available:

- COM
- Connect
- Active/X

See the Viewer for more details.

Refer to the **Tutorial document** to learn how to develop your first STK simulation.

# Settings



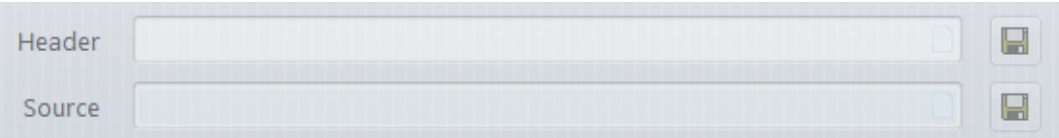

## 1 Mode


Select the kind of integration requested with STK API:

- **COM**: use the published COM interface of STK
- **Active-X**: use STK viewer using the Active-X component
- **Connect**: communicate with STK using the LAN/Socket method.

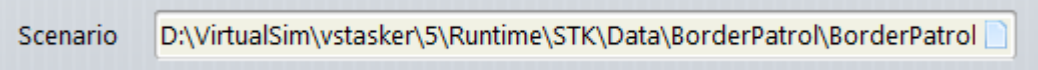

*For COM and Active-X, vsTASKER supports only STK up to version 9.*

## 2 Main Files



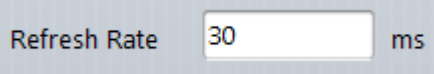

In Connect mode, use `runtime/stk/vt_connect.h|cpp`
In COM mode, use only `runtime/stk/vt_com.cpp`
In Active-X, vsTASKER will automatically generate the code call to the STK viewer.

## 3 Scenario



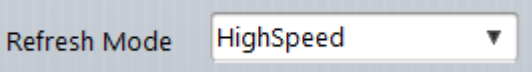

Specify here which scenario STK must load at startup (only for Active-X and Connect mode).

## 4 Cycling




Synchronization frequency for runtime clock (only for Active-X mode).

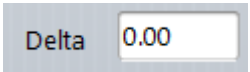
## 5 Cycling Mode




Select between:

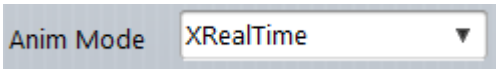
- **HighSpeed**: update data as fast as possible
- **Delta**: update data when Delta milliseconds (below) are elapsed since the last update.

## 6 Delta



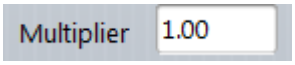

When Delta mode is selected for the refresh, specify the time to wait in milliseconds, between two updates.

## 7 Anim



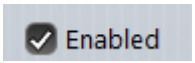

STK internal runtime mode setup. See STK documentation.

## 8 Multiplier



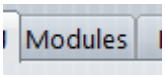

In case XRealTime is selected for Anim Mode, specify the multiplier time factor to use.

## 9 Enabled



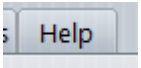

One viewer must be at least enabled for the simulation to compile. Enabling one viewer disable all the other one.
When unchecked, all specific viewer code (and third party software API) are not generated and do not impact the makefile.

## 10 Modules




List all modules that will be included into the makefile for compilation and link. The list content depends on the selected Mode (COM, ActiveX or Connect).
Use ⊕ to add more files and ✕ to remove selected ones.
Default reverts to what the list should be, discarding all user entries (or removal).

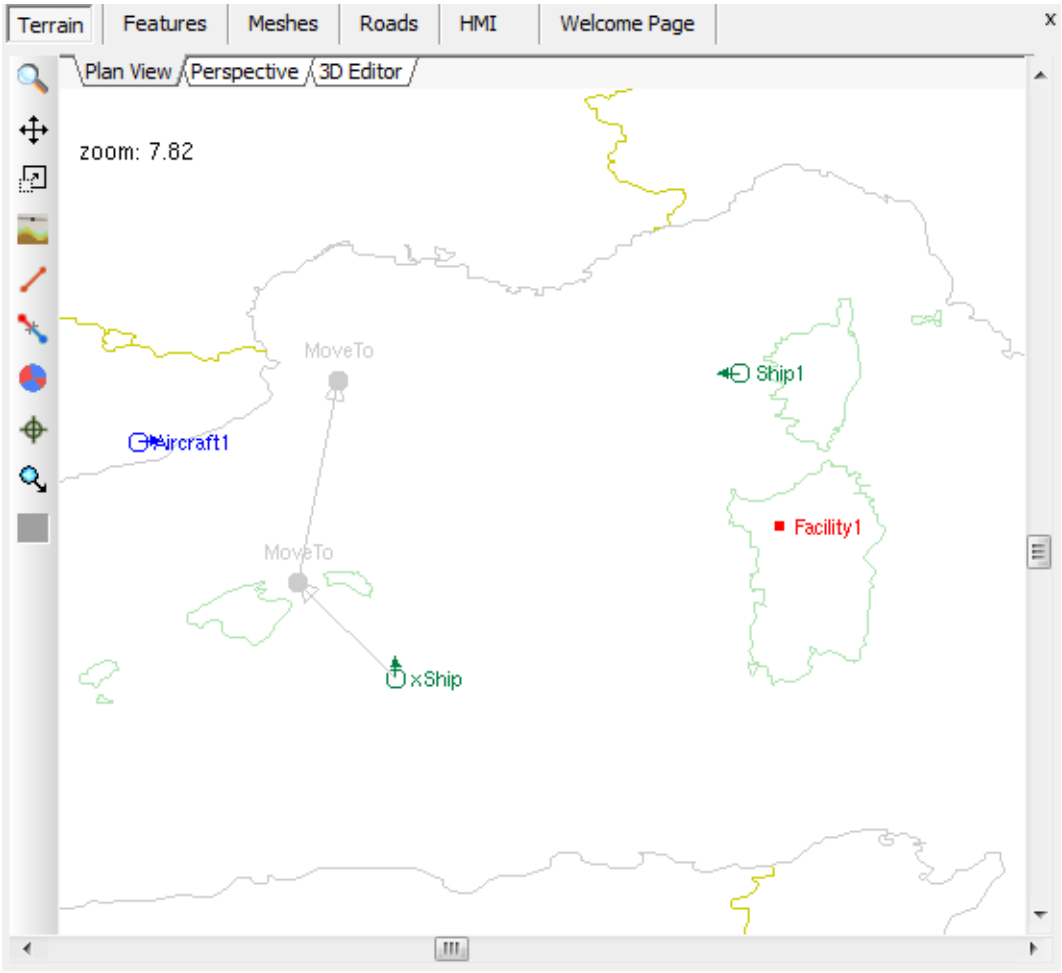
## 11 Help



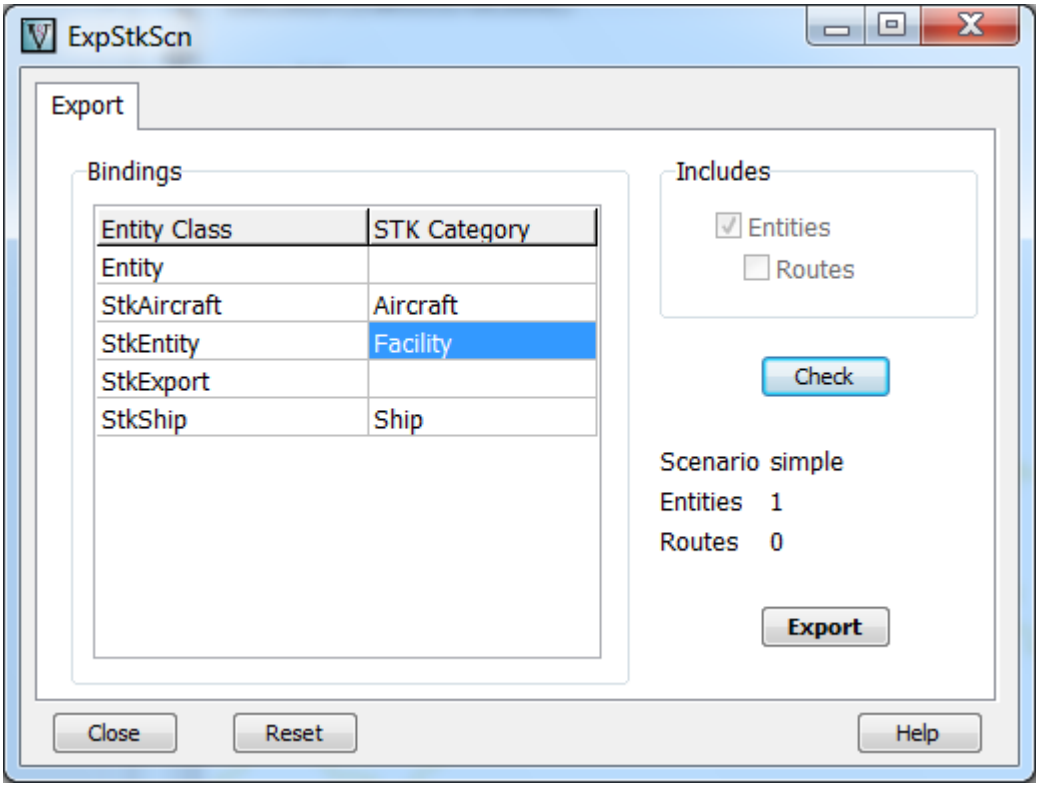

Put here all text explanations or documentation content needed to maintain this Viewer.
The *Help* content is saved but is not inserted into the generated code.
It will be used for the automatic document generation (see Make Documents)

# Exporting Entities

Create a vsTASKER local entity with a route:



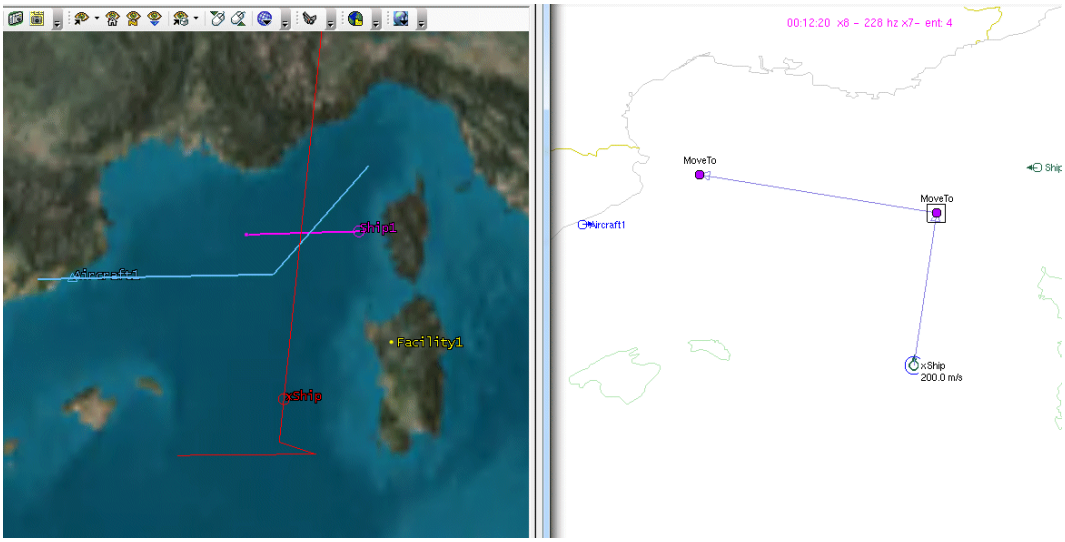

Export it to STK:




For detail explanations on the window, refer here.

Here is what we got at running time, on STK window:

## Exporting Entities



# GL-Studio

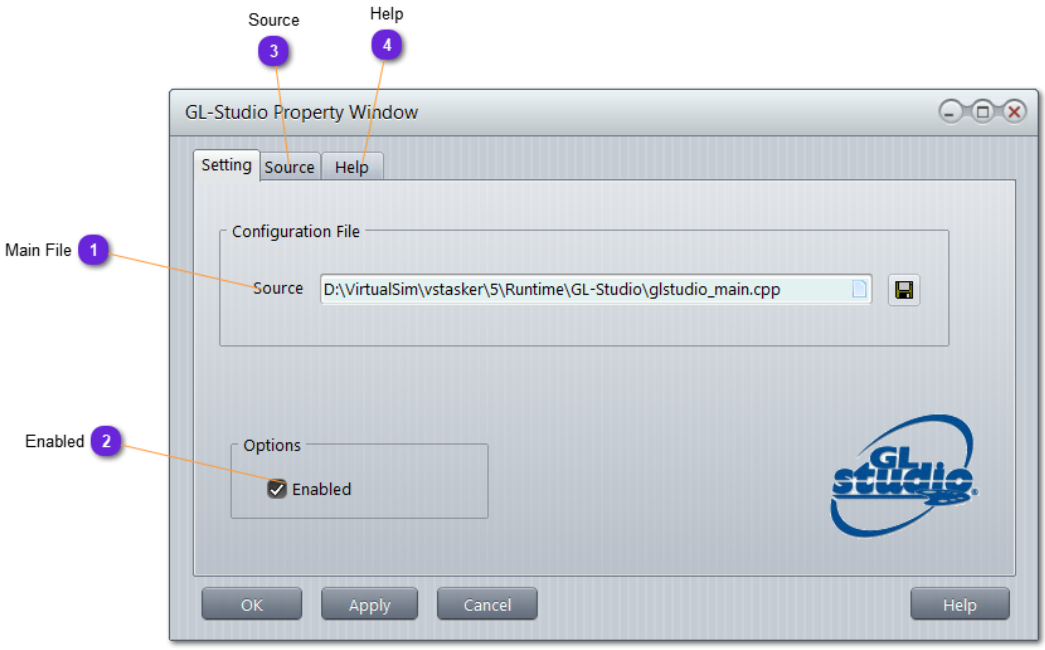
GL Studio user interface development software delivers high fidelity, feature-rich 2D and 3D graphical interfaces regardless of the product domain or industry. vsTASKER gives the capability to generate code to control GL-Studio graphic models, from logic using GL-Studio Objects (models).

See here for how to use.

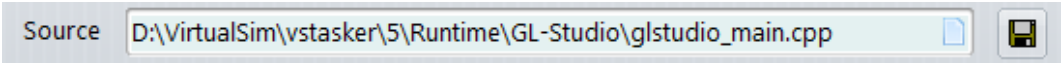

*GL-Studio software and license must be requested from a proper vendor (DiSTI), as vsTASKER does not provide them nor behave as a reseller or does support line for the product.*

Refer to the **Tutorial document** to learn how to develop your first GL-Studio simulation.

# Properties

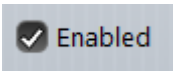



## 1 Main File



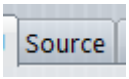

Specifies the C++ wrapper used to run the GL-Studio Objects defined in Models. The default provided one can be found in runtime/gl-studio/glstudio_main.cpp.
Once loaded, the file can be modified in the Source panel.

## 2 Enabled




One viewer must be at least enabled for the simulation to compile. Enabling one viewer disable all the other one.
When unchecked, all specific viewer code (and third party software API) are not generated and do not impact the makefile.

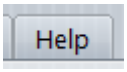
## 3 Source




Source code can be modified into this window or directly from Visual Studio from the solution (or any other text editor).
If code is modified, it will be automatically saved when OK button is depressed.
It is a good idea to save the modified code under another file to keep the original one unchanged. Use  button for that.
Refer to "Understanding Runtime" in the Tutorial document of vsTASKER.


*Do not forget that all provided source code in vsTASKER environment is subject to change from one revision to one another. So, it is a good practice to change the name of any cpp wrapper if any change must be applied on it.*

## 4 Help



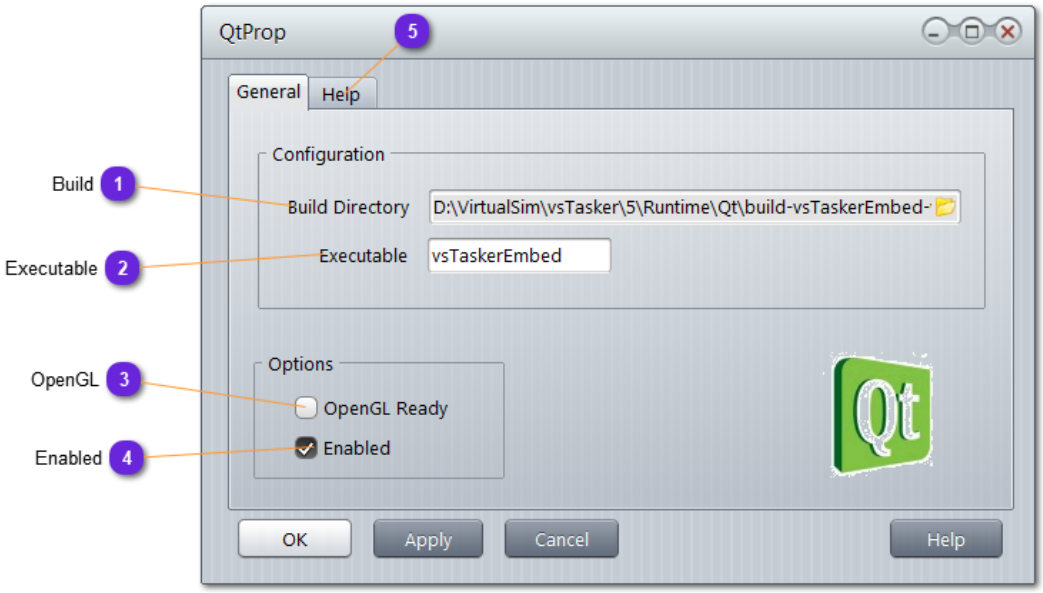

Put here all text explanations or documentation content needed to maintain this Viewer.
The *Help* content is saved but is not inserted into the generated code.
It will be used for the automatic document generation (see Make Documents)

# Qt

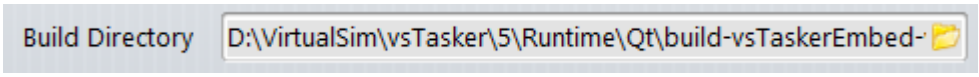
vsTASKER can use a Qt viewer to output code in such a way that the main will be hold into a Qt callback.
Qt viewer can be used when the simulation engine needs a simple user interface to control and interact with the scenario.
Use SimpleMap viewer (and Qt application in /Runtime/Qt/SimQt) to generate an embedded simulation engine into a framework Qt UI including mapping.

Refer to the **Tutorial** and **Developer** manuals to learn how to develop your first Qt simulation and how to embed vsTASKER DLL and libraries.

# Properties

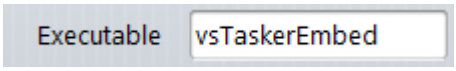



## 1 Build



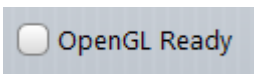

Specify here the directory where Qt generates the build (in either release or debug mode).
vsTASKER relies on the Qt makefile to rebuild the Qt application (and the embedded simulation engine).

## 2 Executable



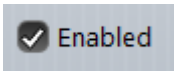

Name of the executable as set in the Qt project file.
vsTASKER will launch the exe name after compilation, in the build directory.

## 3 OpenGL



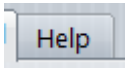

When checked, vsTASKER generated code will enable the OpenGL libraries.
Useful for OpenGL display inside the Qt window.

## 4 Enabled




One viewer must be at least enabled for the simulation to compile. Enabling one viewer disable all the other one.
When unchecked, all specific viewer code (and third party software API) are not generated and do not impact the makefile.

## 5 Help

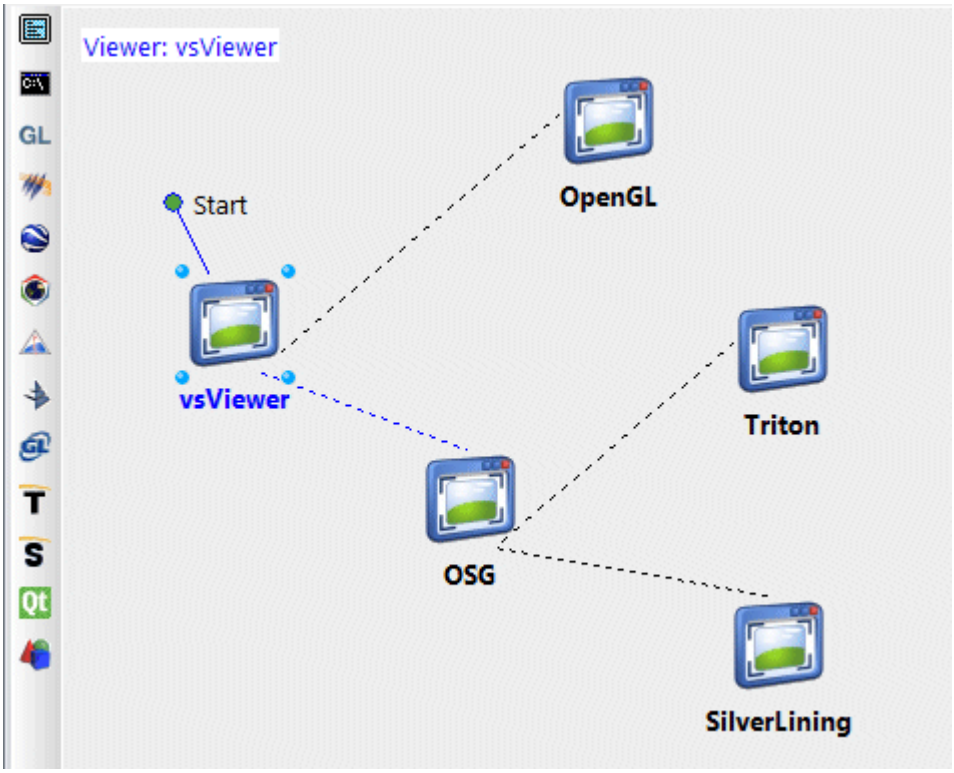



Put here all text explanations or documentation content needed to maintain this Viewer.
The *Help* content is saved but is not inserted into the generated code.
It will be used for the automatic document generation (see Make Documents)

# SimQt

SimQt is a predefined Qt application, ready to embed vsTASKER simulation database (runtime and generated code) and providing 2D OpenGL map, 3D OSG view and the vsTASKER glut HMI based on sprites.

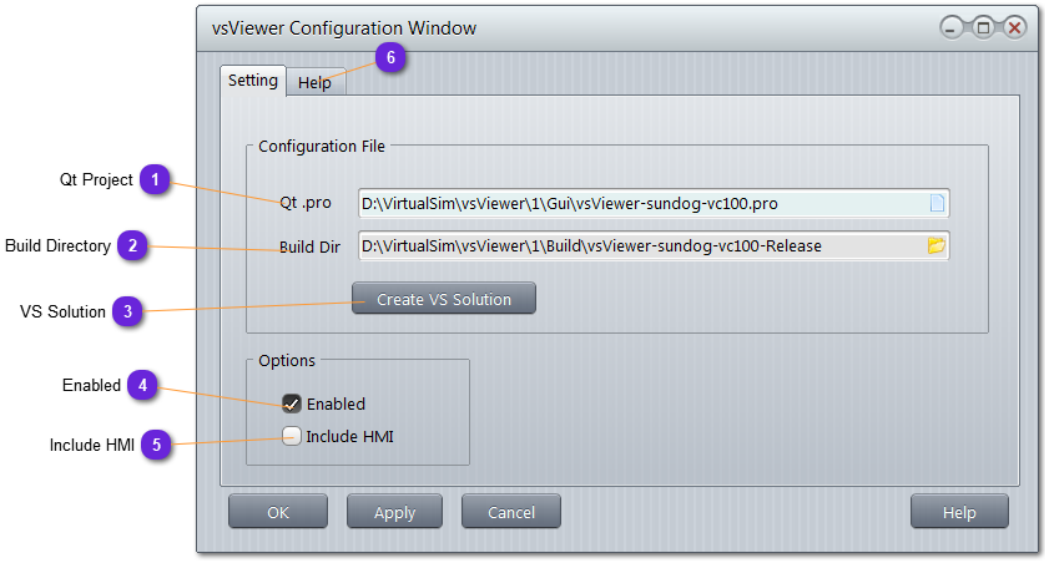
As seen in the image below, SimQt can combine one OpenGL viewer for the 2D/ Globe terrain map and an OSG viewer for the 3D scene.
This OSG viewer can also use Triton and SilverLining viewer.



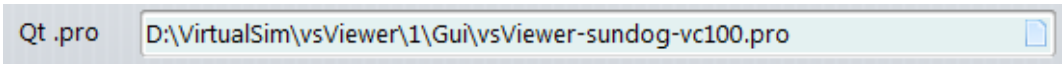

The panel developed in Qt is free and provided as is for the end user to start with a setup environment. User is free to develop his own based on the actual template.

Refer to the chapter **Embedding the SIM** in the **Developer Guide**.

# Properties

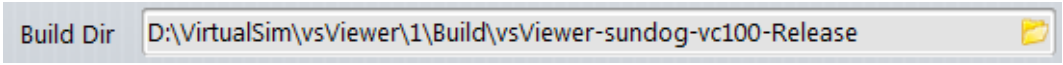



## 1 Qt Project



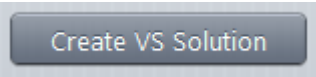

Specify here the project file used by vsVIEWER to run the current scenario database.
The file might be parsed by vsTASKER to extract useful setting.

## 2 Build Directory



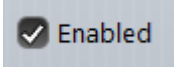

Specify here the directory where Qt generates the build of vsVIEER (in either release or debug mode).
vsTASKER relies on the Qt makefile to rebuild a new vsVIEWER application including the simulation engine.

## 3 VS Solution



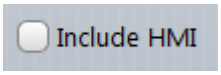

Use this button to produce or update a Visual Studio solution file based on the actual Qt project.
You can use the sln/vcproj solution instead of Qt Creator (Visual Studio is better on the debug side). Nevertheless, you will need to go back to Qt Creator when you change the UI.

## 4 Enabled



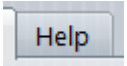

One viewer must be at least enabled for the simulation to compile. Enabling one viewer disable all the other one.
When unchecked, all specific viewer code (and third party software API) are not generated and do not impact the makefile.

## 5 Include HMI




If checked, vsVIEWER will activate the HMI panel and code if an HMI has been defined in the Database.
If no HMI is defined, this option is ignored.

## 6 Help



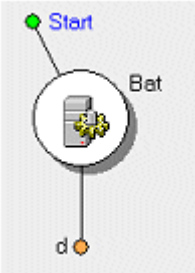

Put here all text explanations or documentation content needed to maintain this Viewer.
The *Help* content is saved but is not inserted into the generated code.
It will be used for the automatic document generation (see Make Documents)

# Analysis

Category for simulation analysis, outside real-time scheme.

# Batch

A Batch object manages a set of simulation runs for a given scenario, changing input variables and analyzing outputs to adjust inputs for a search of optimum or specific goals.
Batch objects can be linked together to perform various processes.
Optimization strategies are not the only use for batches, hundreds or thousands of runs of a single scenario can prove some outcomes in spite of the numerous stochastic models included in the simulation.




A Batch object is persistent compared to the Scenario object. It contains variables that persists the run (start-stop) of a scenario.
These variables can then store or accumulate data for later use or statistics.
It is not the Batch itself that start and stop a given scenario. The RTC is first ticking the current Batch (runtime part) then starts the associated Scenario. From inside the Scenario, the current (or any other) Batch can be retrieved to store or read any of its (persistent) variables. Reading some variables might be used to configure the scenario accordingly to computations or assumptions made in the Batch itself. Writing values to Batch variables will be done during the course of the scenario or at the end for subsequent analysis.
**The Scenario must stop itself !** The RTC nor the Batch won't interrupt a scenario run. It is mandatory to implement inside the scenario a condition for stopping it. This condition can be put into a Logic attached to the Scenario Player.

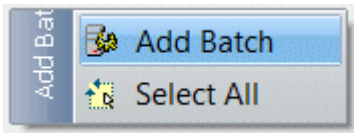

*Batches are raw mechanism to perform data analysis like gradient base searches or heuristic methods. It is up to the user to implement such mechanisms in the batch themselves.*

## • How to Use

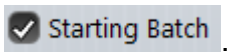
In this sample, we will run a batch that will start in sequence **5 scenarios**. Each scenario **creates** an arbitrary number of **entities**, specified by the batch, then waits for **15 seconds** and terminates.
When a scenario ends, the batch is reactivated and cycle again.

First, we create the batch object:

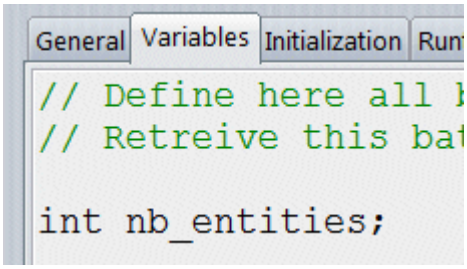
### Batch




Then, we open its property windows and rename it "myBatch" and make it active:



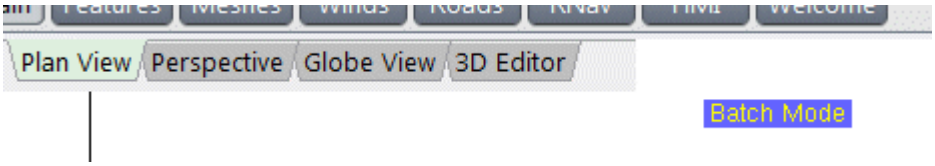

We set **5 runs**.
In Variable panel, we declare the number of entities to create by each scenario:



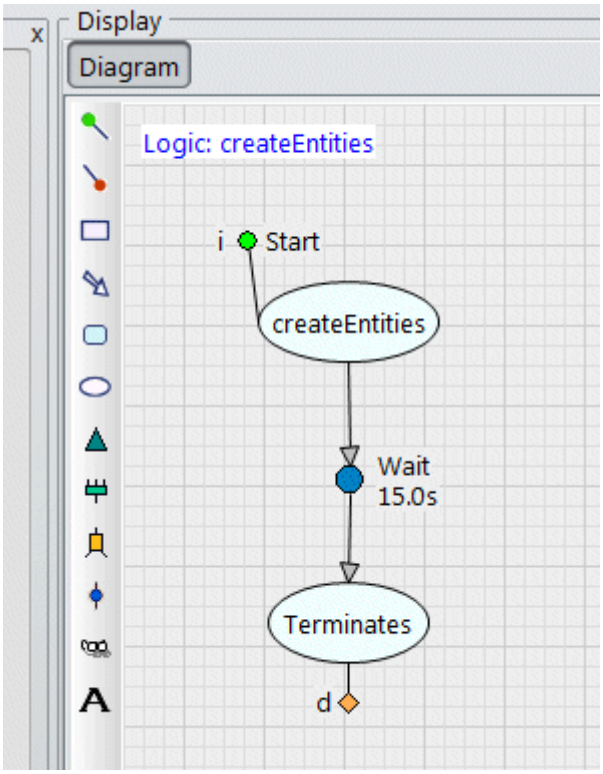

In the Runtime panel, we put this code:

```
printf("Run %d\n", cIter());
nb_entities = RANDOM(1,10);
Sleep(1000); // to give GUI some time to clean up
```

Note that Batch Mode is enabled and visible on the terrain map:




Now, we are ready to setup a scenario that will extract from the batch the number of entities to create, then terminates.
For that, we creates a logic that will be given to the player.




For Action createEntities:

```
 myBatch_BatDef* batch = (myBatch_BatDef*) R
>find("myBatch");
 if (batch) {
    printf("Scenario creates %d entities\n",
    for (int i=0; i<batch->nb_entities; i++)
       WCoord pos;
       pos.setRandom(300);
       new Entity("default", pos);
    }
 }
```

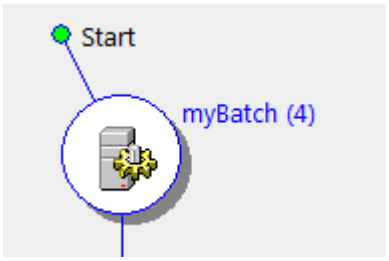
For Action Terminates:

```
  if (condition == true) {  // condition ==
to stop
    puts("Scenario stops!");
    R:stop();
 }
```

Generate the simulation engine and run.
The console will display:

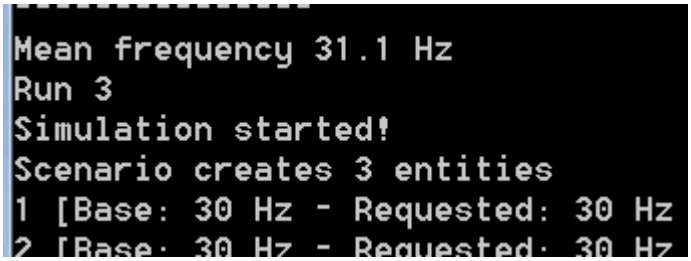
```
Having loaded environnement for: test_
Simulation Engine is ready
Run 1
```

Start the simulation  from the GUI, then display the Batch symbol in Environment::Analysis.
The current `cIter()` value (current run number) is displayed in parenthesis right to the batch name.

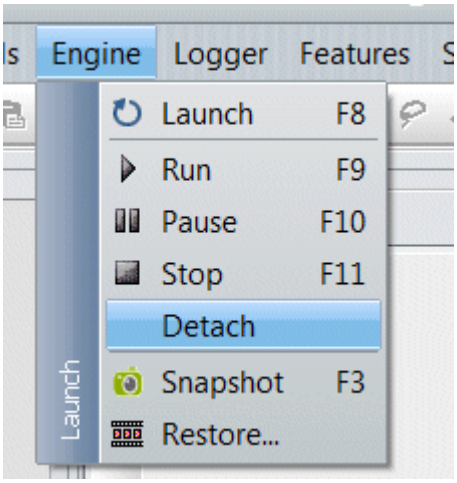


### Batch

Terrain map will display random entities created at each run, for 15 seconds.




To **stop** a Batch run, using the button only terminates the current scenario but the batch will restart it until all runs expire or if the Runtime part returns QUIT.
To force the batch mode to terminate, use the **Detach** option:

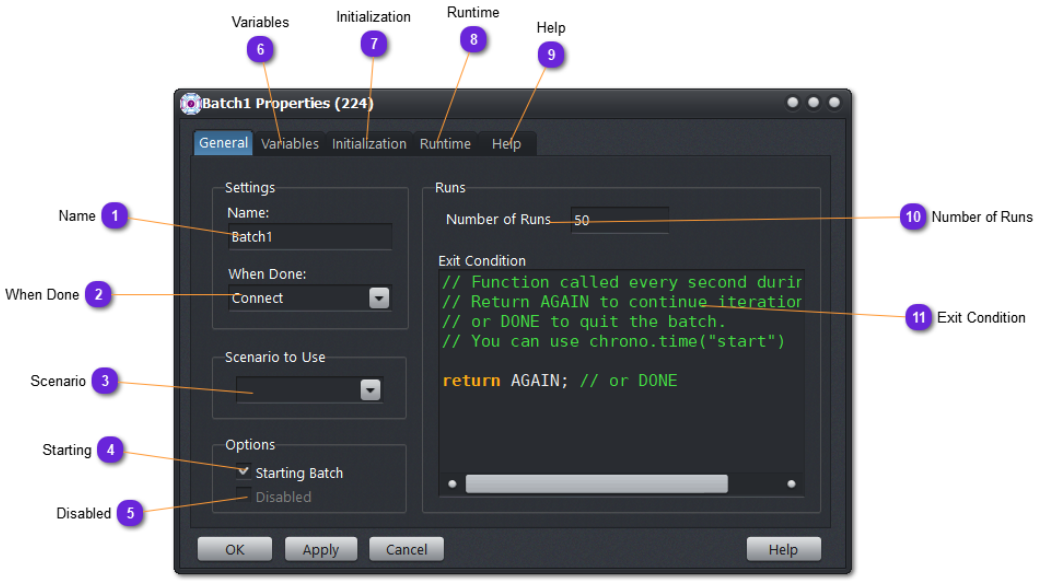



*A simple way to stop scenario execution is to use the Code::Runtime part of the Scenario or by giving a Logic to the Scenario Player. In all situations, when the exit condition occurs, the following code will force the scenario run to stop:* `R:stop();`
*The Exit Condition, called every second, can also be used to stop a scenario and iterates the batch object.*

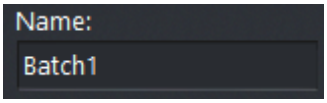
## • Code Hints

***scen()*** or ***S:*** is a macro that returns a pointer to the Scenario instance.
***R:*** is a macro that returns a pointer to the Runtime Controller (`Vt_RTC`).

# Properties

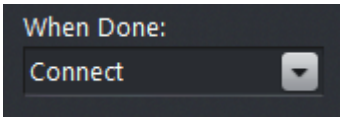



## 1 Name




Name of the Batch object. Must be unique.
The name will be used to create a class. ie. Batch named Analysis1 will create a class named `Analysis1_BatDef`

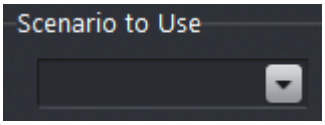
## 2 When Done




Specify what will happen next, when the number of runs will be reached or when Exit Condition will return DONE:

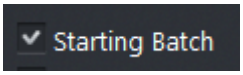
- **Connect**: when the batch object is connected to another batch object
- **Done**: nothing will happen. The batch will just stop
- **Exit**: vsTASKER will exit.

## 3 Scenario



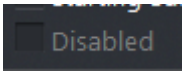

Specify which scenario will start under the Batch object.
Only one scenario can run per Batch object.

## 4 Starting



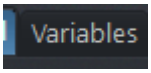

Checked if this batch starts the Batch Mode.

## 5 Disabled



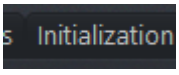

If checked, the Batch object will be skipped. If it is the starting one, the Batch Mode is also disabled.

## 6 Variables




Put in this panel all variables that will be used by the Batch object itself but also by any other simulation objects. Typically, the Batch object contains various variables that change at each iteration, based on various user strategies or based on result of the past runs. Each new run might use the Batch variables to setup their initial values.

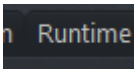
## 7 Initialization




Part called **before** the **first** iteration of the Batch object (`RESET`)

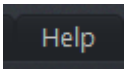

*Refer to the Developer Guide for more details on system phases/events.*

## 8 Runtime



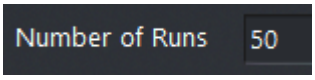

Part called as soon as the called scenario terminates.
Do here all the variable modification requested for the next scenario run.
Must return AGAIN if a new batch iteration is requested.
Return DONE if the batch object must be left. Next connected Batch object will be activated. If none, batch mode is terminated.
Return QUIT if the batch mode must be terminated.
To check the number of iterations, use variable `cIter()` inside the code.

## 9 Help



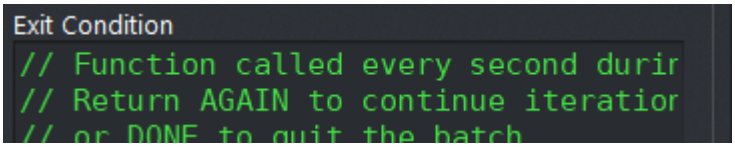

Put here all text explanations or documentation content needed to maintain this Viewer.
The *Help* content is saved but is not inserted into the generated code.
It will be used for the automatic document generation (see Make Documents)

## 10 Number of Runs




If value is provided, it will stop the Batch object processing after the given number of iterations.

## 11 Exit Condition




Put here the code that will be evaluated every second during the simulation run.
If code returns `AGAIN`, the Batch object remains active. If it returns `DONE`, the batch object stops the scenario the same way as using `R:stop()` into the code.
Runtime code of the Batch is then called (9).

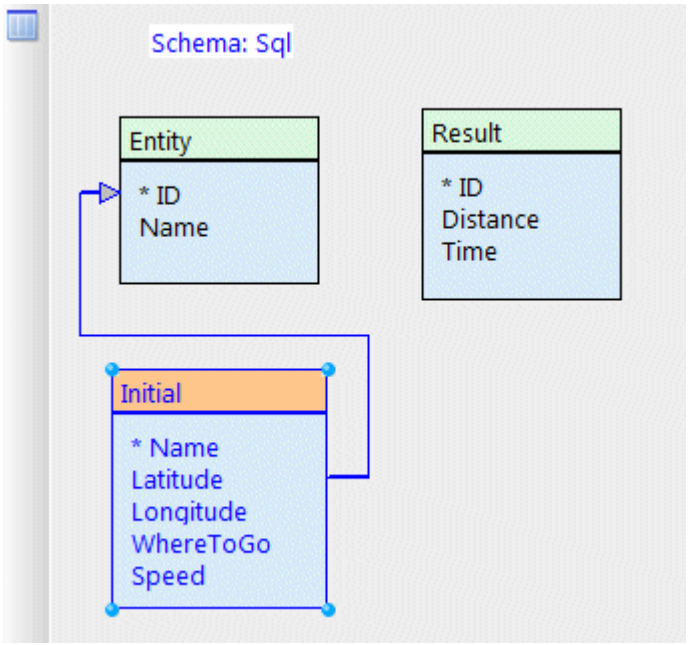
# SQL Support

vsTASKER allows capability to define SQL tables and connect to a mySQL or MariaDB database/schema (local or remote) either to retrieve data or to store results.



*In mySQL, physically, a schema is synonymous with a database. You can substitute the keyword SCHEMA instead of DATABASE in SQL syntax. Some other database products draw a distinction. For example, in the Oracle Database product, a schema represents only a part of a database: the tables and other objects owned by a single user.*

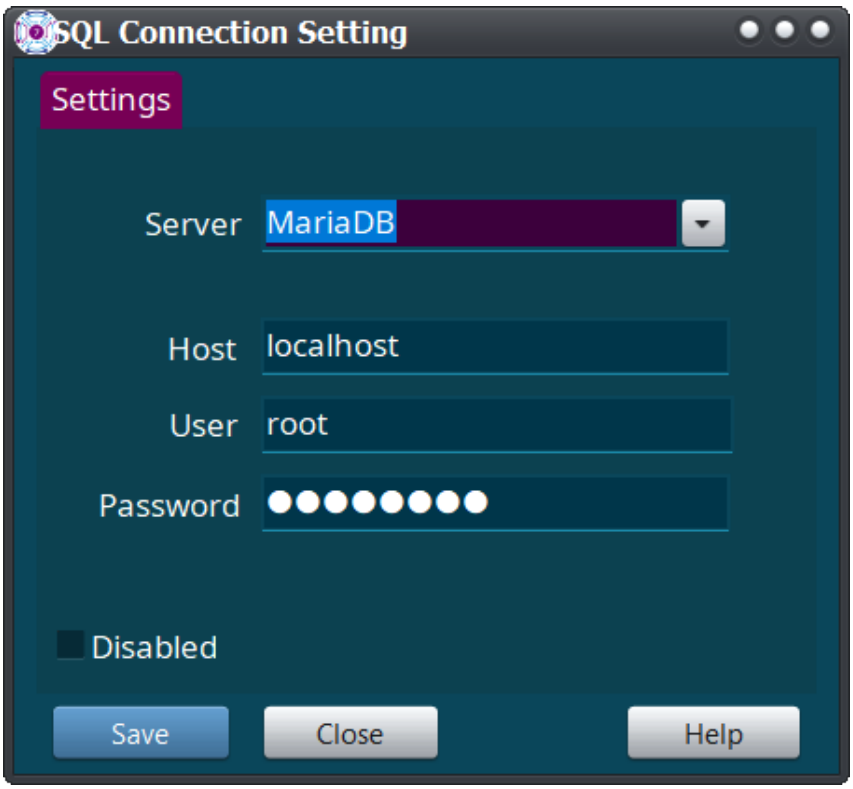
From the SQL environment, user can define simple tables and columns. vsTASKER will generate all necessary code to facilitate the access of the data and the connection to the SQL database.




## • Connection to Server

Once in the SQL environment, right click the mouse button and select Properties.

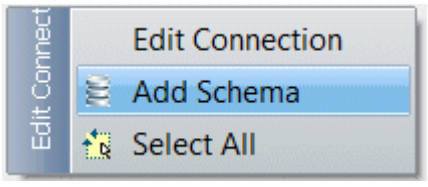
You will get the following window:




Select the server MariaDB. The generated code will adapt to the choice and will use default configuration settings. For more information on the use of the supported SQL server, refer to the **Tutorial**.

## • Popup Menu

Outside an SQL schema:



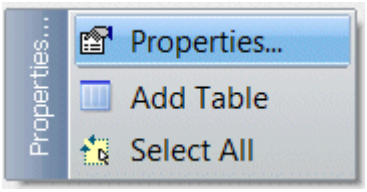

**Edit Connection**: setup the access to the SQL server. See here.

**Add Schema**: create a new schema to store Tables. Normally, only SQL schema must be enabled at a same time, all others must be disabled.

**Select All**: select all defined SQL schemas (useful for removal)

Inside an SQL schema:

### SQL Support




**Properties**: SQL schema setup.

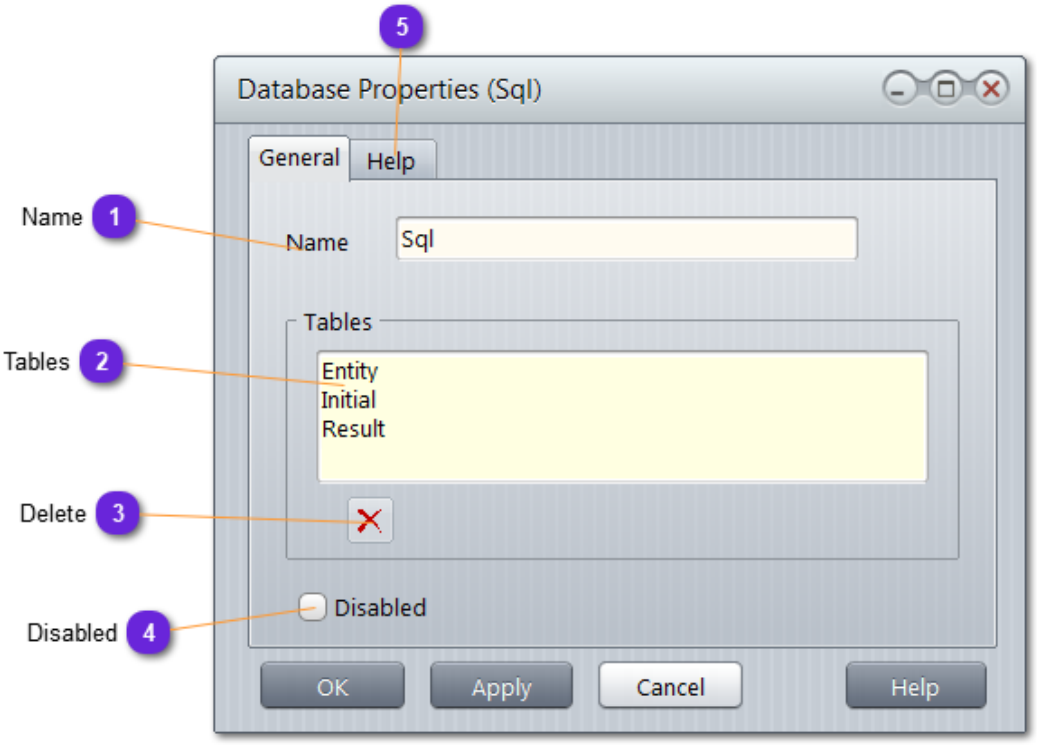
**Add Table**: add a new Table into the SQL schema. See here.

**Select All**: select all defined Tables (useful for removal)

Refer to the **Tutorial** to learn how to develop your first SQL simulation.

# Schema

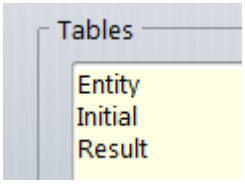
Use this window to change the name of the schema or delete some tables.




## 1 Name


Name of the database as it must exist on the server. vsTASKER does not create databases.
The name is case insensitive.

## 2 Tables

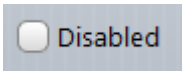



List of all tables belonging to the schema.

## 3 Delete

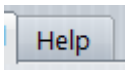
Use this button to remove the selected table in the list above.

## 4 Disabled




Only one schema must be active. Disable all schemas but one.

## 5 Help

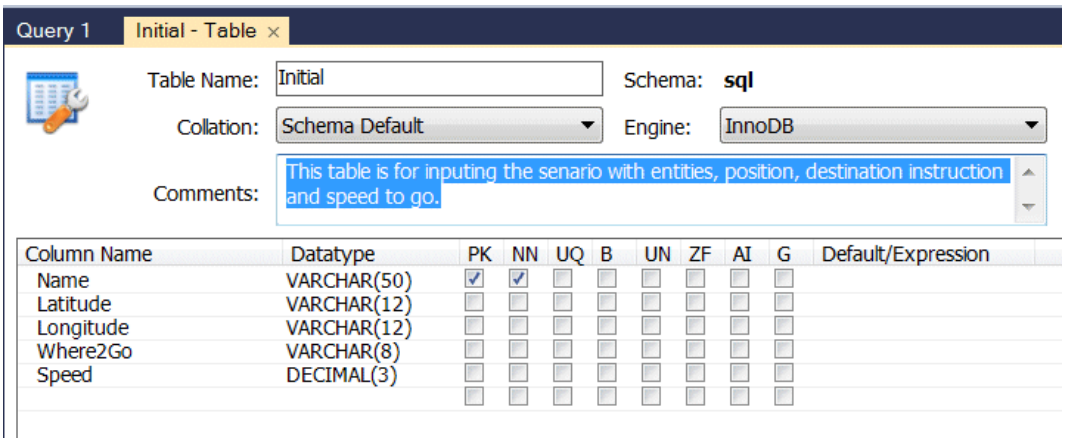



Put here whatever description of information useful to the designer to understand what the State is about.
The help part is also used in the hint and the automatic document generation (see Make Documents).

# Table

Schema tables in vsTASKER replicate the SQL tables in their structure and flags. Open mode tables are loaded  at simulation start while Create table are flushed to the server.

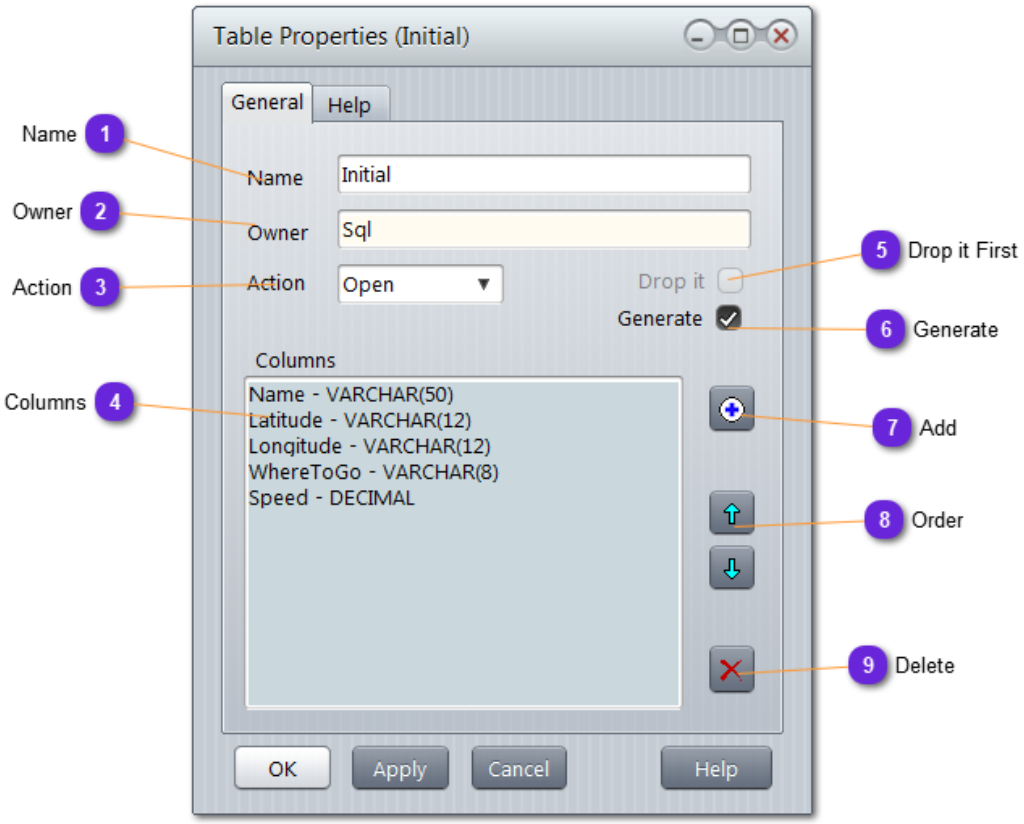

Here, the same Initial table in vsTASKER (above) and in Workbench (below) with the same columns and settings.




Create table are not stored in vsTASKER memory in default mode. vsTASKER will automatically create them in the server (if not already there) using a SQL command. For the above table, the generated command will be as follow:

```
CREATE TABLE `sql`.`initial` (
  `Name` VARCHAR(50) NOT NULL,
  `Latitude` VARCHAR(12) NULL,
  `Longitude` VARCHAR(12) NULL,
  `Where2Go` VARCHAR(8) NULL,
  `Speed` DECIMAL(3) NULL,
PRIMARY KEY (`Name`));
```

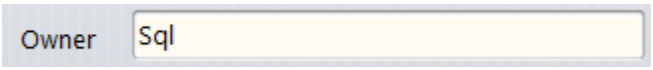
## Table




## 1 Name

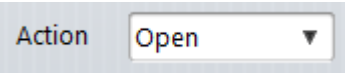

Name of the table.
If SQL server option `lower_case_table_name` is set to **2**, name is not case sensitive. If set to **1**, name must be lowercase.

## 2 Owner




Name of the schema the table belongs to.

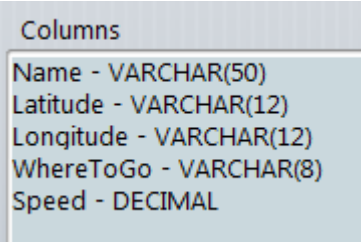
## 3 Action




Set here what vsTASKER shall do with the table:

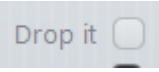
- **Open**: Load the table at simulation start and fill an array with the data.
- **Create**: Generate an SQL command to create the table on the server database.

## 4 Columns



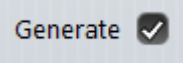

List of all the columns of the table.
Double click any item of the list to display its property window.

## 5 Drop it First




If checked, the table if first removed from the server (with its data) prior to be created.
Use it when each run shall provide a new set.
Uncheck it to append data to the table.
Only for Create mode table.

## 6 Generate




If checked, the table structure will be generated and the SQL table will be loaded as an array of the generated structure, will all the data queried from the server.
If unchecked, the SQL table will not be loaded and the user will have to do the query itself, using SQL commands (see Writing to Table here)
Only for Open mode table.

## 7 Add

Add a new column to the table, at the end.

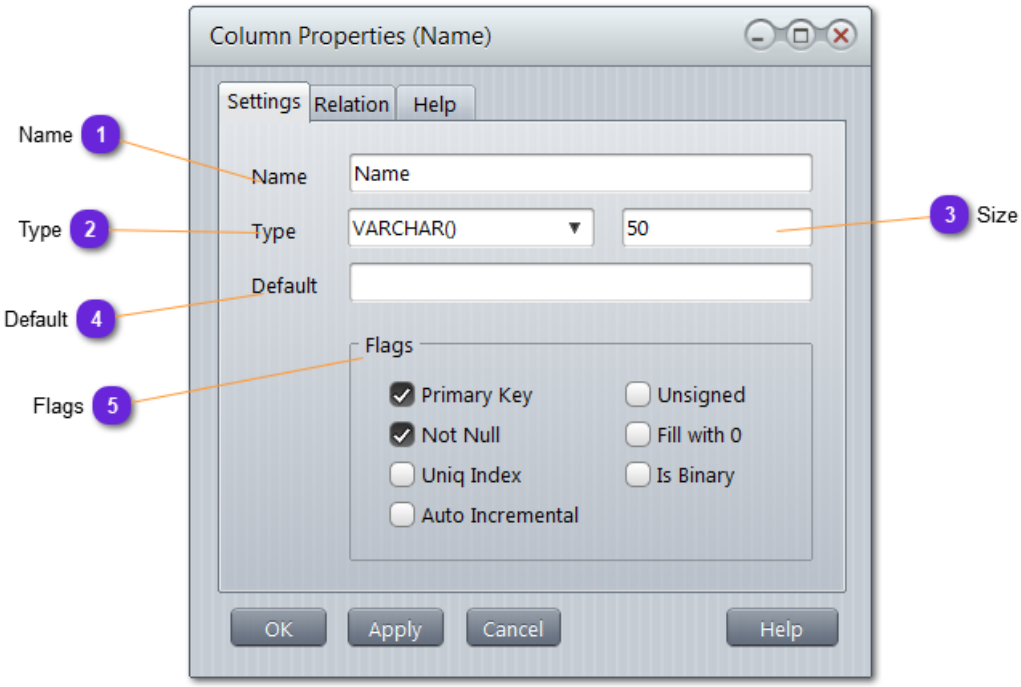
## 8 Order

Select one column then use the  or  button to move the column up or down the list.

## 9 Delete

Use this button to delete the selected column.
There is no undo.

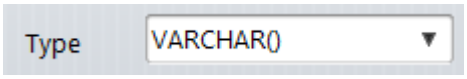
# Column




## 1 Name


Name of the field, must be unique in the table.

## 2 Type




Select from the drop down list the type of the field.
vsTASKER provides less predefined types than SQL.

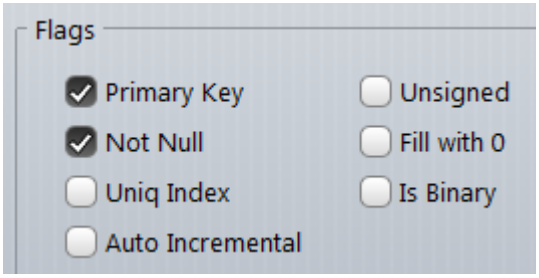
## 3 Size


When the selected type must include a size, specify it here.
In Workbench, the size is expressed between the parenthesis. In vsTASKER, it must be set aside.

## 4 Default


Set here the default value for the field, if any or needed.

## 5 Flags

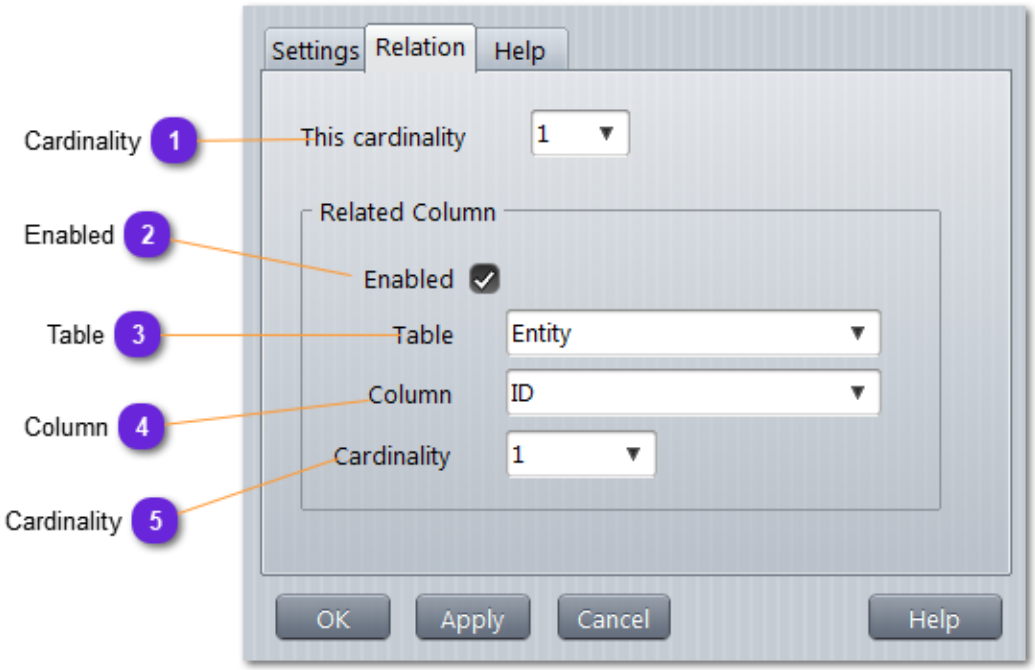



- **Primary Key**: used for sorting or retrieving the field.
- **Not Null**: data shall always be provided
- **Uniq Index**: Create/remove Unique Key
- **Auto Increment**: only for number, like an ID, is automatically incremented for each new data entry.
- **Unsigned**: non-negative numbers only. so if the range is -500 to 500, instead its 0 - 1000, the range is the same but it starts at 0.
- **Fill with 0**: if the length is 5 like INT(5) then every field is filled with 0's to the 5th value. 12 = 00012, 400 = 00400, etc.
- **Is Binary**: stores data as binary strings. There is no character set so sorting and comparison is based on the numeric values of the bytes in the values.

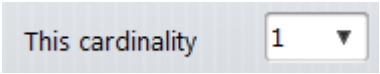

*Refer to mySQL documentation for more information on these flags.*

# Relation

Allow connecting this column with another column of another table, including the cardinality.
The relation will be shown with a line on the diagram.

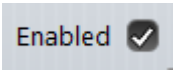



## 1 Cardinality




Specify from the drop down the current cardinality of the relationship with the other field: **1**-1 or **1**-n or **n**-1 or **n**-n.

## 2 Enabled




Check this to specify the target of the relationship. If uncheck, there will be no relation (default).

## 3 Table


Select the table of the field (column) of the relationship.

## 4 Column

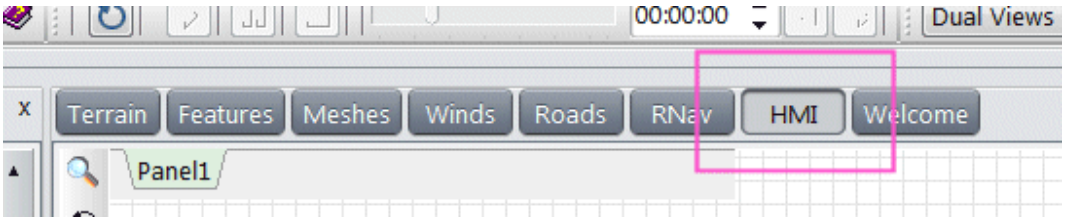

Select the field (column) in the selected table (above).

## 5 Cardinality


Specify from the drop down the target cardinality of the relationship with the current field: 1-**1** or 1-**n** or n-**1** or n-**n**.

# HMI Builder

Most of the time, simulation needs a simple user interface to interact with or display parameters.
vsTASKER offers such capability with its embedded HMI builder.

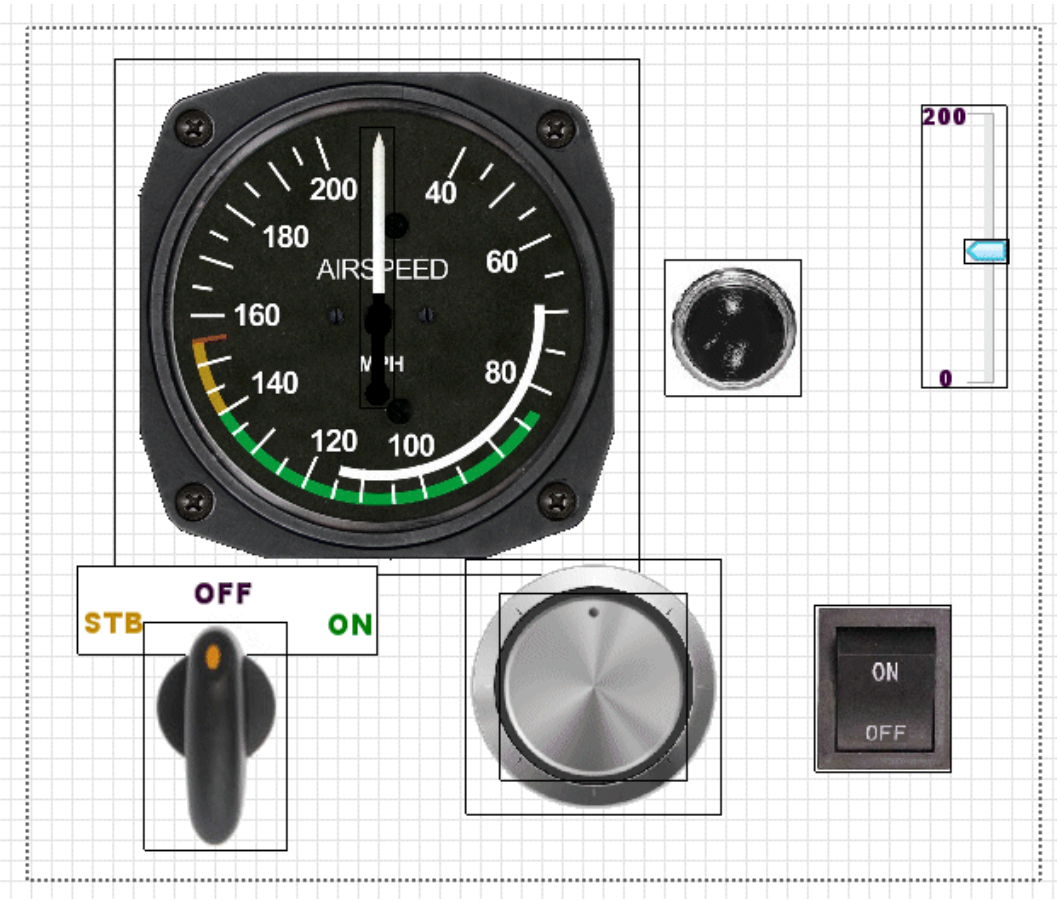



The HMI is based on graphical objects called Sprites.
They are divided into two categories:

- Output: sprite is activated by a value. A rotator, a lamp, a GDI are output sprites.
- Input: sprite provides a value on a user action. A knob, a slider are input sprites.

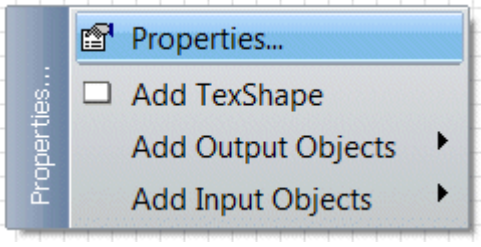
vsTASKER HMI will not replace high end HMI builders like VAPS or GL-Studio that are explicitly designed for such purpose.
Nevertheless, vsTASKER can use and embed VAPS or GL-Studio objects if such interfaces matter.

Several projects have used the embedded HMI provided by vsTASKER for building control panels, instructor consoles or practical and freely deployable standalone interactive simulation tool.




*The HMI is based on GLUT.*
*A simulation engine including HMI will generate a thread for the GLUT display. Thus, it is possible to use any Viewer combined with an HMI display.*

## • Popup Menu



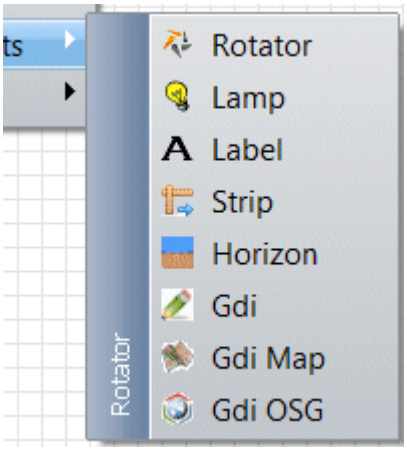

**Properties**: call the HMI property window

**Add TexShape**: add a Texture Shape (see here)

**Add Output Object**: see below

**Add Input Objects**: see below

**Output objects**:




**Rotator**: add a Rotator sprite (see here)

**Lamp**: add a Lamp sprite (see here)

**Label**: add a Label sprite (see here)

**Strip**: add a Strip sprite (see here)

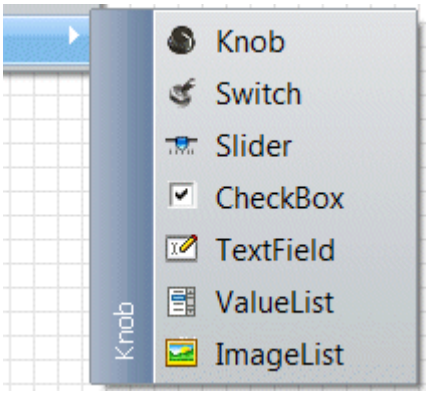
**Horizon**: add a Horizon sprite (see here)

**Gdi**: add a Gdi sprite (see here)

**Gdi Map**: add a Gdi Map sprite (see here)

**Gdi OSG**: add a Gdi OSG sprite (see here)

**Input objects**:




**Knob**: add a Knob sprite (see here)

**Switch**: add a Switch sprite (see here)

**Slider**: add a Slider sprite (see here)

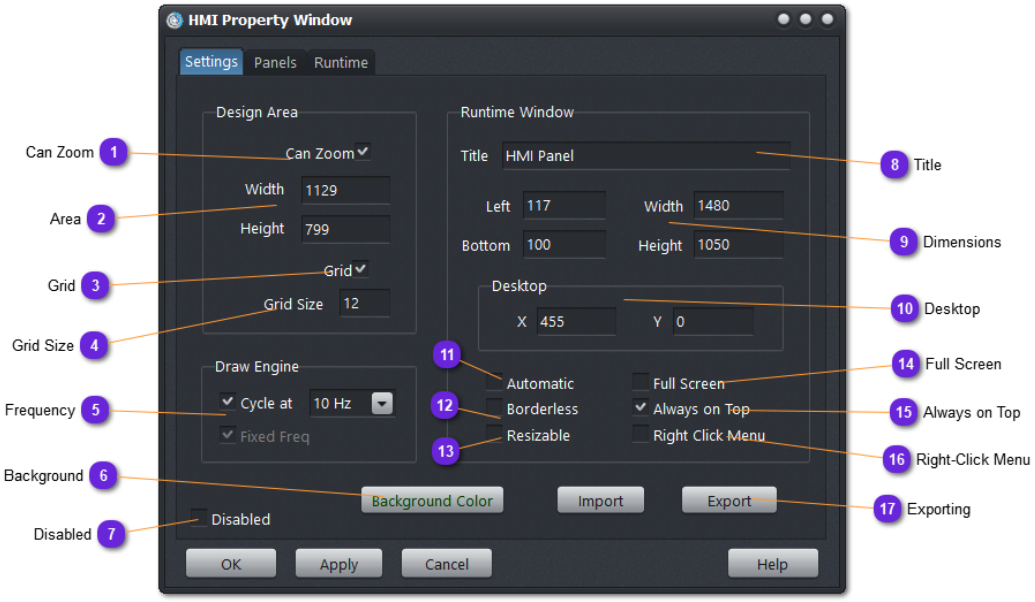
**CheckBox**: add a Check Box sprite (see here)

**TextField**: add a Text Field sprite (see here)

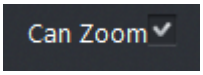
**ValueList**: add a Value List sprite (see here)

**ImageList**: add an Image List sprite (see here)

# Settings




## 1 Can Zoom

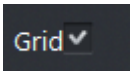



Check this option if the HMI design area can be zoomed or not. Default is checked.

## 2 Area

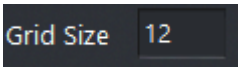

Dimension of the HMI design panel, as visible on the desktop. Read only.

## 3 Grid



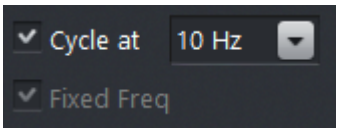

If checked, the HMI design work area will display a square grid for visually aligning Sprites.

## 4 Grid Size



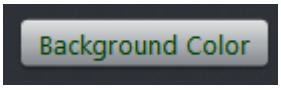

Specify the size of the grid square in pixels.

## 5 Frequency



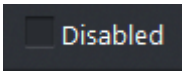

Set here the frequency at which the HMI thread will be called. This frequency will be the refresh rate of the HMI window. The lower the frequency, the jumpier the visual. For some panel that does not request high refresh rate, a low frequency will save CPU for the simulation engine (or the 3D stealth view).

## 6 Background



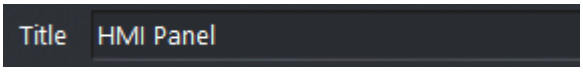

Set here the HMI background color applied before drawing any sprite object. Default is white.

## 7 Disabled



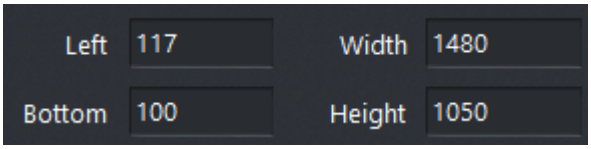

When checked, the HMI will not generate any code (even if Sprites exists) and it will be like deactivating it without erasing its content.
An empty HMI is like a disabled one.

## 8 Title

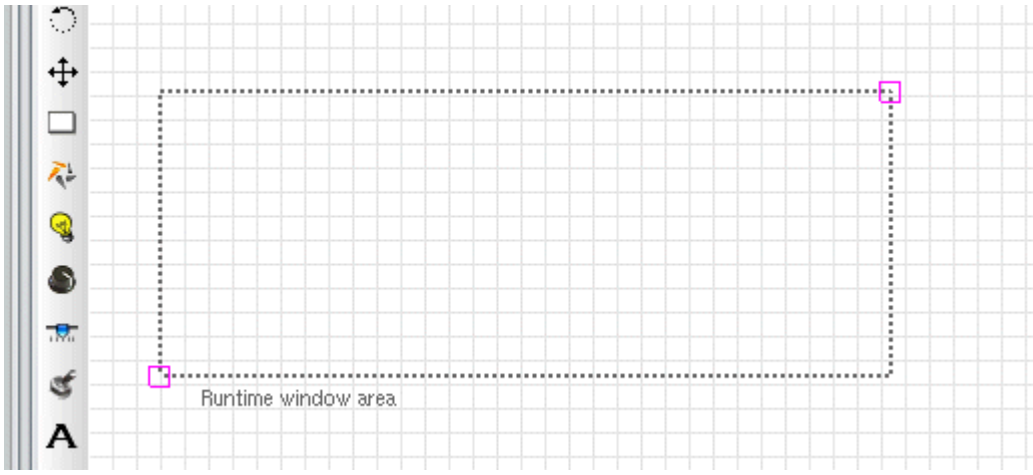



Caption of the HMI panel at runtime (window title).

## 9 Dimensions

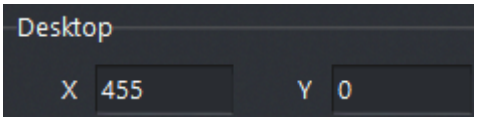



On the HMI panel, a rectangular dashed area represents the runtime (RT) window area. Sprites must be positioned inside to be visible.
If some Sprites are put outside the area, they will exist but not be visible. This can be used to replace some Sprites with some others just by exchanging their coordinates during runtime from the code.
Nevertheless, this area can be modified using the mouse or using the text fields.
To resize it using the mouse, just select the dashed line then drag any of the two magenta corners.



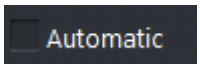

Left is the position of the left side of the RT window on the HMI panel, in pixels.
Bottom is the position of the bottom side of the RT window on the HMI panel, in pixels.
Width and Height express the size in pixels of the RT window.

## 10 Desktop



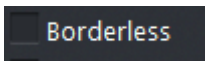

Position of the runtime HMI **top left** window on the desktop.

## 11 Automatic



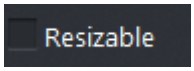

If checked, the runtime window will automatically adjust its size to gather all defined Sprites. Otherwise, the manually defined runtime (dashed) window will be used.

## 12 Borderless



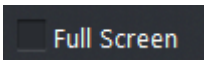

If checked, the runtime window will display no title bar and no border.
This can be useful for incrustation on a part of the desktop or full screen.

## 13 Resizable



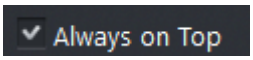

When checked, the runtime window can be resized using the mouse.

## 14 Full Screen



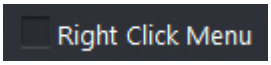

When checked, the runtime window will adjust its size to the desktop size.

## 15 Always on Top




When checked, the runtime window will try to remain always on top of other windows.
This is a Window setup flag.

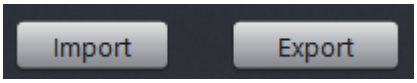
## 16 Right-Click Menu




Select this option if you want to use the general built-in contextual menu using the right mouse button.
Unselect it for no binding.

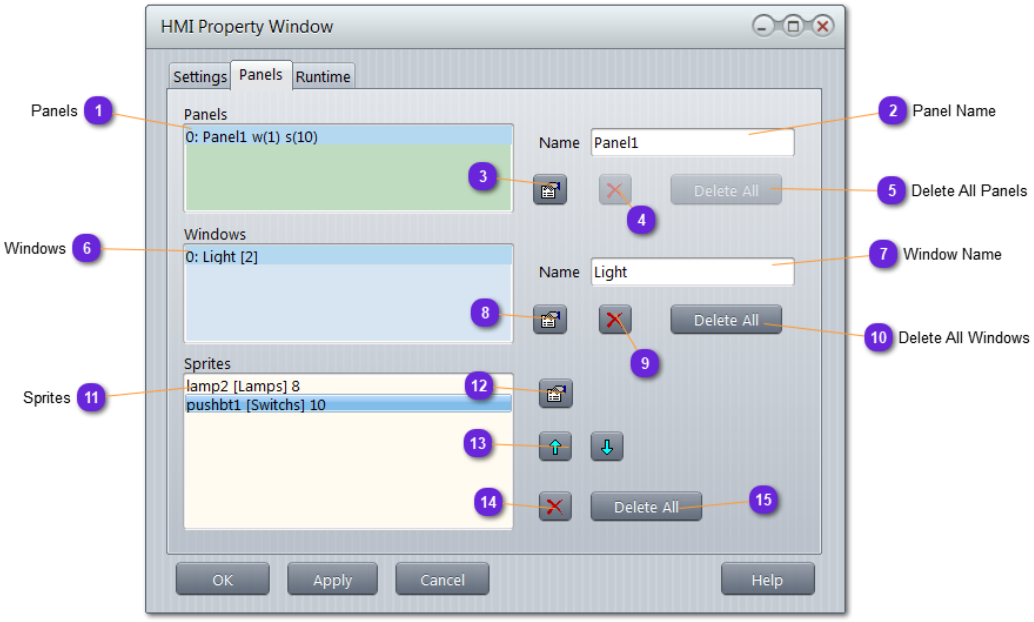

*The actual menu contains only the Quit option. It is Glut based so, it does freeze the display. A new OpenGL version will come in later versions for not freezing the simulation and allowing the user to control the content and the binding.*

## 17 Exporting

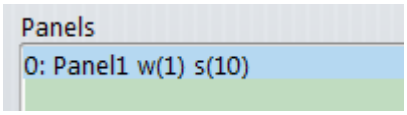



Use **Export** button to **save** the current HMI (including all Panels and Sprites) to a file in `/Data/Shared` with extension .hmi
Use **Import** button to **load** a saved HMI in replacement of the actual one (that will be erased).

# Panels




## 1 Panels




List all defined Panels for the HMI.
Each line shows the Panel **id**, its **Name**, the number of **Windows** and the total number of **Sprites**.
Only one allowed on the current version.

## 2 Panel Name


Name of the selected Panel.

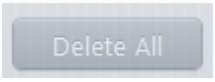
## 3 Panel Properties

Displays the property window of the selected Panel.
*Not available yet.*

## 4 Panel Delete

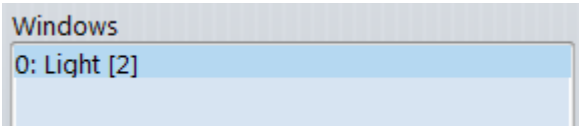
Remove the selected Panel from the HMI.
*Not available yet.*

## 5 Delete All Panels



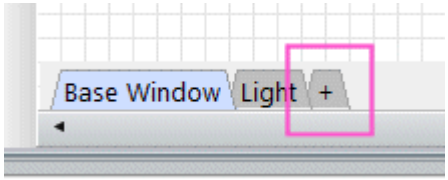

Remove all Panels from the HMI.
*Not available yet.*

## 6 Windows



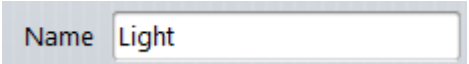

List all defined Windows for the selected Panel.
Each line shows the window **Name** and the number of **Sprites** it contains.

To create a new Window, use the + tab on the bottom tab list:




To edit a defined window, select the corresponding tab.
Select Base Window tab to go back to the main Panel.

## 7 Window Name




Name of the selected Panel Window.

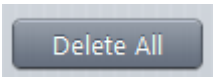
## 8 Window Properties

Pop up the property window of the selected Panel Window.

## 9 Window Delete

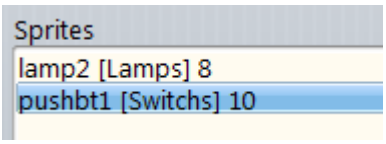
Removes the selected Panel Window and all its embedded Sprites.

## 10 Delete All Windows




Removes all Panel Windows and their embedded Sprites.

## 11 Sprites




List all defined Sprites for the selected Window or Panel.
To list only the Base Panel Sprite, select the Panel in the top list. This will unselect the Window list. If a Window is selected in the above list, only its Sprite list is displayed.
Each line shows the Sprite **Name**, the **Type** and its **id**.

## 12 Sprite Properties

Call the selected Sprite property window.

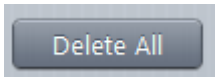
## 13 Rearrange

Move up or down the list the selected Sprite.
This can be useful for overlapping Sprites as they are **processed** and **displayed** from **top** to **bottom**.

## 14 Sprite Delete

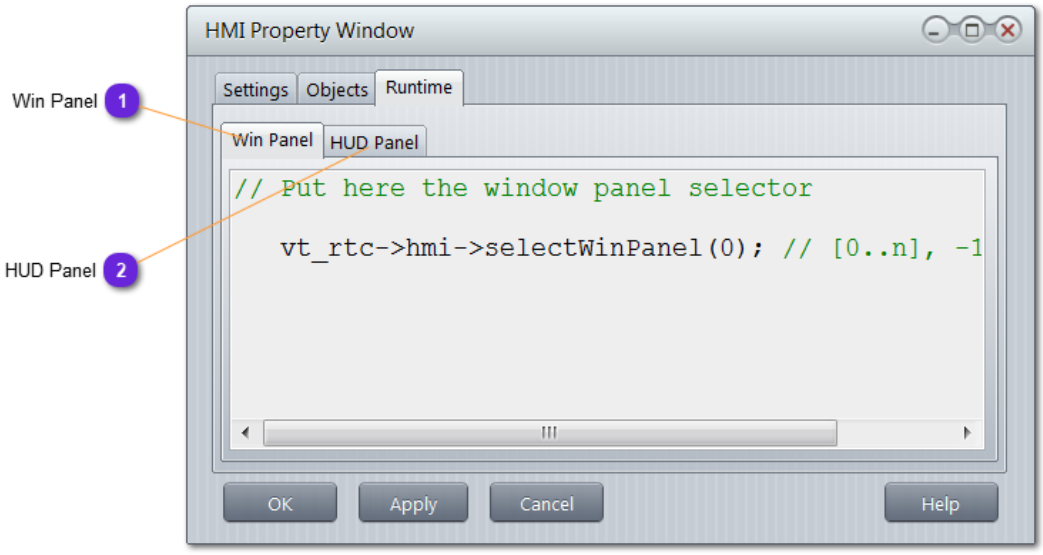
Remove the selected Sprites of the list.

## 15 Delete All Sprites

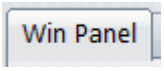



Remove all Sprites of the list.

# Runtime




## 1 Win Panel

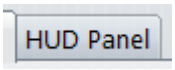



Put here the code that will be called at each HMI cycle, before the processing of the Sprites.
Must select the active panel (only one for the moment) using the code:

```
vt_rtc->hmi->selectWinPanel(0);
```


*This code is embedded into a generated function called* `hmi_display();`

# 2 HUD Panel




This code is only called from the OSG viewer, when 2D inlays must be applied on the OSG windows.
The actual HMI will then be drawn.
To disable this function, write this:

```
vt_rtc->hmi->selectHudPanel(-1);
```

To enable, when HMI HUD has to be drawn on the OSG Viewer display, do the following (given as example):

```
vt_rtc->hmi->selectHudPanel(0); // [0..n], -1 for none
//draw circles
hmi_draw->setTransparency(0.0);
hmi_draw->setColor(0.0, 0.0, 0.0); // black
hmi_draw->setLine(2);
hmi_draw->drawCircle(100, 100, 0, radius);
etc.
```


*This HUD code is embedded into a function called* `hud_display();`
*This function is called from Src/Osg/osg_hud.cpp in* `HMIDrawable::drawImplementation(...)`

# Sprites

Sprites are predefined and user-defined runtime graphical models added to vsTASKER. Sprites are DLL imported modules that register automatically at vsTASKER start. They have been defined with a GUI interface (typically using CodeGear C++ Builder) for the vsTASKER GUI (DLL output) and without any GUI for the simulation engine (lib output).

Sprites can be found in Sprite directory (DLL) and Lib directory for their runtime counterpart. The configuration file sprites.lst (in Sprite directory) lists all the sprites that are available and must be loaded at vsTASKER start.

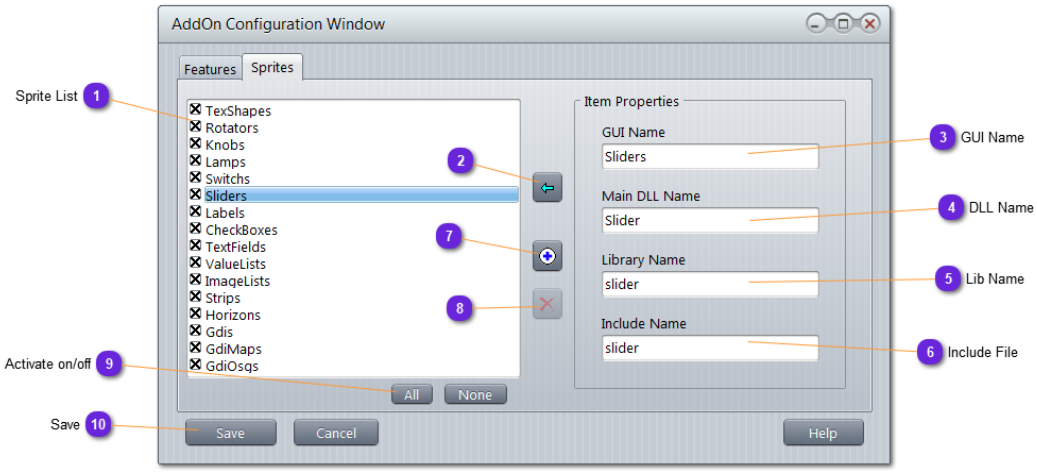
## • Hint

To move a Sprite at design, select it then use the Ctrl key while pressing down the mouse button and drag.
If the selected Sprite displays no hook, it is not resizable. Go to the property window and uncheck the Auto Size box.

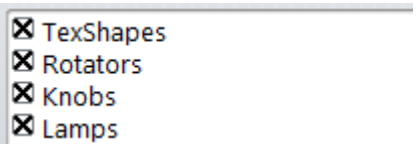
When hooks are displayed, they are used to resize the shape of the Sprite. The center hook is used for dragging.

# Settings

Sprites are defined outside vsTASKER using specific DLL.
They are structured into /Sprite directory. They can be included or excluded from the product environment.




## 1 Sprite List




Uncheck a Sprite to keep it in the file but forbid it to be loaded into vsTASKER.
To reduce the size of the vsTASKER memory footprint, uncheck the feature you will not use in your databases.

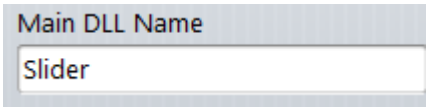
## 2 Update

Use this button to update the selected Sprite (from the left list) with the data from the Item Properties block.

## 3 GUI Name

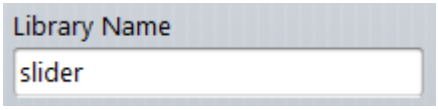

Specify here the **name** to be used in the menus, for the selected Sprite entry.

## 4 DLL Name



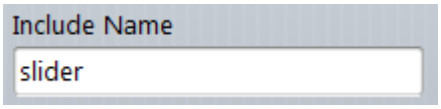

Name of the DLL in the /Sprite directory (needless to add the file extension). The GUI will load the <name>M.dll interface that handle all windows while the <name>.dll (compiled with VisualStudio) will be handling the data objects. The <name>X.dll will be used for drawing the feature onto the OpenGL device.

## 5 Lib Name




Name of the library that will be produced into /lib/vcxx.
This lib will automatically be added into the *makefile* produced by vsTASKER to compile a simulation engine produced with a database using the corresponding Sprite.

## 6 Include File




Name of the include file in /Sprite/Include

## 7 Add

Add a new Sprite (entry) into the list.

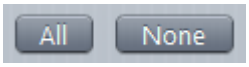

*This capability is only reserved for advanced users who are expanding the GUI using their own CodeGear C++ Builder IDE.*

## 8 Delete

Suppress the selected user added Sprite.

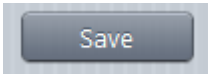
*This capability is only reserved for advanced users who are expanding the GUI using their own CodeGear C++ Builder IDE.*

## 9 Activate on/off




Use these buttons to tick/activate **all** or **none** of the Sprite Manager.
When a Manager is not ticked, it will not be available next time vsTASKER will be started. That means the associated DLL will not be loaded and the corresponding button in the toolbar will be disabled. When a database is loaded, if a Manager is disabled, all corresponding Sprites of the database will not be loaded. This can be useful when working with databases that do not have HMI. Starting vsTASKER will be faster as HMI DLL will not be loaded. Memory footprint will also be reduced.
Do not forget to activate some or all Managers if you intend to use the HMI panel.

## 10 Save

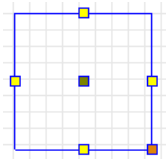


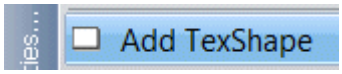

Fix the configuration on a file so that next time vsTASKER is loaded, only the selected Sprite DLLs will be loaded.

*This capability is only reserved for advanced users who are expanding the GUI using their own CodeGear C++ Builder IDE.*

# TexShape

Sprites are simple square shapes that can be either painted or textured.
Most of the sprites (input and output) are inheriting from the TexShape class.
Some sprite have two representations: a moving TexShape sprite and a background sprite.

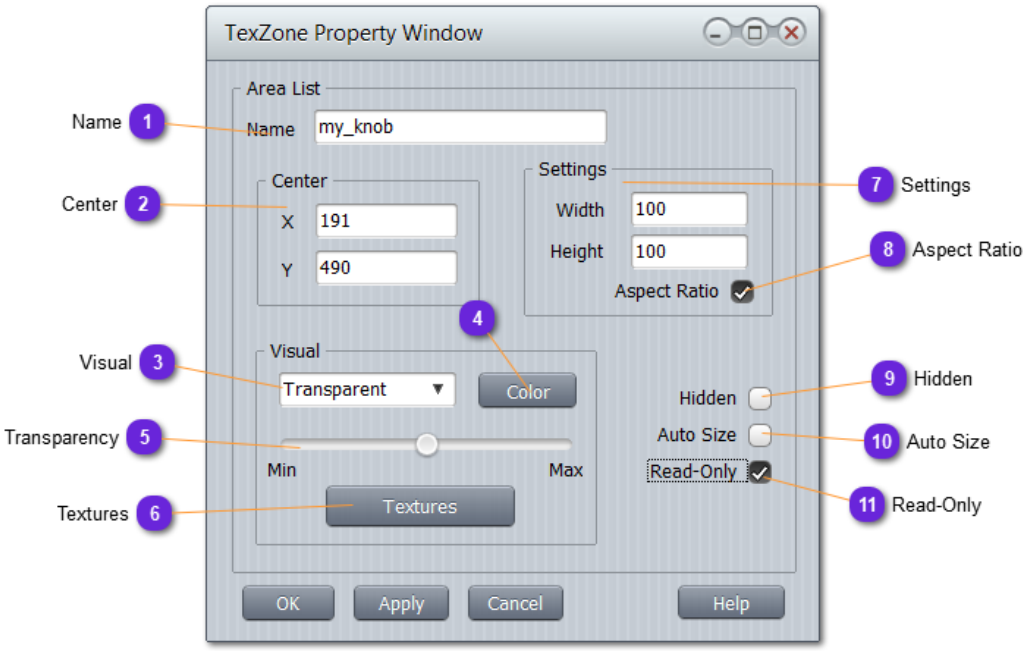


To add a TexShape, do 

or use 

Give it a name:

then **Create**.



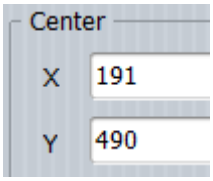
## 1 Name


Name of the TexShape. Must be unique whatever its type.
vsTASKER will generate a class whose name will be `<Name>_Spt`. For the moment, this class will inherit from `Vt_Sprite` class.
Any Sprite can be accessed using its name in a generated union named `s`.


*For ie, if a Sprite is named my_knob, the associated class will be `my_knob_Spt` and from the code, retrieving the instance object of the HMI will be done with `s.my_knob` or, in a more formal manner:*

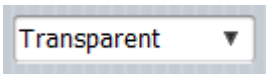
```
my_knob_Spt* my_knob = (my_knob_Spt*) R:findSprite("my_knob");
```

## 2 Center




Position on the HMI grid of the center of the shape.

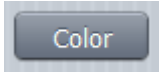
## 3 Visual




Select how the shape must be drawn:

- **Outline**: only the square shape will be drawn
- **Opaque**: the square shape will be filled with plain color
- **Transparent**: same as Opaque but the plain color will be transparent

## 4 Color

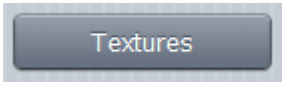



Color used for the outline or the bakcground color.

## 5 Transparency

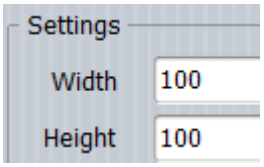

In case of a Transparent visual, use this slider to specify the amount of transparency to apply to the selected color, from **Min** (opaque) to **Max** (glass).

## 6 Textures




See here.

## 7 Settings

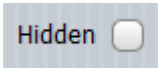



Width and Height of the sprite shape. If a texture has been loaded, will automatically be set with the dimension of the texture.

## 8 Aspect Ratio

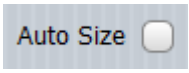

If checked, the Height (or Width) will automatically adapt to keep the ratio at the time the option is selected.

## 9 Hidden



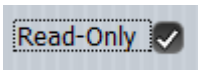

If checked, the sprite shape will not be visible (textured or not).

## 10 Auto Size



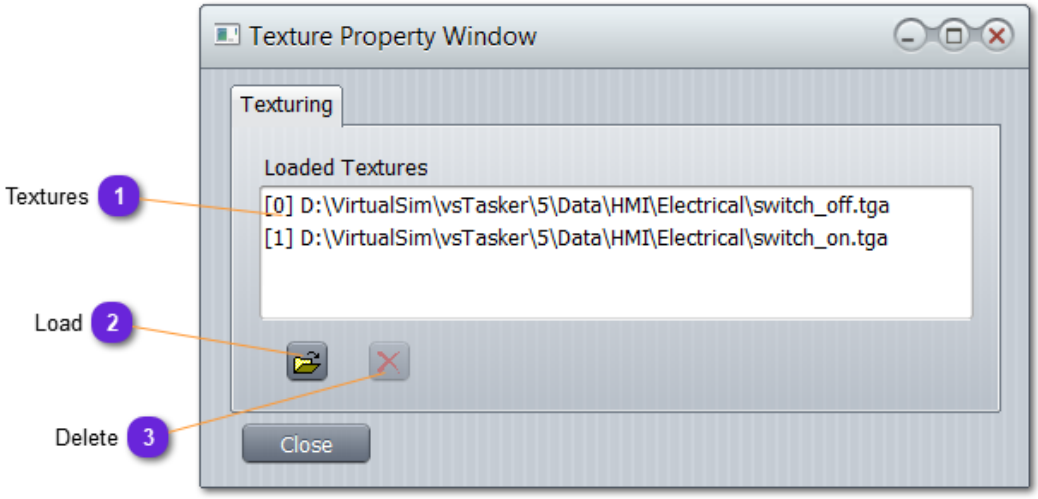

If checked, the size of the texture will be kept and won't be modifiable.
If no texture, (0,0) will be used.

## 11 Read-Only

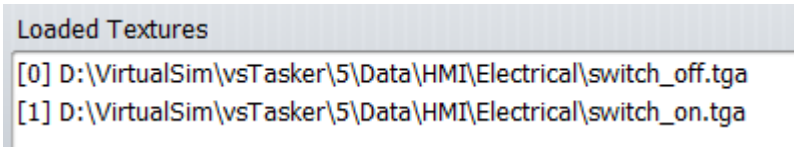



If checked, the sprite cannot be displaced or resized with the mouse.

# Texturing




## 1 Textures




List all loaded textures for the sprite.
The first one will be used by the object. The other ones might be used from the code to replace the first one.
All textures are indexed. Using the code `set(ID)`, the numbered texture will be used. Id ranges from **0** to **n-1**, **n** = number of textures loaded.
The texture ID is displayed in brackets.
In the example above, if the sprite name is my_knob, to set the switch_on.tga texture, just do `s.my_knob.set(1);`

## 2 Load

Add more sequential textures to the list.

## 3 Delete

Suppress the texture from the list (and not from the disk).
Select first the texture before using the button.

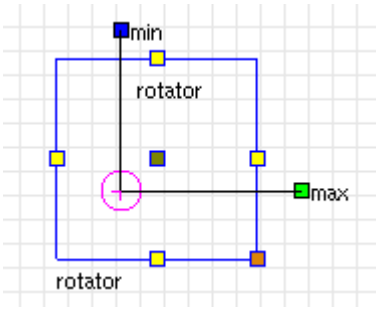
# Output

**Output** Sprites are used to display values or data to the user.

Although they are categorized in the **Simulation** ⇨ **User** directional way, some of these Sprite can also accept some input from the user (like the GDI with the mouse interaction).

# Rotator

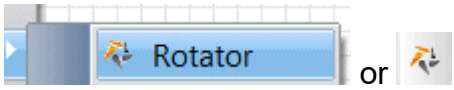
A Rotator is a textured bitmap that can rotate around a center, over a textured background that remains fixed.
It is typically used to represent dials (the needed being the rotating element)





## • How to Use

On the HMI grid, add a Rotator:

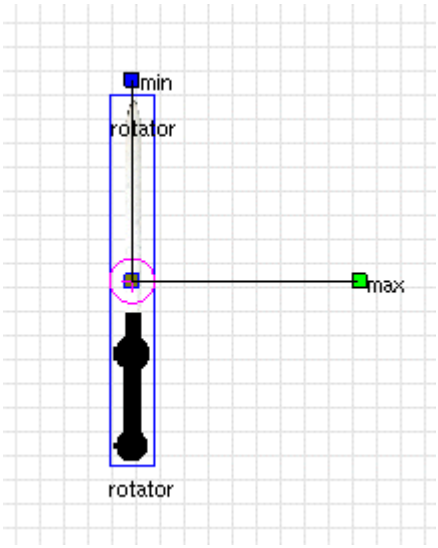


or


Give it a name, then open the property window.

Set the **minimum** and **maximum angles** (using the property window or by rotating the blue and green point of the Rotator graphic)
Set then the corresponding values for these minimum (0) and maximum (200). All values in between will output an angle between minimum and maximum.

Let's give to the rotator the air_needle.tga located in /Data/Hmi/Airspace
Select Properties, then Textures and load the file.




Select Background, then Textures and load the file air_panel.tga, then recenter the background or the needle to put the rotating center of the needle on the air_panel hole.
Also set the maximum angle to 342 degrees for a value of 200 (mph)


Now, time to connect the rotator to an entity speed.

In Initialization panel, `RESET`, put:

```
entity = S:findEntity("ac");
```

*`Vt_Entity* entity;` is already defined in `Vt_Base` and `Vt_Sprite` inherit from `Vt_Base`, so, needless to redefine it locally.*

In the Runtime panel, put the following code:

```
float spd = entity->getDyn()->getSpeed();
set(convMsToKnot(spd));
```

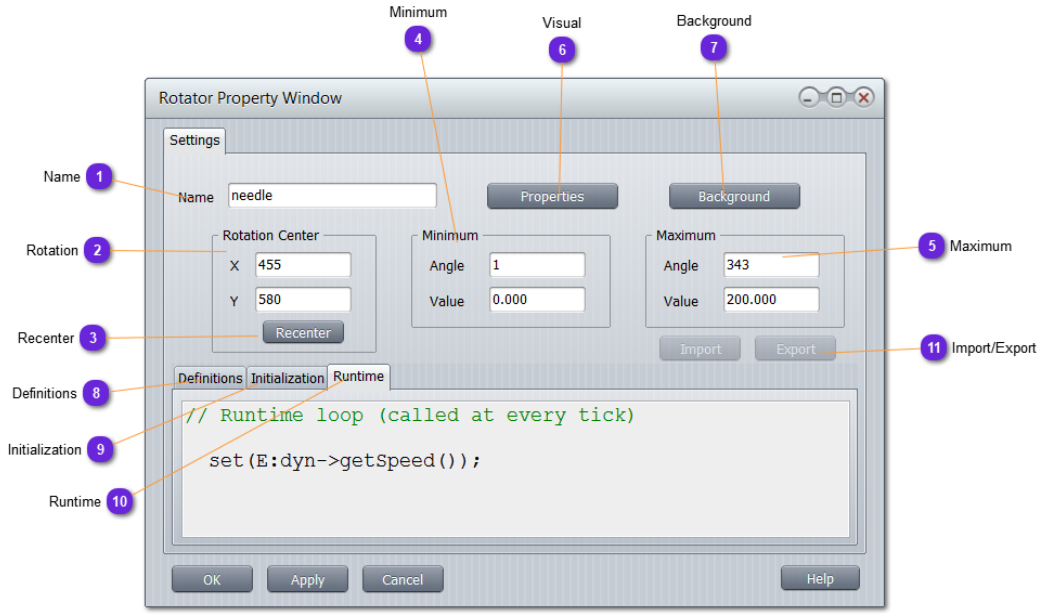
Back in the Terrain scenario, add one entity, name it "ac", compile run.
During the runtime, set the entity speed to "140" knots (using the hook window) and see the needle indicating the current entity speed.

- **Hint**

To move a Rotator, select it then use the Ctrl key while pressing down the mouse button and drag.
If the rotation center mark is on top of the center hook, the selection of the hook will not be possible.

# Properties

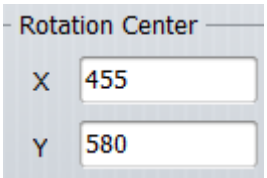
This property window setup the rotating shape only.
Most of the time, the shape is a needed.




## 1 Name


Name of the Rotator. Must be unique whatever its type.
vsTASKER will generate a class whose name will be `<Name>_Spt`. For the moment, this class will inherit from `Vt_Sprite` class.
Any Sprite can be accessed using its name in a generated union named `s`.

## 2 Rotation

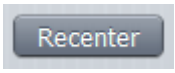



Coordinates, in the HMI design panel, of the sprite rotation center,

materialized on the design display with: 

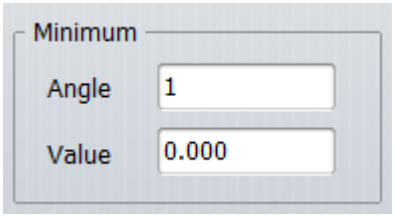
The values can be changed here for perfect precision, or using the mouse by just selecting the + cross and dragging it anywhere on the HMI.

## 3 Recenter




Put back the rotation center at the sprite center position. On some images, it works. Some have the rotation center shifted and must be manually setup.

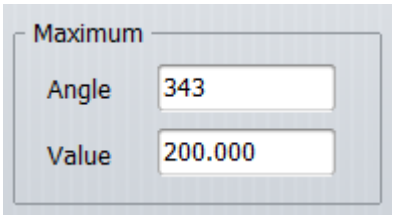
## 4 Minimum




Sprite actual minimum Angle in degrees for the given Value. 0 is up, -90 or 270 is left, 90 is right and -180 or 180 is down.
Minimum angle must always be **lesser** than Maximum.


*Even if the runtime value drops below the minimum Value above, the Angle will keep its Minimum.*

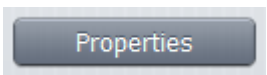
## 5 Maximum




Sprite actual maximum Angle in degrees for the given Value. 0 is up, -90 or 270 is left, 90 is right and -180 or 180 is down.
Maximum angle must always be **greater** than Minimum.

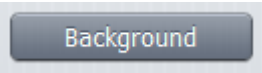

*Even if the runtime value goes above the maximum Value above, the Angle will keep its Maximum.*

## 6 Visual




This button opens the TexShape property window for texture/visual setting of the **rotating shape**.

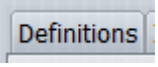
## 7 Background




**Deprecated !**

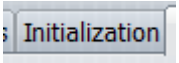
Only visible on database which have defined a background for Rotator sprites. It is advised to create a separate TexShape sprite to hold the background.
Create a TexShape with the same texture (the button is here to let you know which one is used), then delete the texture in this sprite. The button will later go.

## 8 Definitions



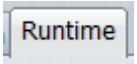

Put here all the local variables needed for this specific Sprite runtime computation.

## 9 Initialization




Initialize here (mostly in `RESET` part) all the variables defined in Definition section.
Default `set()` can also be used here for putting the Sprite in a specific state.

## 10 Runtime



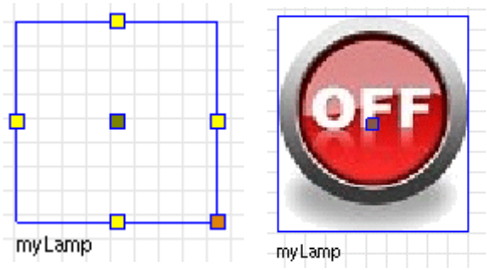

Holds the runtime code called at every cycle (HMI cycling rate).
Use `get()` to retrieve the actual value of the Sprite (the one that actually controls the texture rotation between Minimum and Maximum values)
Use `set()` to force the value of the Sprite (and control the corresponding rotation value of the texture).

## 11 Import/Export


*Not available yet.*

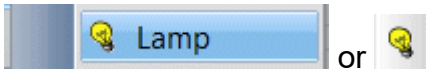
# Lamp

A Lamp is a textured or not Sprite that can change color or texture according to various states.
The states are user controlled from the code.




## • Using Textures

On the HMI grid, add a Lamp:

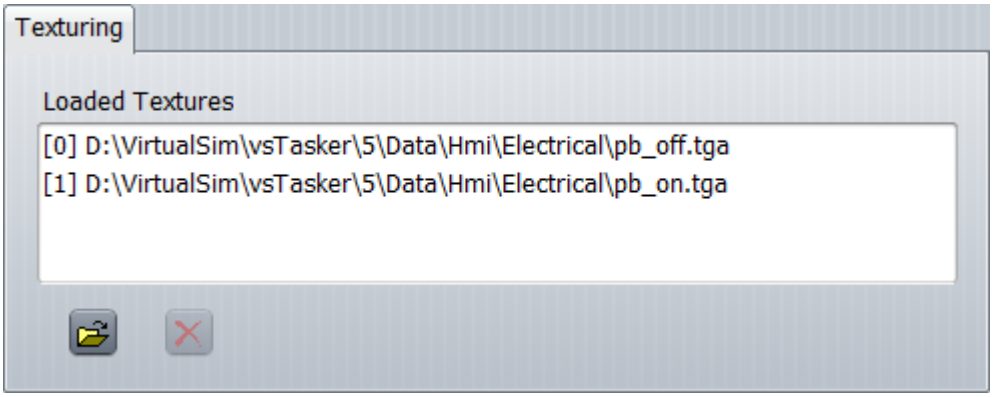


or 


Give it a name, then open the property window.

Select the Visual Aspect button and then Textures

Let's give to our lamp the pb_off.tga and pb_on.tga textures located in /Data/Hmi/Electrical




Now, time to connect the rotator to an entity speed.

### Lamp

In Initialization panel, `RESET`, put:

```
entity = S:findEntity("ac");
```

*`Vt_Entity* entity;` is already defined in `Vt_Base` and `Vt_Sprite` inherit from `Vt_Base`, so, needless to redefine it locally.*

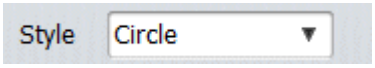
In the Runtime panel, put the following code:

```
 if (entity->getSpeed()>100) set(1);  // select texture id 1,
pb_on.tga
 else set(0);                         // select texture id 0,
pb_off.tga
```

Back in the Terrain scenario, add one entity, name it "ac", compile run.
During the runtime, set the entity speed to "140" m/s (using the hook window) and see the lamp changing from Off to On.

## • Using Painting

In this example, we will not use texture but we will paint our lamp from the code.
To process, to the above but do not load any texture.
Change the shape from Rectangle to Circle.



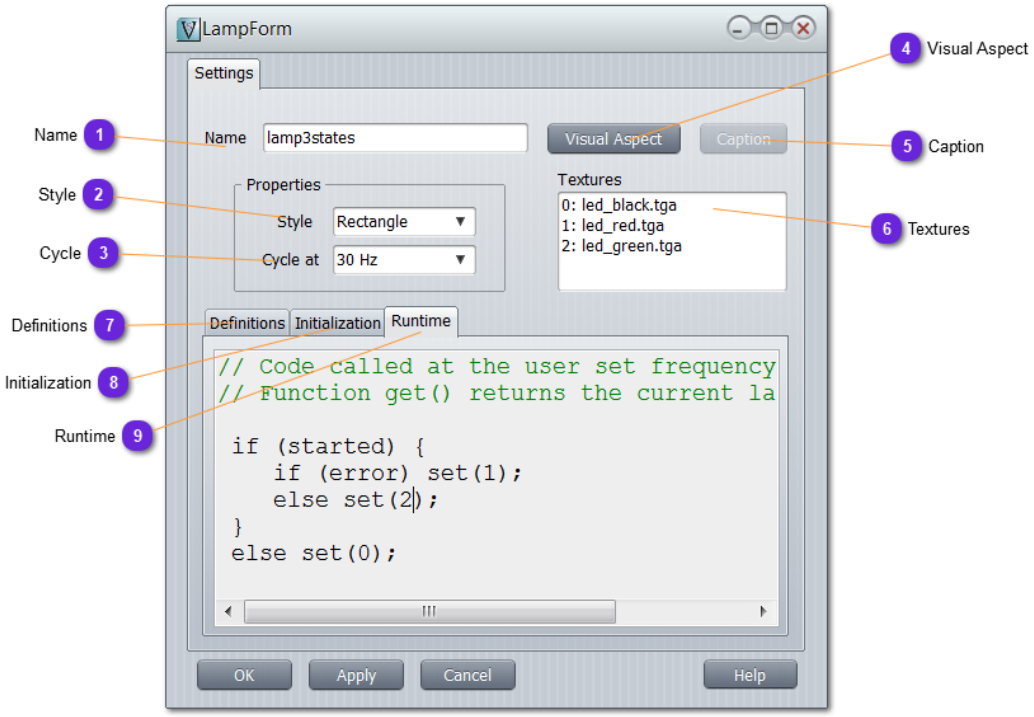

In the Runtime panel, put the following code:

```
 if (entity->getSpeed()>10) paint(clBlue);
 else paint(clRed);
```

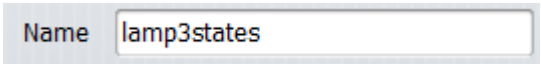
And at runtime, you will get the following:

 changing to  when entity speed get greater than 10 m/s.

# Properties

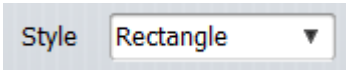



## 1 Name




Name of the Lamp. Must be unique whatever its type.
vsTASKER will generate a class whose name will be `<Name>_Spt`. For the moment, this class will inherit from `Vt_Sprite` class.
Any Sprite can be accessed using its name in a generated union named `s`.

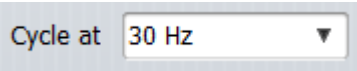
## 2 Style




Define the type of the drawing shape.:

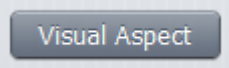
- 
- Circular: round shape for painted lamps

## 3 Cycle



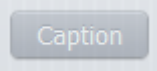

Select here the frequency call for the runtime code (see below).

## 4 Visual Aspect



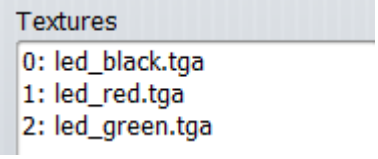

Call the Visual Aspect definition window.

## 5 Caption




Call the (optional) Label definition window to format the text that can appear below the Switch.

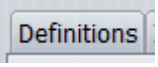
## 6 Textures




List all loaded textures that will be displayed upon a `set()` command.
The first number (**id**) is the one to be given to the command (at runtime).
For i.e, on the above list, to select the led_green texture, put in the code:

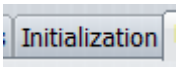
```
set(2);
```

## 7 Definitions



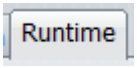

Put here all the local variables needed for this specific Sprite runtime computation.

## 8 Initialization




Initialize here (mostly in `RESET` part) all the variables defined in Definition section.
Default `set()` can also be used here for putting the Sprite in a specific state.

## 9 Runtime



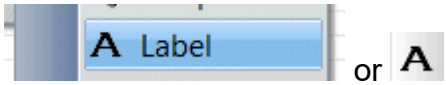

Holds the runtime code called at every cycle (HMI cycling rate).
Use `get()` to retrieve the actual value of the Sprite (the one that actually select the texture or memorize the current state).
Use `set()` to force the value of the Sprite (and select the corresponding texture to be displayed).

# Label

A Label is formatted text string that can be put anywhere on the HMI.
Some Sprites already embed a Label.


## • How to Use

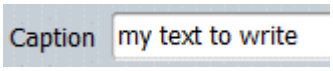
On the HMI grid, add a Label:



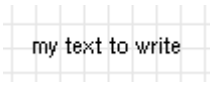
or 


Give it a name, then open the property window.

Put the text to display in the Caption field




to get



# Properties


## 1 Name

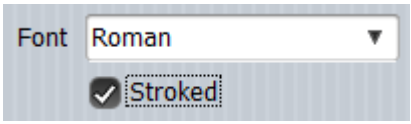

Name of the Label. Must be unique whatever its type.
vsTASKER will generate a class whose name will be `<Name>_Spt`. For the moment, this class will inherit from `Vt_Sprite` class.
Any Sprite can be accessed using its name in a generated union named `s`.

## 2 Caption


Put here the text that will be displayed on the HMI.
This can be modified from the code of another Sprite or from any Component or Logic of the simulation with a simple call:

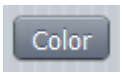
```
s.<Name>_Spt->db->setCaption("my new text");
```

## 3 Font



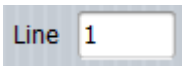

Select the available font.
Default OpenGL fonts use **bitmaps** and vsTASKER uses `glutBitmapCharacter()` to render them. These fonts cannot be scaled so, have to be selected carefully.
If **Stocked** options is checked, vsTASKER uses `glutStrokeCharacter()` routine that renders a single stroked character from a specified GLUT stroke font (Roman and Mono Roman)

## 4 Color




Select here the color of the font to be used on the HMI.

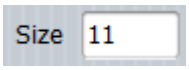
## 5 Line




Width of the line drawing each character.

*Only for Stroked fonts.*

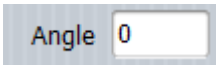
## 6 Size



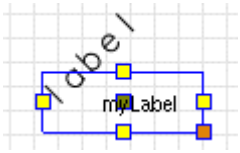

Set the size of the character, in pixels.

*Only for Stroked fonts.*

## 7 Angle



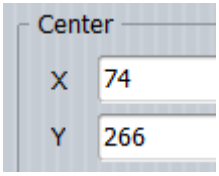

Set the angle of the character, in degrees.
Example with 45° value:





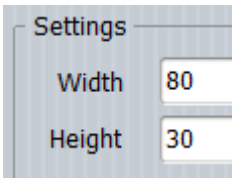
*Only for Stroked fonts.*

## 8 Center



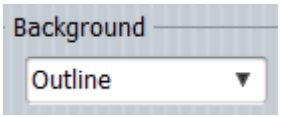

Coordinates on the HMI panel of the center hook.
The values can be changed here for perfect precision, or using the mouse by just selecting the hook and dragging it anywhere on the HMI.

## 9 Size




Actual size of the Label Sprite.
Can matters if the background if filled or outlined.

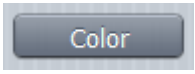
## 10 Background




Set the background style:

- **Outlined**: use only one line of size 1 of the selected color.
- **Opaque**: fill the Sprite rectangle with the selected color.
- **Transparent**: same as opaque but using a transparency mask on the selected color.

## 11 Color




Select the color to be used for the background shape.

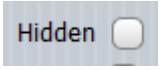
## 12 Transparency


When Transparent background is used, drag the slider to select the level of transparency from Min (opaque) to Max (pure glass).

## 13 Import/Export

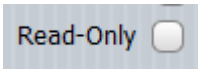

*Not available yet.*

## 14 Hidden




If checked, the label will not be displayed.
Useful at runtime to display a text on the HMI or hide it.

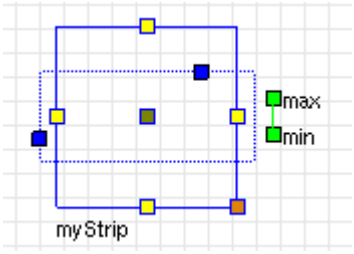
## 15 Read-Only

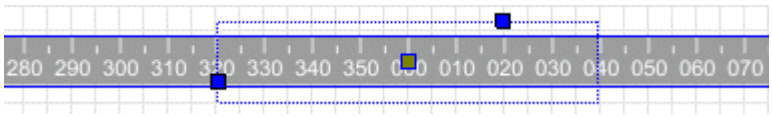



When checked, the Sprite cannot be dragged.

# Strip

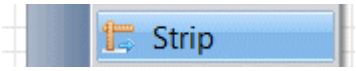
A strip is used to slide a texture behind a squared window.
The sliding can be either vertical or horizontal.




## • How to Use

On the HMI grid, add a Strip:



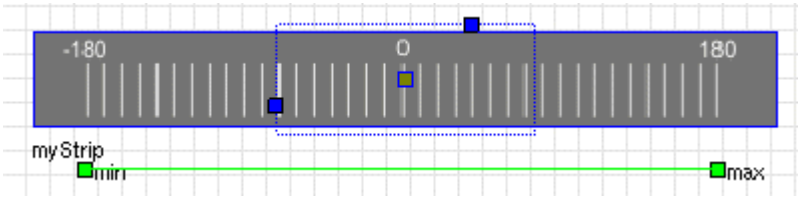
or 


Give it a name, then open the property window.

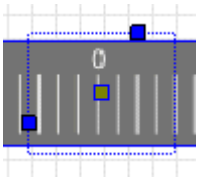
Make the **Way** Horizontal.

Then select **Visual Aspect** to import the following texture: `azimuth.gif` in `/Data/HMI/Aerospace`

On the HMI panel, drag the Min and Max green hooks so that to visually align them with the -180 and 180 values of the strip:




Then use the blue hooks to resize the peek window of the Strip (everything outside will be scissor).




Open again the Property window and set the value bounds:


By giving 180 to Max, for the runtime Value of 180, the texture (at the position of the green Max hook) will be centered in the middle of the peek window.
Same thing with Min hook for the Value of -180.

In Initialization panel, `RESET`, put:

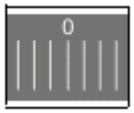
```
entity = S:findEntity("ac");
```


*`Vt_Entity* entity;` is already defined in `Vt_Base` and `Vt_Sprite` inherit from `Vt_Base`, so, needless to redefine it locally.*

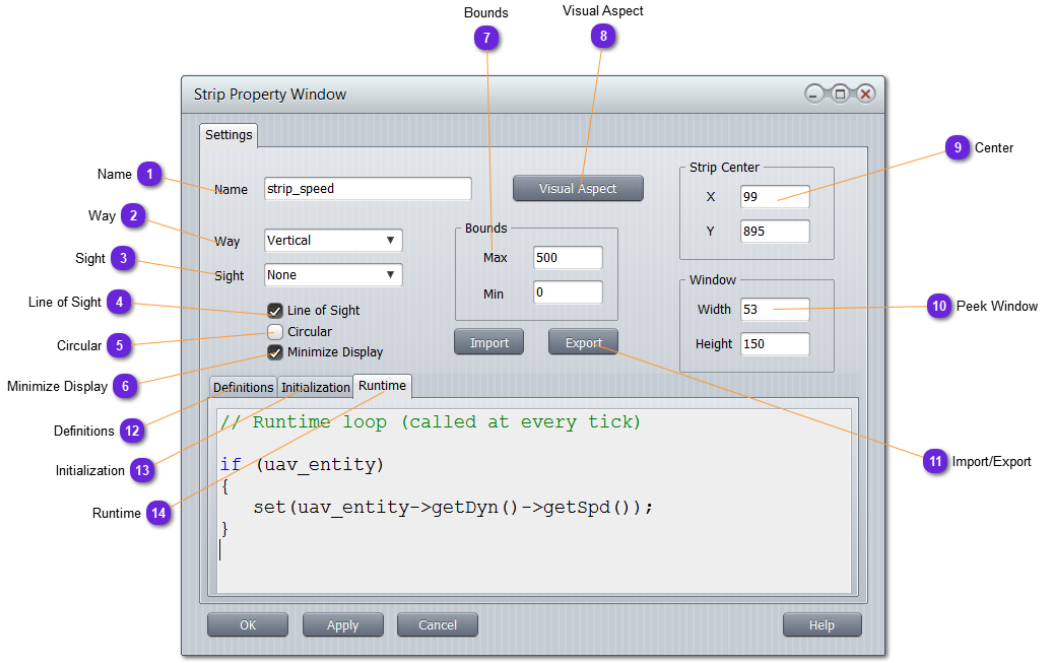
In the Runtime panel, put the following code:

```
set(convRadToDeg(entity->getDyn()->getHdg()));
```

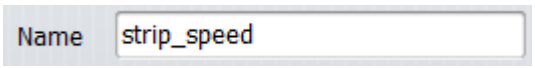
Back in the Terrain scenario, add one entity, name it "ac", compile run.
Now, change the heading of the aircraft from the hook window and see the Strip in action on the HMI window.



# Properties

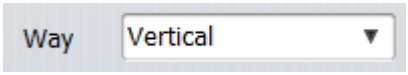



## 1 Name




Name of the Strip. Must be unique whatever its type.
vsTASKER will generate a class whose name will be `<Name>_Spt`. For the moment, this class will inherit from `Vt_Sprite` class.
Any Sprite can be accessed using its name in a generated union named `s`.

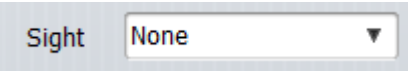
## 2 Way




Select here the motion that will be applied on the texture:

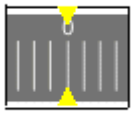
- **Vertical**: strip moving up and down
- **Horizontal**: strip moving left and right.

## 3 Sight



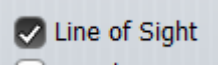

Select here if nicks must be used in the middle of the peek window:

- **None**: not displayed
- **Left/Up**: according to the Way, only on one side
- **Right/Down**: according to the Way, on the other side
- **Both**: both side (see picture below)

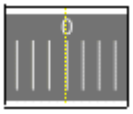


*Color used for the nicks is yellow by default.*

## 4 Line of Sight

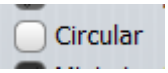



If checked, the peek window will display in its middle a vertical or horizontal dashed line, as in the picture below:



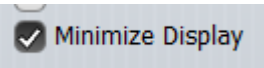
*Color used for the line is yellow by default.*

## 5 Circular



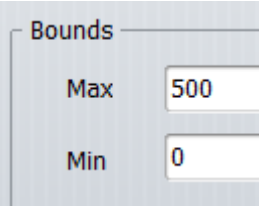

When checked, the value given using `set()` will be modulo [Min..Max]. For i.e, `set(181)` with bounds defined as [-180,180] will give value -179.
If unchecked, the value `set()` will be bound by [Min..Max]. For i.e, `set(180)` with bounds defined as [-180,180] will give value 180.

## 6 Minimize Display




When checked, the strip (texture) is not displayed on the HMI at design. Only the viewing window is displayed.
To display the strip, select it first.
If unchecked, strip (texture) is always displayed.

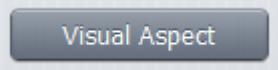
## 7 Bounds




Set here the values that correspond to the visual hooks of the strip.

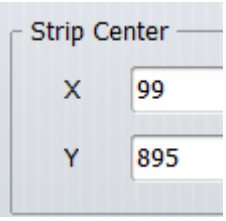

The strip value is set using the `set()` function in the runtime code.
Unless **Circular** is checked, the value is always bounded into [Min..Max]

## 8 Visual Aspect



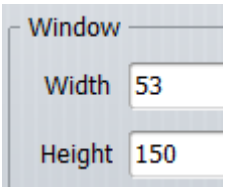

Call the Visual Aspect definition window.

## 9 Center




Coordinates, in the HMI design panel, of the peek window center, materialized on the design display with: 
The values can be changed here for perfect precision, or using the mouse by just selecting the hook and dragging it anywhere on the HMI.

## 10 Peek Window

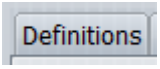



Actual size in pixels of the peek Window.
The values can be changed here for perfect precision, or using the mouse by just selecting the vertical or horizontal hook and dragging it to resize.

## 11 Import/Export

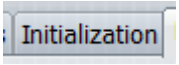

*Not available yet.*

## 12 Definitions



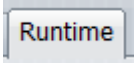

Put here all the local variables needed for this specific Sprite runtime computation.

## 13 Initialization




Initialize here (mostly in `RESET` part) all the variables defined in Definition section.
Default `set()` can also be used here for putting the Sprite in a specific state.

## 14 Runtime

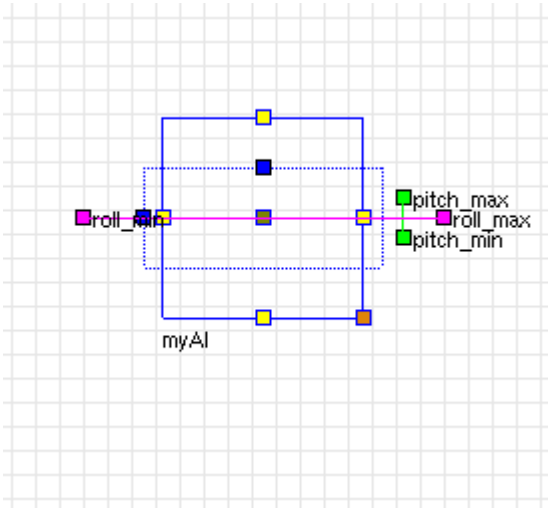


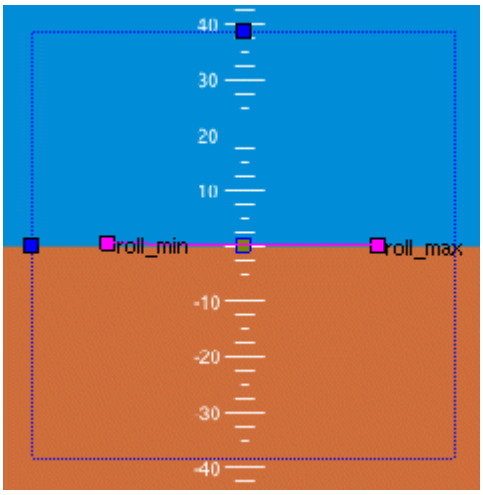

Holds the runtime code called at every cycle (HMI cycling rate).
Use `get()` to retrieve the actual value of the Sprite (the one which actually controls the strip position between Minimum and Maximum values)
Use `set()` to force the value of the Sprite (and control the corresponding position of the strip).

## Properties

# Horizon

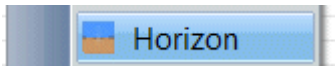
An Horizon is a combination of rotating and sliding a texture, behind a squared window.
The Horizon is mainly used to represent an Attitude Indicator (AI) normally present into any PFD (primary flight display) for aircraft.




## • How to Use

On the HMI grid, add an Horizon:



or 


Give it a name, then open the property window.

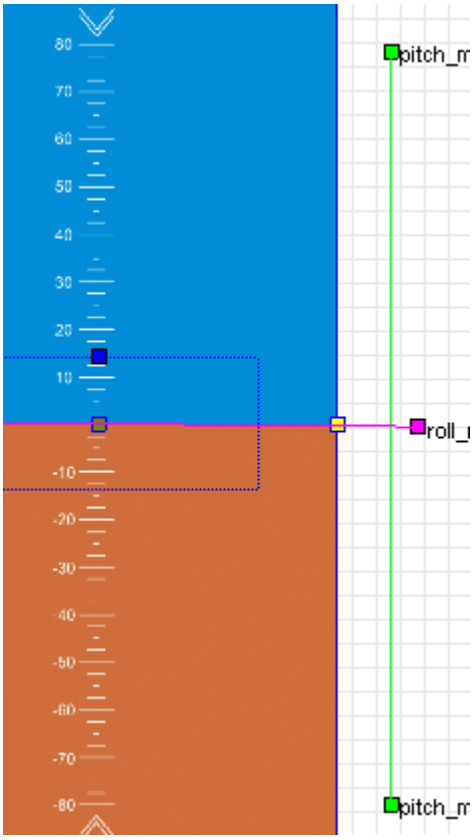
Make the **Way** Horizontal.

Then select **Strip Definition**, then **Visual Aspect** and **Textures** to import the following one: `horizon.png` in `/Data/HMI/Aerospace`
Uncheck **AutoSize**.

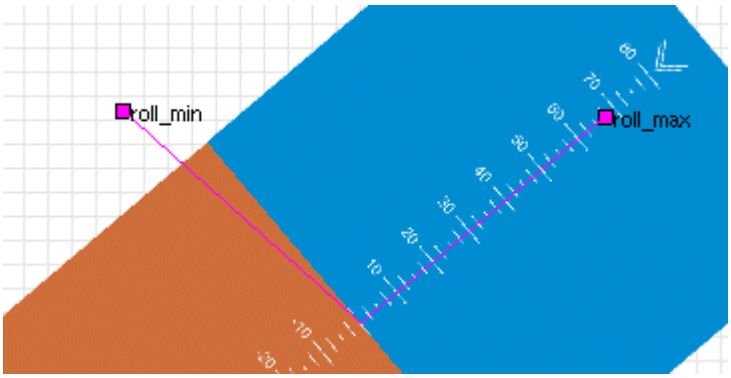
Resize the texture using the  hook

On the HMI panel, drag the Pitch-Min and Pitch-Max green hooks so that to visually align them with the -80 and 80 values of the horizon vertical scale.
On the code, the vertical (pitch) value will be given using `setPitch()` function.

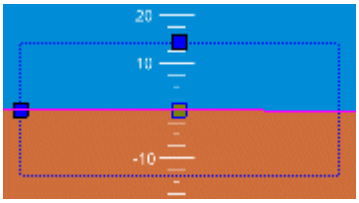
### Horizon



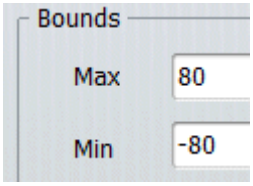

Rotate the magenta hooks to set the maximum angles of the texture.
On the code, the bank (roll) value will be given using `setRoll()` function.




Then use the blue hooks to resize the peek window of the Horizon (everything outside will be scissor).



Open again the Property window, select **Strip Definition** and set the bound values for the pitch:




On the Horizon window, make sure the visual rotating angles (Min, Max) correspond to their matching Values.


Now, let's add some code:

In Initialization panel, `RESET`, put:

```
entity = S:findEntity("ac");
```

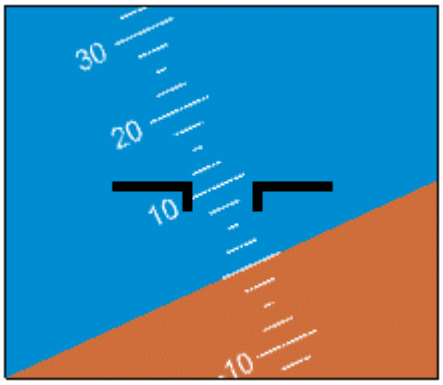
*`Vt_Entity* entity;` is already defined in `Vt_Base` and `Vt_Sprite` inherit from `Vt_Base`, so, needless to redefine it locally.*

In the Runtime panel, put the following code:

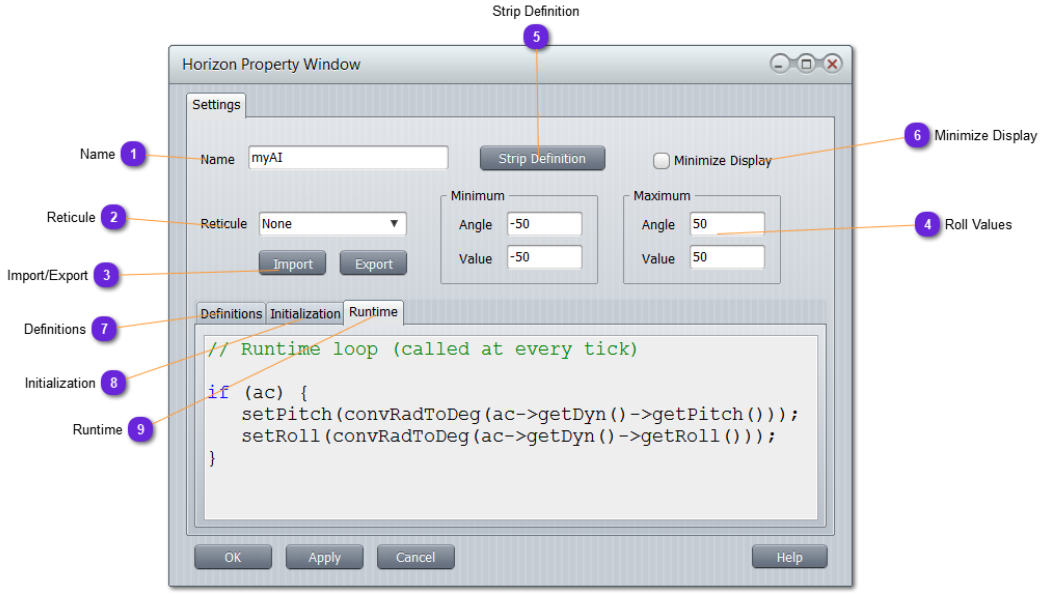
```
setPitch(convRadToDeg(entity->getDyn()->getPitch()));
setRoll(convRadToDeg(entity->getDyn()->getRoll()));
```

Back in the Terrain scenario, add one entity, name it "ac", compile run.
Give the entity a WingDyn dynamic. Set initial speed and altitude.
Now, change the heading of the aircraft from the hook window and see the Horizon in action on the HMI window.

## Horizon



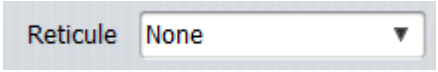
# Properties




## 1 Name


Name of the Horizon. Must be unique whatever its type.
vsTASKER will generate a class whose name will be `<Name>_Spt`. For the moment, this class will inherit from `Vt_Sprite` class.
Any Sprite can be accessed using its name in a generated union named `s`.

## 2 Reticule



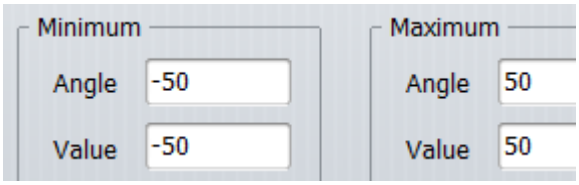

Specify how the reticule in the center of the peek window shall be drawn:

- None: no reticule
- Cross: a simple cross in the middle of the peek window: 
- Simple: draw this black shape: 

## 3 Import/Export

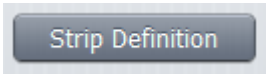

*Not available yet.*

## 4 Roll Values



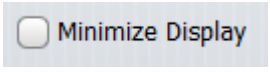

Specify here the Minimum and Maximum pairs of (Angle,Value) for the texture to rotate according to the roll/bank angle.
Angle unit is degrees.
Value can be anything as it is a float number set by the used with `setRoll()` function.

## 5 Strip Definition



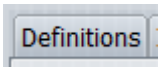

Set here the Minimum and Maximum pairs of (Angle,Value) for the texture to slide according to the pitch/elevation angle.
Angle unit is degrees.
Value can be anything as it is a float number set by the used with `setPitch()` function.
See the property window.

## 6 Minimize Display



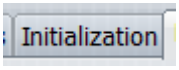

When checked, the horizon (texture) is not displayed on the HMI at design. Only the viewing window is displayed.
To display the texture, select it first.
If unchecked, horizon (texture) is always displayed.

## 7 Definitions



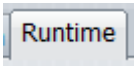

Put here all the local variables needed for this specific Sprite runtime computation.

## 8 Initialization




Initialize here (mostly in `RESET` part) all the variables defined in Definition section.
Default `set()` can also be used here for putting the Sprite in a specific state.

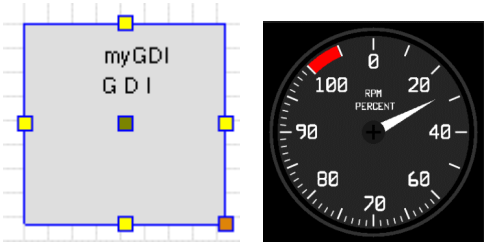
## 9 Runtime



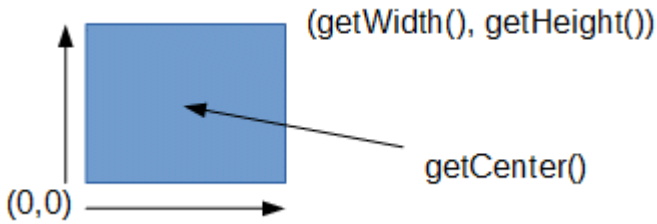

Holds the runtime code called at every cycle (HMI cycling rate).
Use `getPitch()` or `getRoll()` to retrieve the actual values of the Sprite (the ones which actually control its roll and pitch of the horizon between Minimum and Maximum values)
Use `setPitch()` and `setPitch()` to force the value of the Sprite (and control the corresponding roll and pitch values of the horizon).

# GDI

The GDI Sprite is a rectangular display that can display any OpenGL user code. It can be used to represent any specific gauge or graphic that is not supported by the actual given Sprites.




## • Good to Know




`getWidth()`: return the width of the GDI (horizontal), same as `db->width`
`getHeight()`: return the height of the GDI (vertical), same as `db->height`
`getCenter()`: return a `Coord2D` for the center of the GDI

## • How to Use

On the HMI grid, add a GDI:

or 

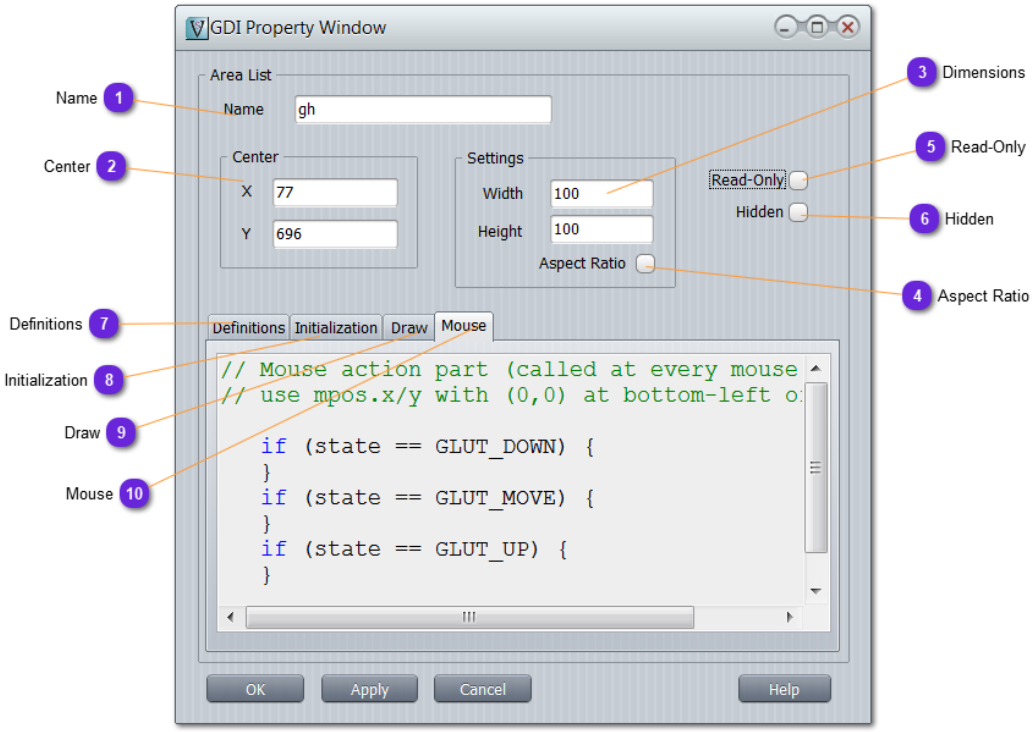

Give it a name, then open the property window.

In the Runtime panel, put the following code:

```
ogl->drawCircle(db->width/2,db->height/2,10);
```

This will draw a circle of 10 pixels centered in the middle of the GDI

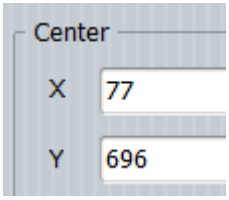
# Properties




## 1 Name

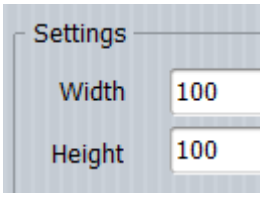

Name of the Gdi. Must be unique whatever its type.
vsTASKER will generate a class whose name will be `<Name>_Spt`. For the moment, this class will inherit from `Vt_Sprite` class.
Any Sprite can be accessed using its name in a generated union named `s`.

## 2 Center



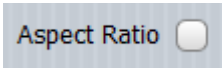

Coordinates, in the HMI design panel, of the GDI center position, materialized on the design display with: ■
The values can be changed here for perfect precision, or using the mouse by just selecting the hook and dragging it anywhere on the HMI.

## 3 Dimensions



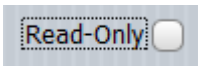

Specify the width and the height of the GDI as they will be kept in the HMI.

## 4 Aspect Ratio



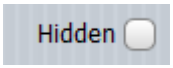

When checked, the resizing will keep the ratio memorized at the time the option was checked.
If unchecked, width and height can change freely from the hooks or from the Dimensions text fields.

## 5 Read-Only



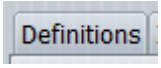

If checked, the GDI cannot be modified.

## 6 Hidden



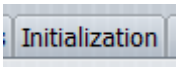

If checked, the GDI will not be displayed on the runtime HMI.
This value can be changed using code to make the GDI appear and disappear.

## 7 Definitions



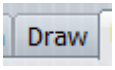

Put here all the local variables needed for this specific Sprite runtime computation.

## 8 Initialization



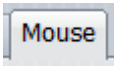

Initialize here (mostly in `RESET` part) all the variables defined in Definition section.

## 9 Draw




Holds the runtime code called at every cycle (HMI cycling rate) that will display the user defined graphic.
Put any OpenGL code here.

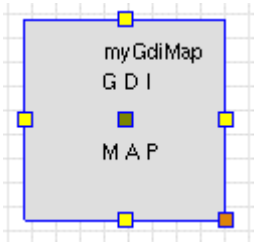
## 10 Mouse

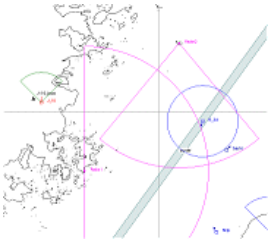



This part is called for any mouse selection on the GDI.
The mouse **button** is defined by the `state`.
The mouse **position** inside the GDI (from 0, 0 to width, height) is hold in `mpos` variable.

# GDI Map

The GDI Map Sprite mimic the vsTASKER 2D terrain display on the HMI.
It will load the libraries defined in plugins SIM, the same way as the OpenGL viewer does.





*Only one GDI Map can be instanciated on one HMI.*

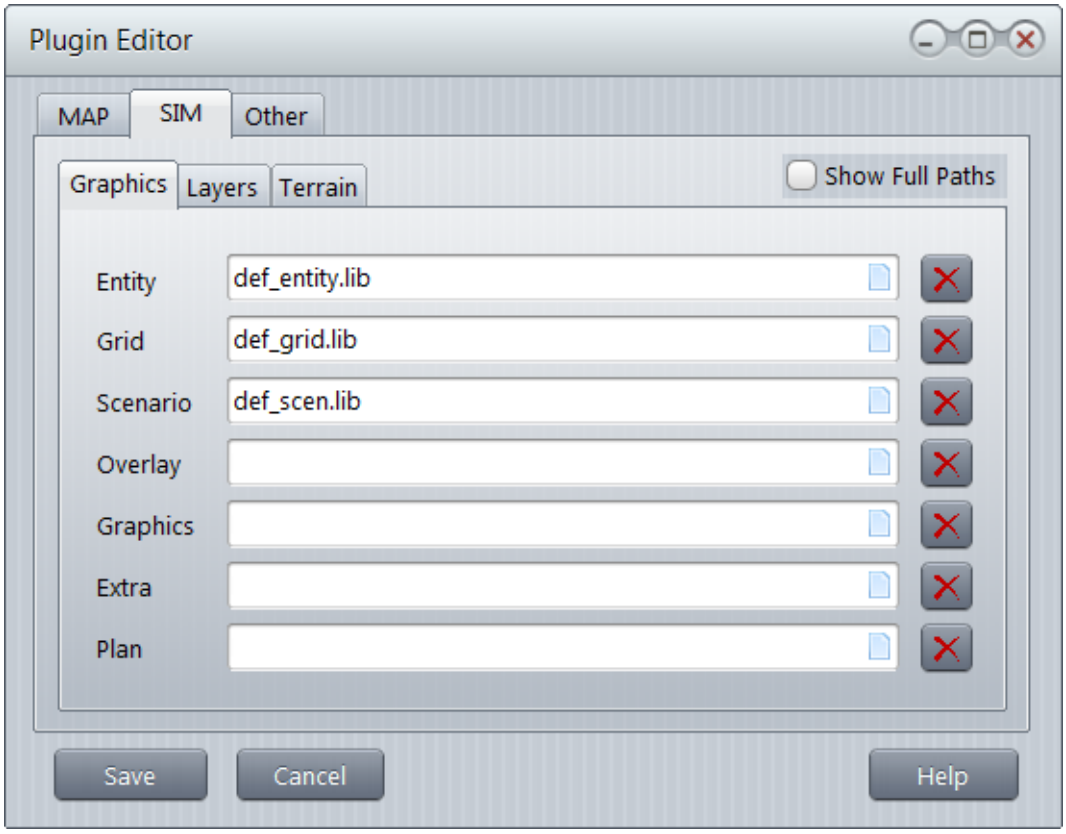
## • How to Use

On the HMI grid, add a GDI Map:

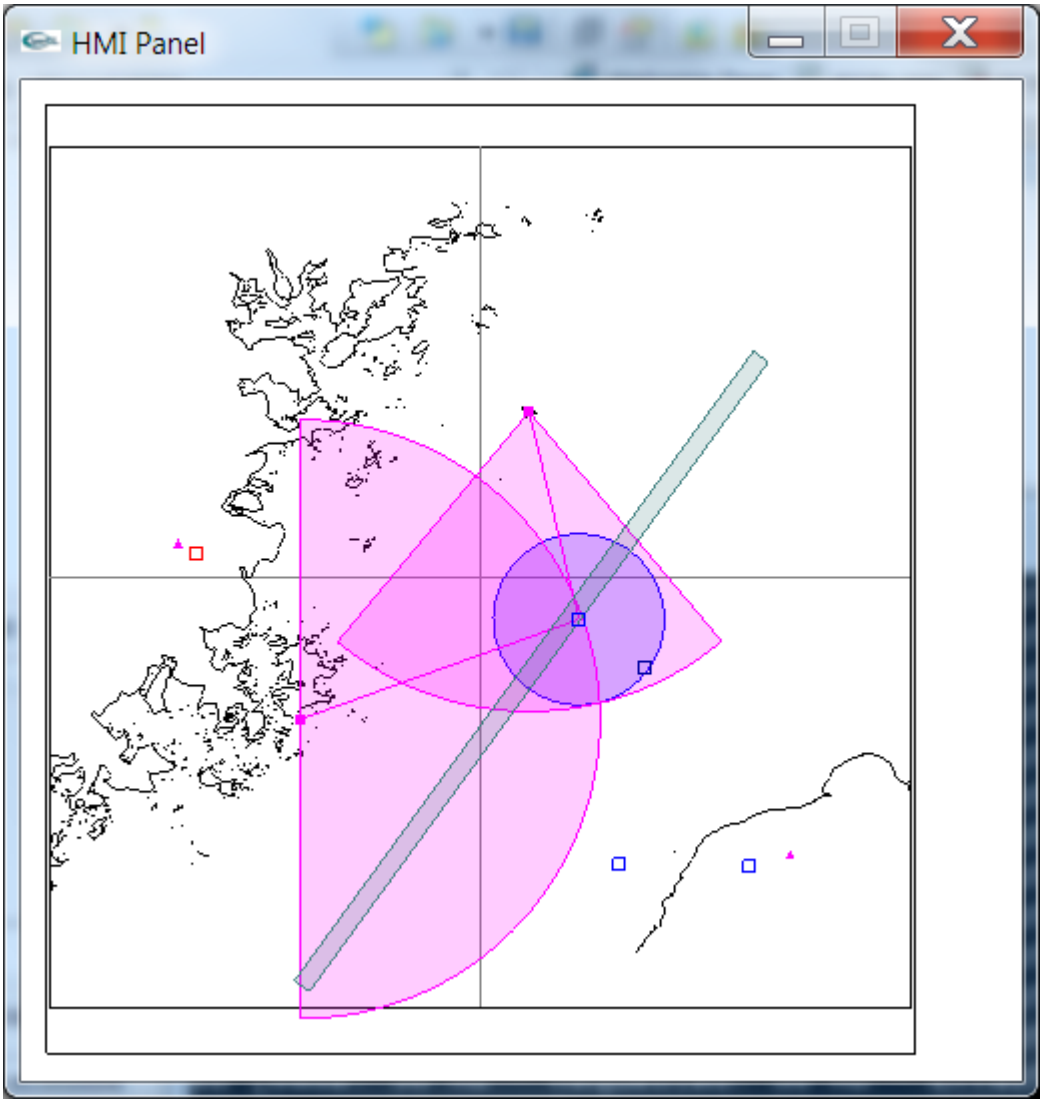
or 


Give it a name, resize it so that it get a good size of the HMI.
Then, open the Database Plugin Editor window and check that for the SIM, the requested libraries are set.
If some are missing (like Scenario for example), add them (can be found in /lib/vc<your environment>/def...)

### GDI Map




Compile and run.
(if you just do that on the existing scenario, /samples/console/elecwfare, you would get the following):




## • Entity Labels

Like for the GUI map, entity labels can be repositioned on the GdiMap using the mouse (select, hold and drag). The selection area is predefined and defaulted to 100 pixels width and 50 pixels height.

To change theses values, you must individually (for each entity) specify the new width and height value. It can be computed in a task or wherever part of the code you like, according to the characters length displayed (moreover if you replaced the default dll/lib)

**Method 1:** (for version 6.0.23 and above)

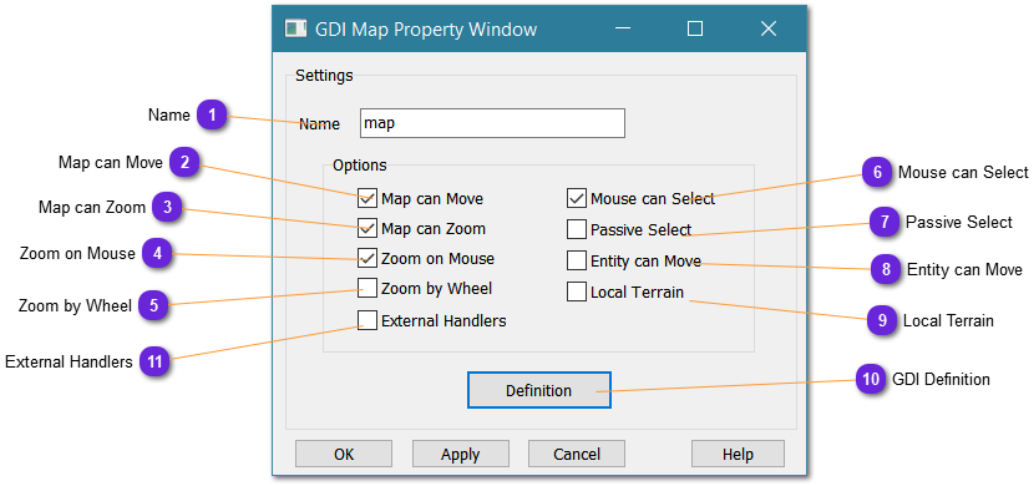
### GDI Map

```
Vt_Entity* my_ent = S:findEntity("my_entity");
my_ent->storage.set(ENT_LABEL_WIDTH, 200); // defined in vt_hmi.h
```

**Method 2:** (for version 7 and above)

```
Vt_Entity* my_ent = S:findEntity("my_entity");
my_ent->db->label.width = 200;
```

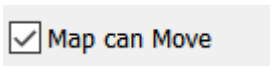
# Properties




## 1 Name

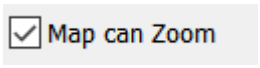

Name of the GdiMap. Must be unique whatever its type.
vsTASKER will generate a class whose name will be `<Name>_Spt`. For the moment, this class will inherit from `Vt_Sprite` class.
Any Sprite can be accessed using its name in a generated union named `s`.

## 2 Map can Move



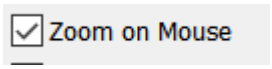

When **enabled**, the map can be moved using the **left mouse button** + **CTRL** key.
If **disabled**, the map is fixed.

## 3 Map can Zoom




When **enabled**, the map can be zoomed using the **mouse wheel** or the **left mouse button** (see Zoom by Wheel).
If **disabled**, the map is maximized.

## 4 Zoom on Mouse

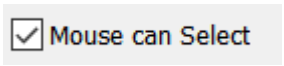



When **checked**, the map zoom will **focus** on the **mouse position**.
If **unchecked**, the zoom **focus** on the **map center**.

## 5 Zoom by Wheel


If **enabled**, the **mouse wheel** is used to control the zoom.
If **disabled**, the zoom is done using the **mouse motion** by combining the **SHIFT** key and **left mouse button**.

## 6 Mouse can Select

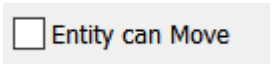



When checked, entities can be selected by the mouse.
If unchecked, nothing can be selected.

## 7 Passive Select

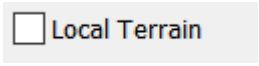

When this option is enabled, the mouse displacement on the map will provide continuous `mouseMove` function call with the position of the mouse. This can be used to highlight some information when mouse move over without selection.

## 8 Entity can Move



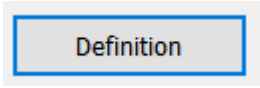

When checked, any selected entity (from the mouse or any other way) can be draged on the map with the left mouse button down.

## 9 Local Terrain



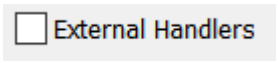

When checked, the terrain used by the GDI will be the current scenario terrain.
If unchecked, the terrain will not be defined.

## 10 GDI Definition




Use this button to setup the GDI parent layer.

## 11 External Handlers

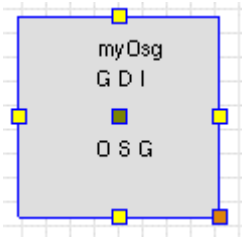



Select this options when user needs to define himself some mouse handler functions normally hidden inside the generated code. This can be the case when the simulation engine is embedded into another application (like Qt) and need to be informed outside the GDI Map object.
Here is the list of the available handlers:

- `void focusChange()`: called each time a new entity is selected in the GDI Map (or when deselected). Use `vt_rtc->getFocus()` to know which one is selected (NULL if none).

# GDI OSG

The GDI OSG Sprite allows an OpenSceneGraph window to be displayed on the HMI.
It will use a 3D terrain and will add all options defined in a (optional) OSG Viewer (than must be disabled).



*Only one GDI OSG can be instanciated on one HMI.*
*When an HMI is using an OSG output, the OSG Viewer cannot be enabled. It must be disabled and setup for the HMI OSG only.*

## • How to Use

On the HMI grid, add a GDI OSG:

or 

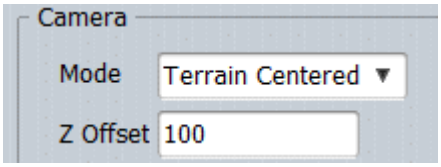

Give it a name, resize it so that it get a good size of the HMI.

Create a scenario.
Open the Terrain::Raster Maps:

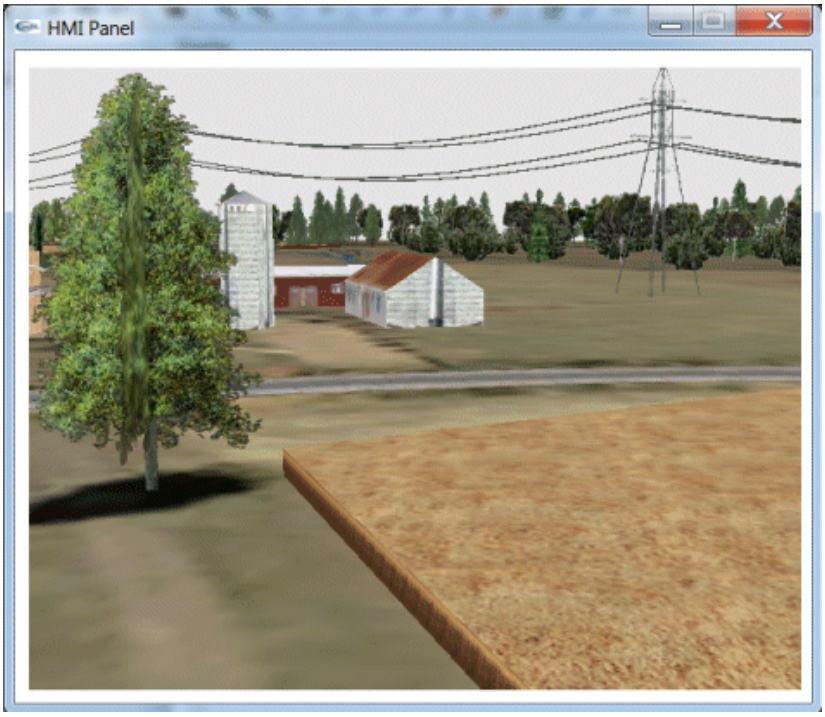

and load the following synthetic map:
D:\VirtualSim\vsTasker\7\Data\Map\Synthetic\field.map

Now, go back to the HMI.
Open the GDI OSG Sprite and in its property window, load the corresponding terrain database in File:
D:/VirtualSim/vsTasker/7/Runtime/OSG/Terrains/field/master.ive

set **Camera Mode** to Terrain Centered and the **Z Offset** to 100 meters.




Compile and run.
The HMI window will display the OSG terrain in 3D.




## • Fine Tuning

The GdiOsg Sprite can use the setting of an OSG Viewer if one is defined. Otherwise, it will create one with default values.
The OSG Viewer must be disabled (and a Console viewer can be used, or any other).

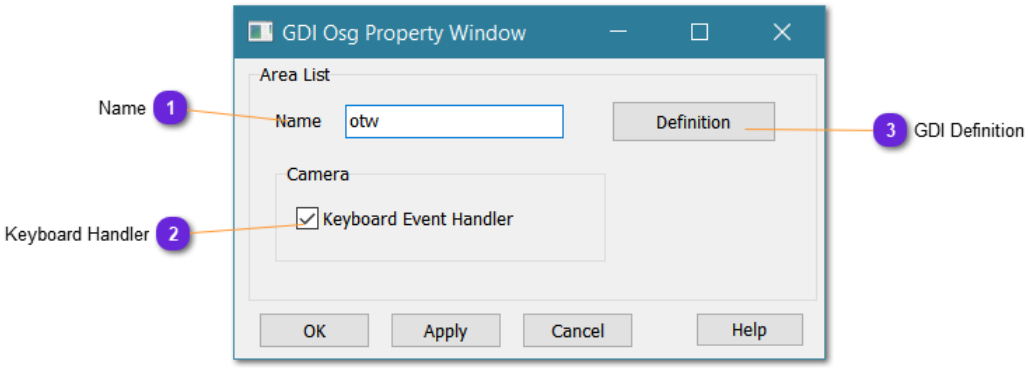
The GdiOsg filename and Camera settings have been deprecated and removed from the window. The terrain filename must now be set in the Terrain::3D Settings and the camera must be set in the OSG Viewer itself.

vsTASKER simulation engine will create only one `osg_root` node for the whole simulation.

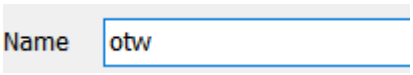
### GDI OSG

The OSG setting is done in the user modifiable /src/OSG/osg_setup.cpp
This file is compiled with the project when needed.

# Properties

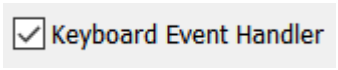



## 1 Name



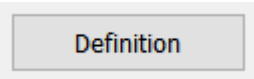

Name of the GdiOsg. Must be unique whatever its type.
vsTASKER will generate a class whose name will be `<Name>_Spt`. For the moment, this class will inherit from `Vt_Sprite` class.
Any Sprite can be accessed using its name in a generated union named `s`.

## 2 Keyboard Handler




Generates a keyboard handler for the user to use.

# 3 GDI Definition



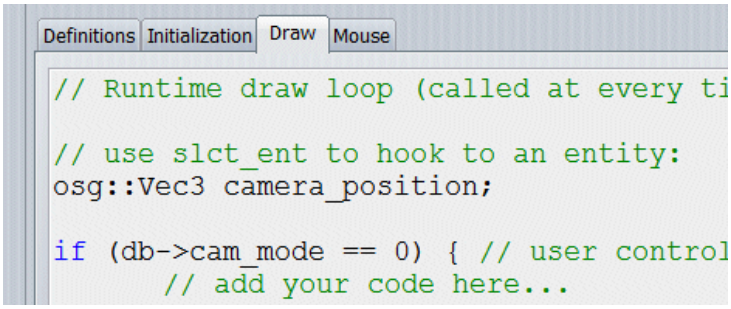

Call the GDI property window.

## • Initialization

```
case RESET: {
  slct_ent = vt_rtc->scenario->entities.find(db->cam_hook);
} break;
```

By default, the `slct_ent` pointer is set with the camera hook value (5).
NULL if `cam_hook` is empty.

## • Draw




This part can be modified. The code is defaulting a behavior for Terrain Centered and Entity Hook modes.
It can be clean to put user code in the `cam_mode == 0` block. By doing so, activating it would be done from the GdiOsg drop box itself.
In this part, the code is only setting the camera position and orientation.
It is called at maximum frequency.
User can add whatever OSG code it needs.


*If the code that must be added in this part is quite big and is mostly developed and debug in Visual Studio, it is advisable to put it into the file myOsgCode.cpp in D:\VirtualSim\vsTasker\7\Sprite\GdiOsg. This code is automatically inserted into the Draw part.*

## • Mouse

```
if (state == GLUT_DOWN) {
}
if (state == GLUT_MOVE) {
}
```

Put here the code that reacts to the mouse input (position and button) when used over the GdiOsg vindow.
Use `mouse.button` to get the actual depressed button.

*By default, the Mouse inner part takes care of the mouse to control the camera position using `ms_azim` and `ms_pitch` variables.*

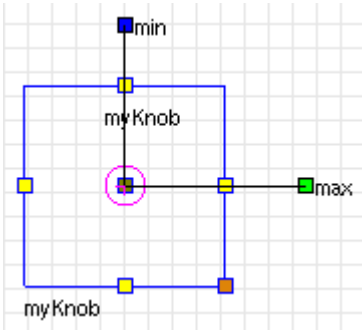
# Input

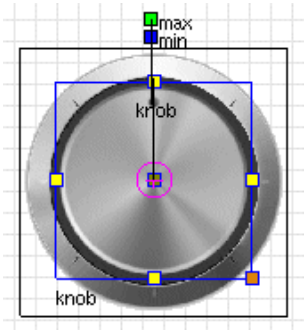
**Input** Sprites are used to give values or data to the simulation.

Although they are categorized in the **User** ⇨ **Simulation** directional way, some of these Sprite can only be used for display (like the Text Field).

# Knob

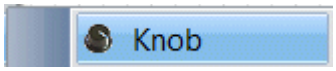
A Knob if a continuous device that the user can rotate using the mouse, between two values.
It is represented by a rotating texture above an optional background texture.




## • How to Use

On the HMI grid, add a Knob:



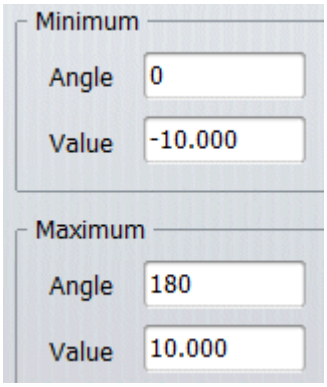
or 


Give it a name, then open the property window.

Select **Visual Aspect** then load the following texture:
\Data\HMI\Electrical\volume_knob.gif

Unckeck **AutoSize** in order to resize the knob to make it fit into the HMI panel.

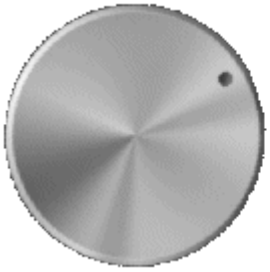
Now, in Minimum, set the value that will be attached to the minimal angle.
We set value [-10..10] for angles [0..180]

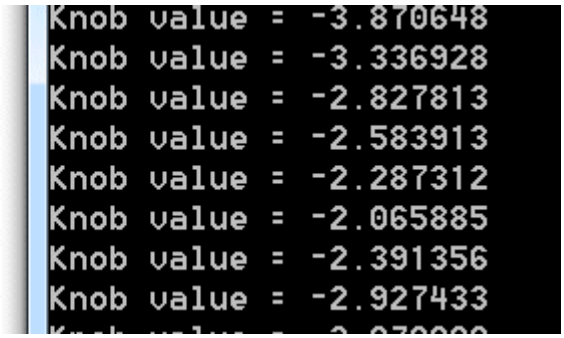


In the Runtime panel, let's add this code:

```
printf("Knob value = %f\n", get());
```

Then compile and run:



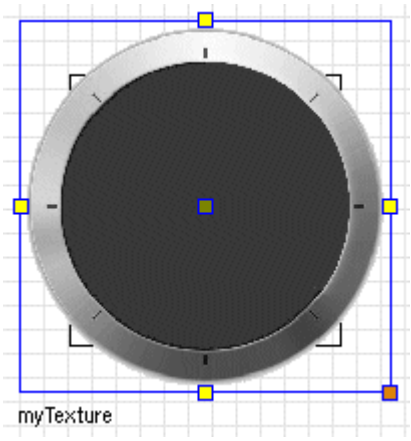



## • Background

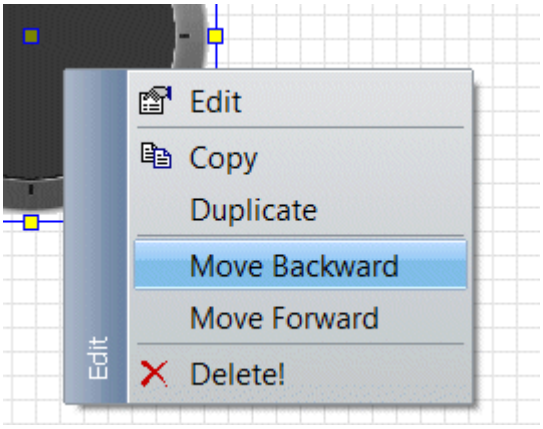
The Knob Sprite does not have a background.
It must be added manually (this might change in future versions).

Add now a simple texture (TexShape)
Load the following texture:
\Data\HMI\Electrical\volume_knob_back.gif

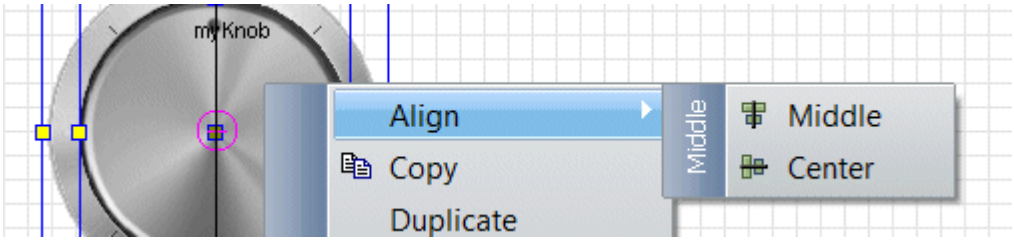
Resize and position it over the knob texture:



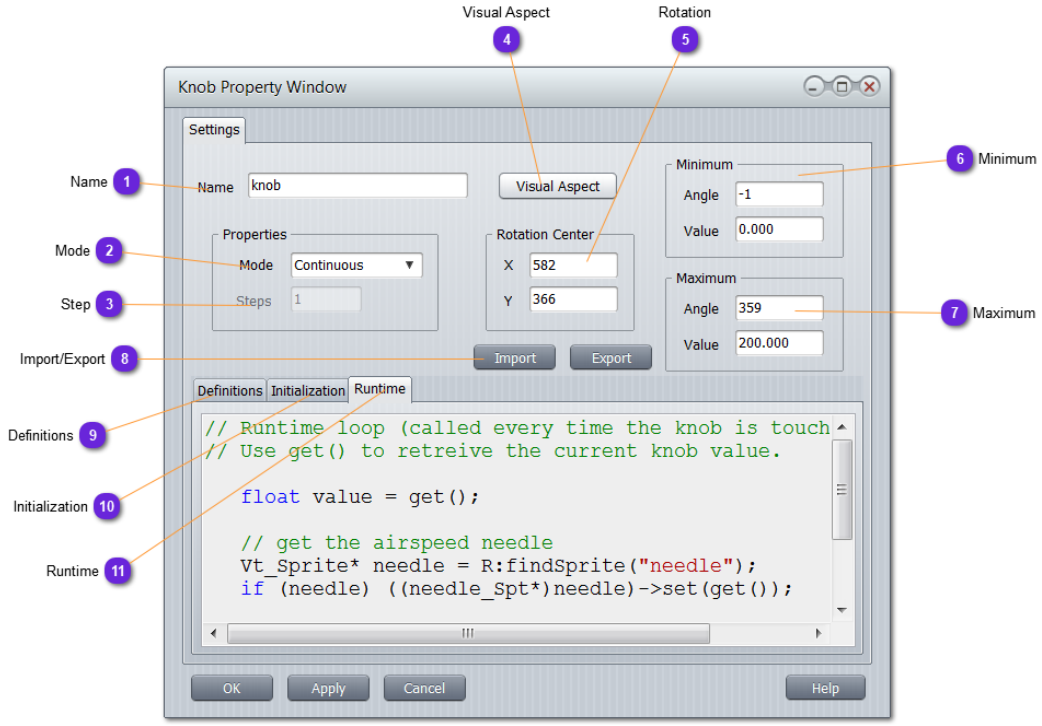

then move it backward:




You can resize it to finely adjust the knob size and position it using the keyboard arrow keys.
To automatically align (if the textures have been correctly centered), you can select both objects (select one, then the second with the Shift key down), then right click:



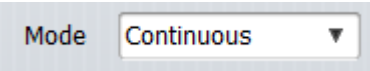
# Properties




## 1 Name


Name of the Knob. Must be unique whatever its type.
vsTASKER will generate a class whose name will be `<Name>_Spt`. For the moment, this class will inherit from `Vt_Sprite` class.
Any Sprite can be accessed using its name in a generated union named `s`.

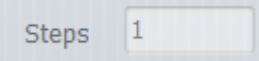
## 2 Mode




Specify how the knob turns around its center of rotation:

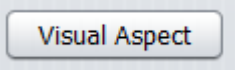
- **Continuous**: rotation varies continuously from Minimum to Maximum angles
- **Discrete**: rotation jumps from one graduation to the next one, between Minimum and Maximum angles. The number of graduations (or steps) is defined below.

## 3 Step



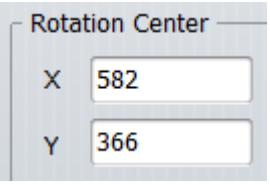

Only for Discrete mode (2).
Minimum value is 1, going then from Minimum to Maximum angle (2 values). With a value of 2, there will be one stop between the Minimum and Maximum angles (3 values), etc.

## 4 Visual Aspect



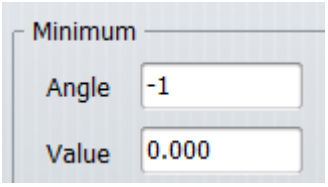

Call the Visual Aspect definition window.

## 5 Rotation




Coordinates, in the HMI design panel, of the sprite rotation center, materialized on the design display with: 
The values can be changed here for perfect precision, or using the mouse by just selecting the + cross and dragging it anywhere on the HMI.

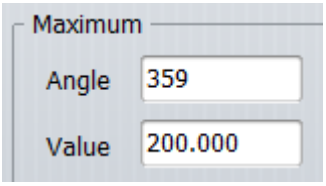
## 6 Minimum




Knob actual minimum Angle in degrees for the given Value. 0 is up, -90 or 270 is left, 90 is right and -180 or 180 is down.
Minimum angle must always be **lesser** than Maximum.


*The knob cannot rotate less than the minimum angle.*

## 7 Maximum



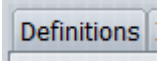

Knob actual maximum Angle in degrees for the given Value. 0 is up, -90 or 270 is left, 90 is right and -180 or 180 is down.
Maximum angle must always be **greater** than Minimum.


*The knob cannot rotate more than the maximum angle.*

## 8 Import/Export

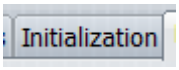

*Not available yet.*

## 9 Definitions



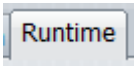

Put here all the local variables needed for this specific Knob runtime code.

## 10 Initialization




Initialize here (mostly in `RESET` part) all the variables defined in Definition section.
Default `set()` can also be used here for putting the Knob in a specific rotation (set get a value and not an angle).

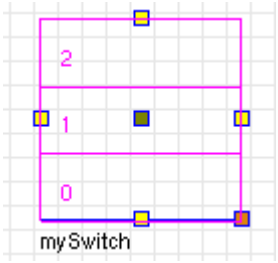
## 11 Runtime




The runtime code is called every time the Knob is touched by the mouse.
Use `get()` to retrieve the actual value of the Knob according to its rotation.

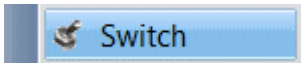
# Switch

A Switch if a discrete states device that the user can select or change using the mouse, between several states.
It is represented by an array of areas (vertical or horizontal) that can be selected using the mouse to activate the corresponding state.




## • How to Use

On the HMI grid, add a Switch:



or



Give it a name, then open the property window.

Select **Visual Aspect** then load the following textures:
\Data\HMI\Electrical\switch_on.gif
\Data\HMI\Electrical\switch_off.gif

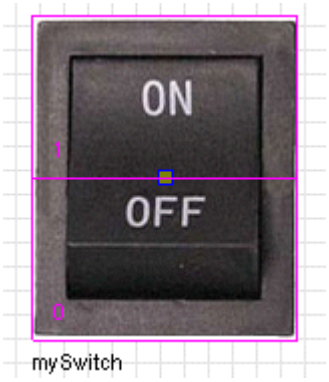
Unckeck **AutoSize** in order to resize the knob to make it fit into the HMI panel.

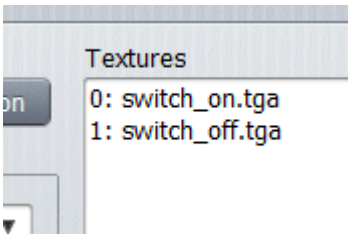
We needed 2 textures because we will create a 2 state switch. One texture will be associated with its corresponding state. It is good to remember that the state shall correspond to the area of the texture where the user will be invited to press to activate it.

In the Runtime panel, put the following code:

```
switch (db->selected) {
   case 0: set(1); break;
   case 1: set(0); break;
}
```

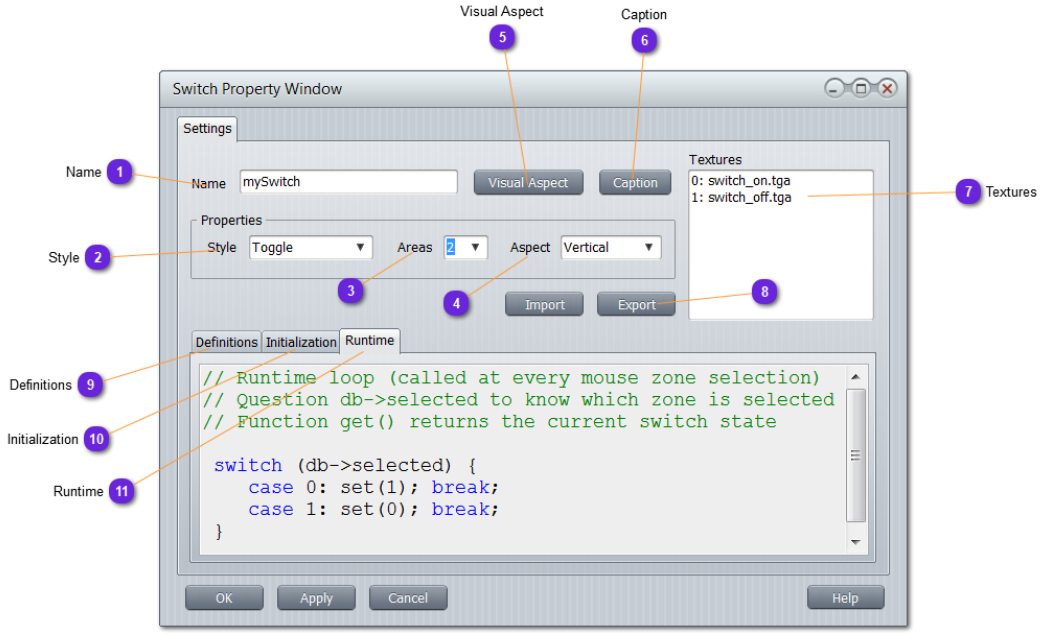
`db->selected` holds the area id selected with the mouse.
In our example, area 1 selects the ON state which is the first loaded texture with id 0. We then use `set(0)` to activate this texture.




Compile and run:

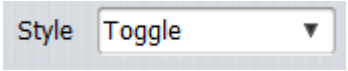
# Properties




## 1 Name


Name of the Switch. Must be unique whatever its type.
vsTASKER will generate a class whose name will be `<Name>_Spt`. For the moment, this class will inherit from `Vt_Sprite` class.
Any Sprite can be accessed using its name in a generated union named `s`.

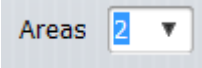
## 2 Style




Specify its behavior:

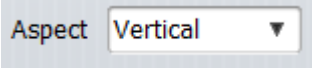
- Toggle: change from one state to one another, according to the number of areas. `db->selected` will hold the area id selected.
- Push Button: the texture is one only area. Pressing down mouse button on the area calls the runtime code with `db->selected == 0`. Do `set(1)` or more according to any other condition. When mouse button is depressed, `set(0)` is automatically called (and the corresponding texture displayed).

## 3 Areas



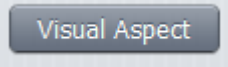

Set here the number of areas that can be selected for the Switch.
For a press button, use 1 area (full) then change the texture from 0/down to 1/up at each press.

## 4 Aspect



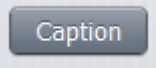

Chose here if the areas have to be horizontally or vertically aligned.

## 5 Visual Aspect



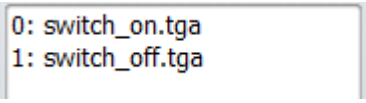

Call the Visual Aspect definition window.

## 6 Caption




Call the (optional) Label definition window to format the text that can appear below the Switch.

## 7 Textures

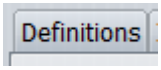



List all loaded textures that will be displayed upon a `set()` command.
The first number (**id**) is the one to be given to the command (at runtime).
For i.e, on the above list, to select the switch_off, put in the code: `set(1);`

## 8 Import /Export

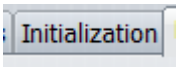

*Not available yet.*

## 9 Definitions



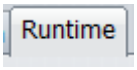

Put here all the local variables needed for this specific Switch runtime code.

## 10 Initialization




Initialize here (mostly in `RESET` part) all the variables defined in Definition section.
Default `set()` can also be used here for putting the Switch in a specific state.

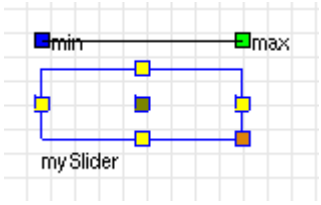
## 11 Runtime




The runtime code is called every time an area is selected by the mouse.
Use `get()` to retrieve the actual value of the Switch according to its state.

# Slider

A Slider is a vertical or horizontal graduation with a cursor that can be slide between extremities (Min, Max) to get corresponding values.
It is represented by two textures (one that slide, the cursor, and the background).




## • How to Use

On the HMI grid, add a Slider:

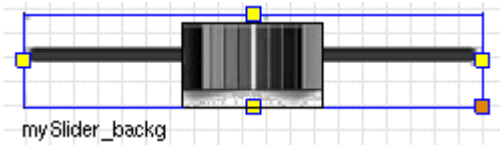
or 


Give it a name, then open the property window.

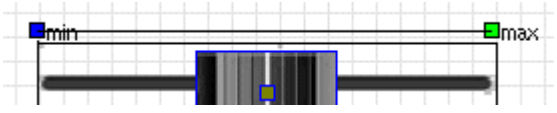
Select **Visual Aspect** then load the following texture for the cursor:
\Data\HMI\Electrical\slider4_knb.png

Select then **Background** then load the following texture:
\Data\HMI\Electrical\slider4_bkg.png
Uncheck **AutoSize** and **Hidden**.

Move the background texture shape below the cursor:




Set the Min and Max limits visually, using the blue and the green anchors:




then with the values on the text fields:

### Slider

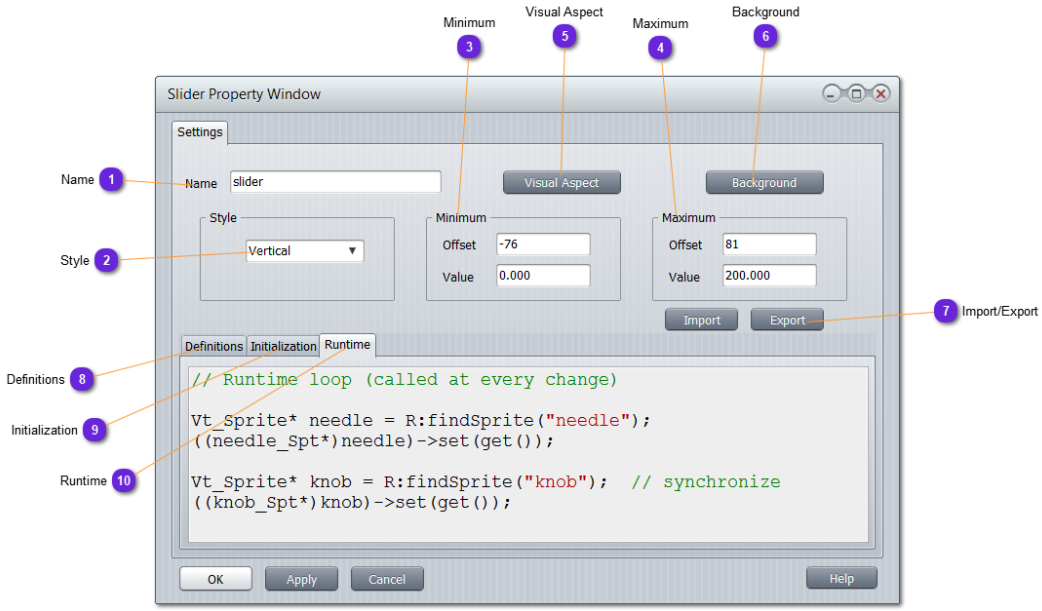

In the Initialization panel, put the following code:

```
case RESET: {
   set(-10); // minimal value
} break;
```

In the Runtime panel, put the following code:

```
printf("Slider value is: %d\n", int(get()));
```

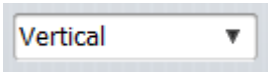
# Properties




## 1 Name

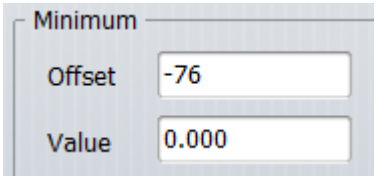

Name of the Slider. Must be unique whatever its type.
vsTASKER will generate a class whose name will be `<Name>_Spt`. For the moment, this class will inherit from `Vt_Sprite` class.
Any Sprite can be accessed using its name in a generated union named `s`.

## 2 Style



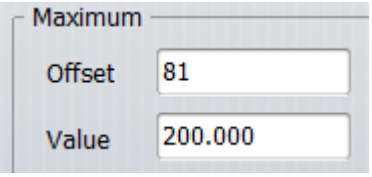

Select here the type of motion to apply to the cursor, vertical or horizontal.

## 3 Minimum



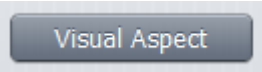

Set here the Offset value of the **Minimum** blue hook (you can also use the mouse for that) and its corresponding Value.

## 4 Maximum



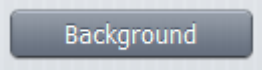

Set here the Offset value of the **Maximum** green hook (you can also use the mouse for that) and its corresponding Value.

## 5 Visual Aspect




Use this button to load the cursor texture that slide between the Minimum and Maximum values.

## 6 Background



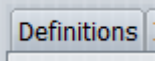

**Deprecated !**

Only visible on database which have defined a background for Slider sprites. It is advised to create a separate TexShape sprite to hold the background.
Create a TexShape with the same texture (the button is here to let you know which one is used), then delete the texture in this sprite. The button will later go.

## 7 Import/Export

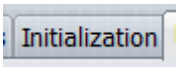

*Not available yet.*

## 8 Definitions



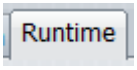

Put here all the local variables needed for this specific Slider runtime.

## 9 Initialization




Initialize here (mostly in `RESET` part) all the variables defined in Definition section.
Default `set()` can also be used here for putting the Slider at a specific position between Minimum and Maximum values.

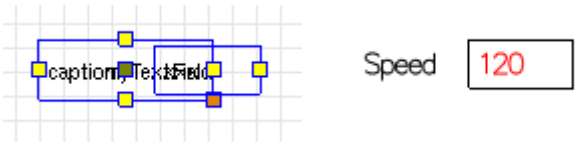
## 10 Runtime




The runtime code is called every time the cursor is touched by the mouse.
Use `get()` to retrieve the actual value of the Slider according to its position set by the mouse.

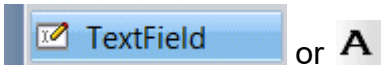
# Text Field

A Text Field is either a display field, an entry field or both.
It contains an embed caption and a display area.




## • How to Use

On the HMI grid, add a Text Field:



or 


Give it a name, then open the property window.

### Entry Field

Check **Editable** and **Display Frame**.

Click on **Caption** and write *Speed* on the text field for Caption. Check **Stroked**.
**Close**.
Click on **Field Settings** and write *120* on the text field for Caption. Check **Stroked**.
Select a Red color. **Close**.

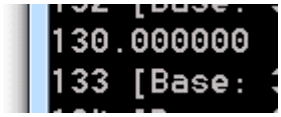
In the Runtime panel, put the following code:

```
// ... user code
printf("%f\n", atof(val));
```

### Compile and Run:

Click on the text field, close to 120, to change the value. A little cursor appears.
Use backspace key to erase and write 130.
Then press Enter.

The new value is printed on the console.


## Display Field

Uncheck **Editable**.
Check **Cycle at** and select 1 hz

On the Definition panel, put the following code:

```
private:
   int speed;
```

On the Initialization panel, put this code:

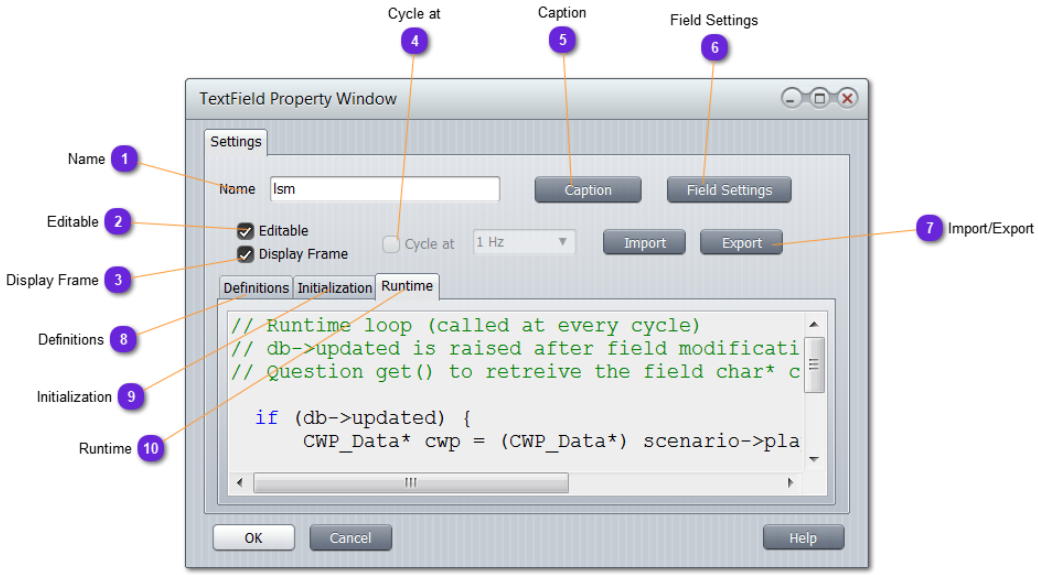
```
  case RESET: {
     speed = 0;
```

On the Runtime panel, put that:

```
  set(speed++, 0);
```

Compile and run:

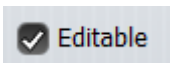

# Properties




## 1 Name

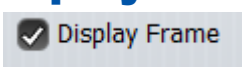

Name of the Text Field. Must be unique whatever its type.
vsTASKER will generate a class whose name will be `<Name>_Spt`. For the moment, this class will inherit from `Vt_Sprite` class.
Any Sprite can be accessed using its name in a generated union named `s`.

## 2 Editable



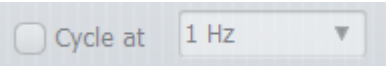

If checked, the text field can be edited with the mouse and changed with the keyboard.
If unchecked, the text field is read-only.

## 3 Display Frame



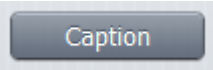

If checked, the value text field is surrounded with a frame line.
If unchecked, there will be no frame line.

## 4 Cycle at



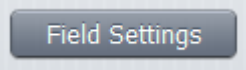

Only available for not editable text field.
If **Cycle at** is checked, chose the frequency at which the Runtime code will be called.
If unchecked, the Runtime code will not be called and the text field (data) will only be set from any other piece of code.

## 5 Caption




Use this button to set the caption font for the text field.
See here.

## 6 Field Settings

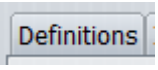



Use this button to set the data font for the text field.
See here.

## 7 Import/Export

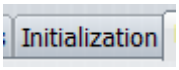

*Not available yet.*

## 8 Definitions



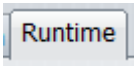

Put here all the local variables needed for this specific text field runtime code.

## 9 Initialization




Initialize here (mostly in `RESET` part) all the variables defined in Definition section.

## 10 Runtime




The runtime is called differently if the Text Field is editable or not:

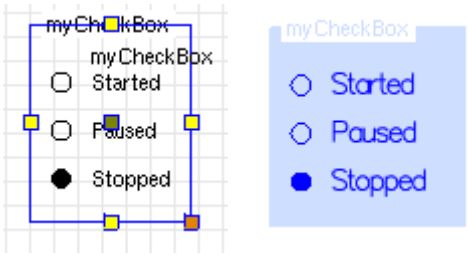
**Editable**:
Code only called when the Enter button is depressed on the keyboard.
get() retreive the data as a character string.
Up to the user to convert and process it:

```
if (db->updated) {
  char* val = get();
  // ... user code
  db->updated = 0;  // job done
}
```

**Read-Only**:
Code is not call or, if **Cycle at** checked, at the specified frequency.
Use `set(...)` to display any data on the field.

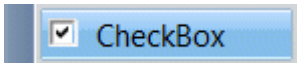
# Check Box

A Check Box is a group of either **radio buttons** (only one is checked, all others unchecked) or **check boxes** (any can be checked or unchecked).
This Sprite does not use texture and is purely OpenGL drawn.




## • How to Use

On the HMI grid, add a Check Box:



or



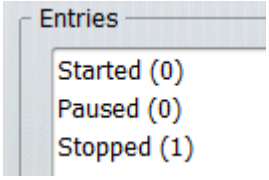
Give it a name, then open the property window.

Clean the Entries list.
Add successively the following entries:

and


Started 0
Paused 0
Stopped 1

to get:



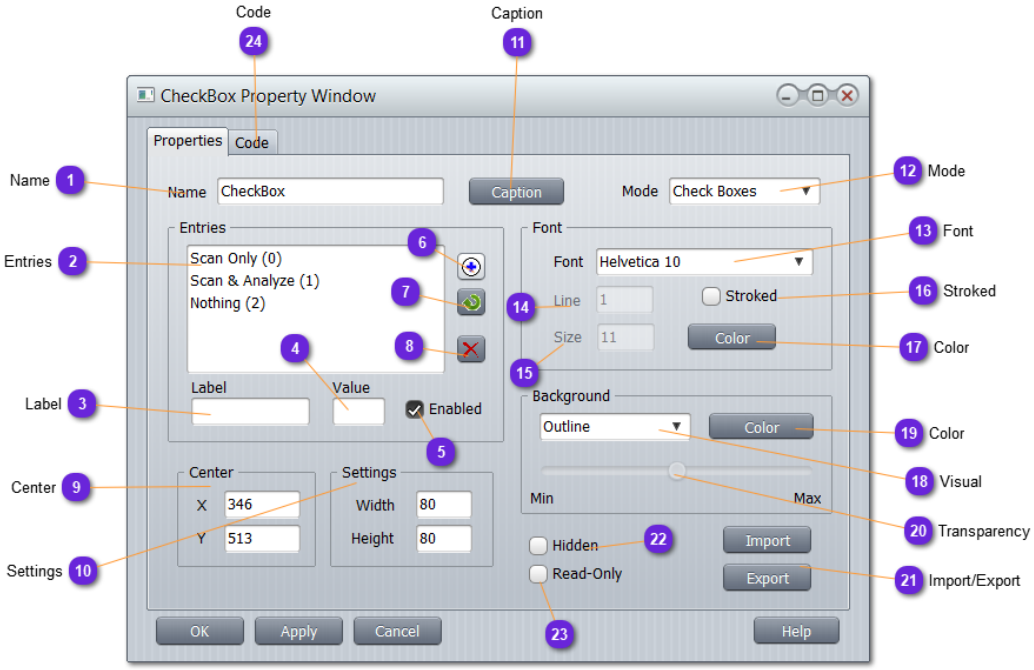

Then, in the Runtime panel, put the following code:

### Check Box

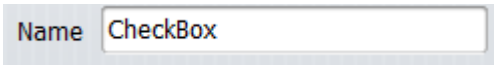
```
printf("State: %d\n", getRadio());
```

Compile and run:

# Properties

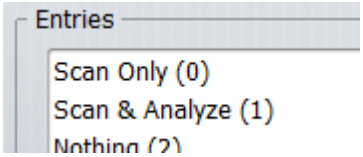



## 1 Name



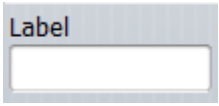

Name of the Check Box. Must be unique whatever its type.
vsTASKER will generate a class whose name will be `<Name>_Spt`. For the moment, this class will inherit from `Vt_Sprite` class.
Any Sprite can be accessed using its name in a generated union named `s`.

## 2 Entries




List of all **options** or **states** belonging to the group.

## 3 Label

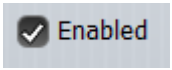



Caption of the **option** or **state** either selected in the entry list or intended to be added to the list.

## 4 Value


Initial value of the **option** or **state** either selected in the list or intended to be added to the list.
0 for unchecked, any other value for checked.

## 5 Enabled




Initial status of the **option** or **state** either selected in the list or intended to be added to the list.
When disabled, it will appear grayed out in the group. Can be enabled from the code.

## 6 Add

Append in the list the new entry line (Label, Value, Enabled).
Multiple same labels can exist in the group.

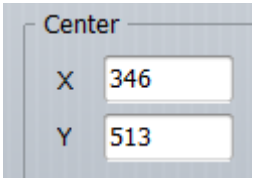
## 7 Update

Replace the selected entry in the list with the (Label, Value, Enabled) updated data.

## 8 Delete

Remove from the list the selected entry.

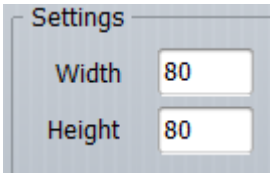
## 9 Center




Coordinates, in the HMI design panel, of the box center, materialized on the design display with: 
The values can be changed here for perfect precision, or using the mouse by just selecting the hook and dragging it anywhere on the HMI.
The Sprite can also be dragged while selecting any location and depressing the Ctrl key.

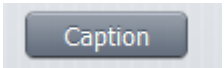

*The Sprite can also be dragged while selecting any location and depressing the Ctrl key.*

## 10 Settings



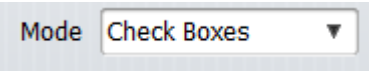

Dimension of the group outline.

## 11 Caption




Call the (optional) Label definition window to format the text that can appear below the Switch.

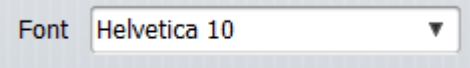
## 12 Mode




Select the behavior and the design of the entries (**options** or **states**):

- **Radio Buttons**: list of states with only one selected (filled in black).
- **Check Boxes**: list of options that can be individually selected (filled in black) or unselected (unfilled). Square box.

## 13 Font




Select the available font.
Default OpenGL fonts use **bitmaps** and vsTASKER uses `glutBitmapCharacter()` to render them. These fonts cannot be scaled so, have to be selected carefully.

## 14 Line


Width of the line drawing each character.

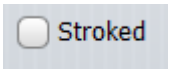

*Only for Stroked fonts.*

## 15 Size


Set the size of the character, in pixels.

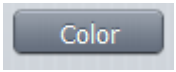

*Only for Stroked fonts.*

## 16 Stroked




If **Stocked** options is checked, vsTASKER uses `glutStrokeCharacter()` routine that renders a single stroked character from a specified GLUT stroke font (Roman and Mono Roman)

## 17 Color



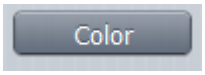

Select here the color of the font to be used on the HMI.

## 18 Visual


Select how the group box must be drawn:

- **Outline**: only the square shape will be drawn
- **Opaque**: the square shape will be filled with plain color
- **Transparent**: same as Opaque but the plain color will be transparent

## 19 Color




Color used for the outline or the background color.

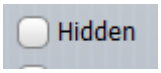
## 20 Transparency


In case of a Transparent visual, use this slider to specify the amount of transparency to apply to the selected color, from **Min** (opaque) to **Max** (glass).

## 21 Import/Export

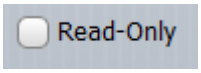

*Not available yet.*

## 22 Hidden



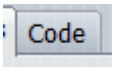

When checked, the group box will not be displayed at runtime. Can be set from code.

## 23 Read-Only




When checked, position and size cannot be changed with the mouse.

# 24 Code




## • Definition

Put here all the local variables needed for this specific Slider runtime.

## • Initialization

Initialize here (mostly in `RESET` part) all the variables defined in Definition section.
Default `set()` can also be used here for putting the Slider at a specific position between Minimum and Maximum values.

## • Runtime

The runtime code is called every time an **option** or **state** is changed by the mouse.
For the Radio Button mode, use the following function to retrieve the selection radio button (state) engaged:

```
int radio_id = getRadio();
```

If returned value is `-1`, none is depressed (might happen)
If returned value is `-2`, the Check Box is not in the Radio Button mode.

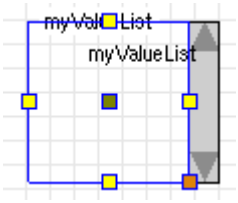
For the Check Button mode, use the following code to check the value of each options:

```
for (int i=0; i<nb(); i++) {
    int check_value = getBox(i);
    ...
```

returned value is either `0` (unchecked) or `1` (checked).

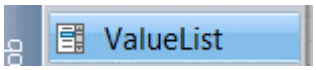
# Value List

A Value List is a table of (key, value) pairs that can be predefined at design, added or modified using code or user input with mouse and keyboard.
This Sprite does not use texture and is purely OpenGL drawn.




## • How to Use

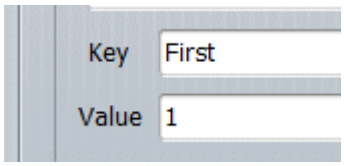
On the HMI grid, add a Value List:



or 

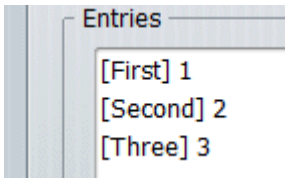

Give it a name, then open the property window.

You can add some entries manually (or from the code).
Let's do both. First, in the property windows, add the following:



then: 

Add then (Second, 2) then (Three, 3) to get:

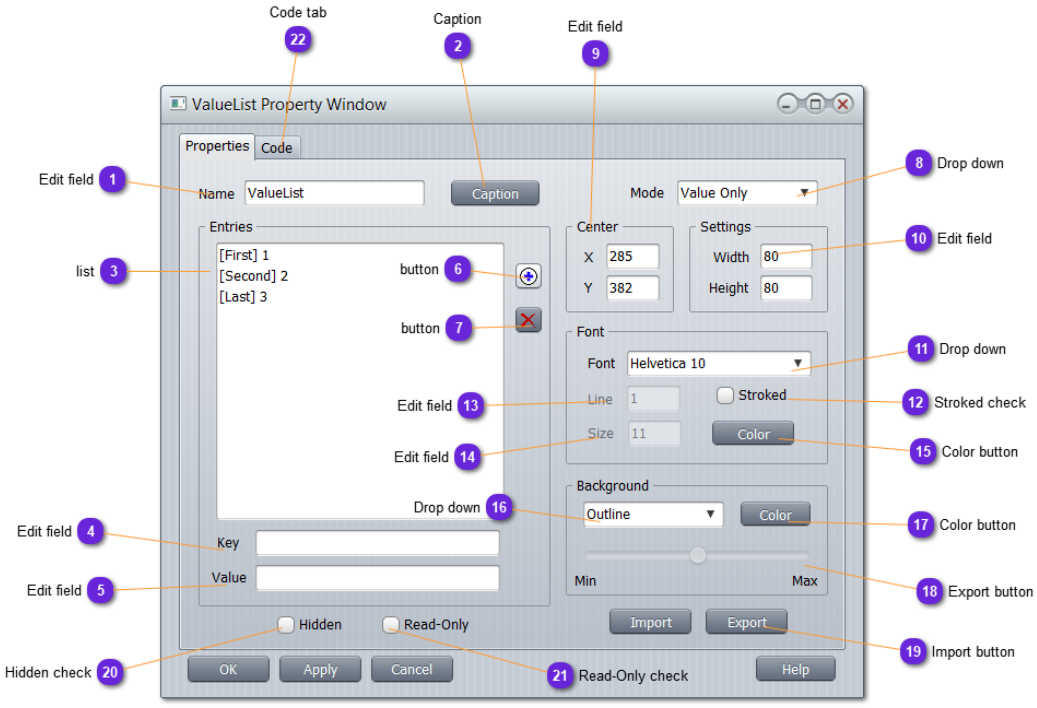



Then, open the Scenario property window and add the following in the `START` case:

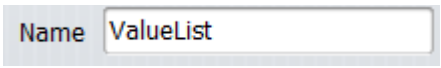
```
int selected_row = get();
if (selected_row > -1) {
   VlistEntry* vl = db->rows[selected_row];
```

```
   // get the selected entity
   Vt_Entity* selected_ship = scen()->findEntity(vl->value); // vl-
>value : ship identifier (name)
   if (selected_ship) {
     ScenarioSpecifications* scenario_specs =
(ScenarioSpecifications*)scen()->findPlayer()-
>findDataModel("ScenarioSpecifications");
     if (scenario_specs) {
       // get the current uav
       Vt_Entity* current_uav = scen()->findEntity(scenario_specs-
>current_uav_name);
       if (current_uav) target_list->db->add(entity->getName(),
entity->getName());
       // first target ? if yes automatic selection
```

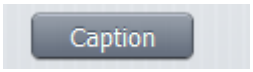
# Properties




## 1 Edit field

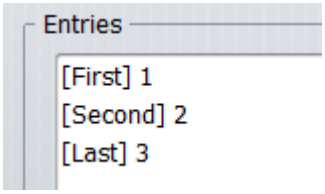



## 2 Caption




Call the (optional) Label definition window to format the text that can appear below the Switch.

## 3 list



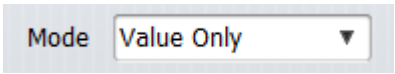
## 4 Edit field


## 5 Edit field

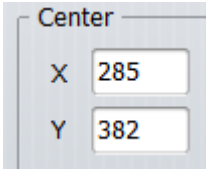

## 6 button

## 7 button

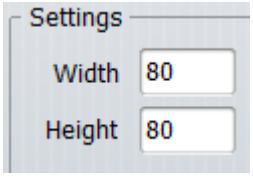
## 8 Drop down



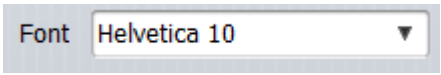

## 9 Edit field




## 10 Edit field

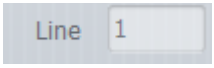


## 11 Drop down



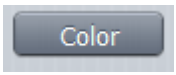

## 12 Stroked check


## 13 Edit field



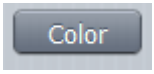

## 14 Edit field


## 15 Color button




## 16 Drop down

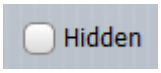

## 17 Color button



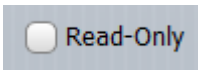

## 18 Export button

## 19 Import button

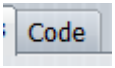

## 20 Hidden check




## 21 Read-Only check




## 22 Code tab

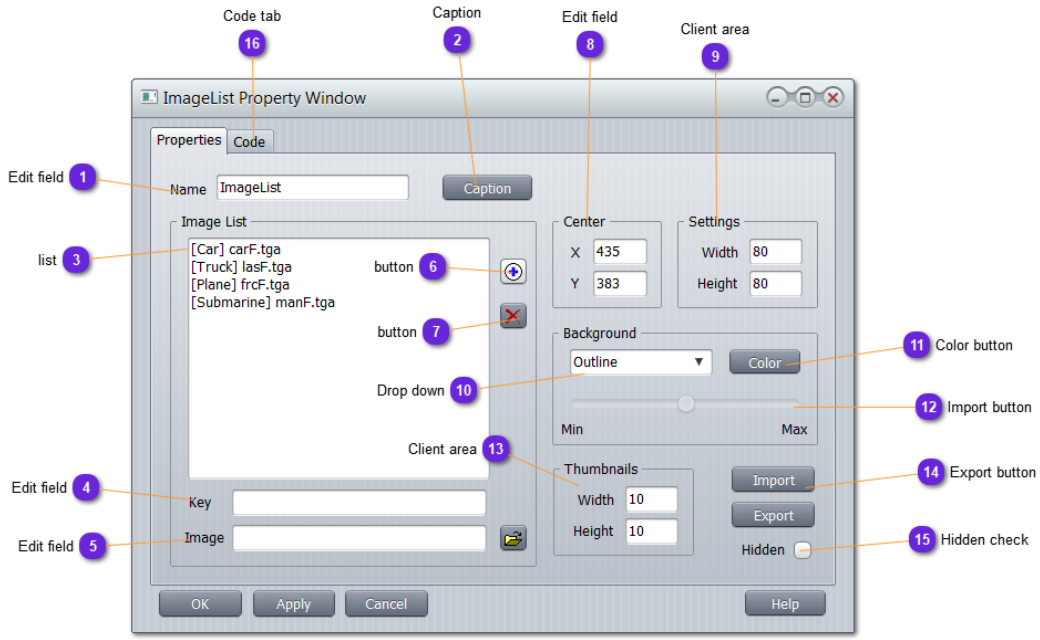


# Image List

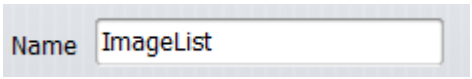
Coming soon...

```
int selected_row = get();
if (selected_row > -1) {
   IlistEntry* il = db->rows[selected_row];
   // user code...

   // get scenario specifications in order to get the current uav
entity
   ScenarioSpecifications* scenario_specs
= (ScenarioSpecifications*)S:findPlayer()-
>findDataModel("ScenarioSpecifications");
   if (scenario_specs) {
      // get the current uav
      Vt_Entity* current_uav = S:findEntity(scenario_specs-
>current_uav_name);
      if (current_uav)
```

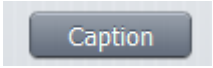
# Properties




## 1 Edit field

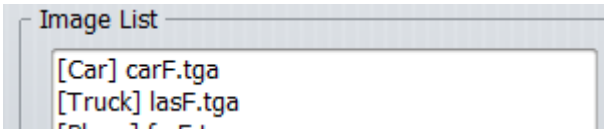



## 2 Caption




Call the (optional) Label definition window to format the text that can appear below the Switch.

## 3 list



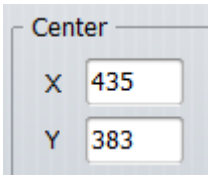
## 4 Edit field


## 5 Edit field

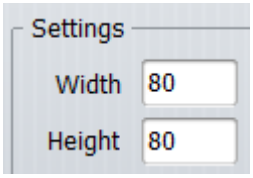

## 6 button

## 7 button

## 8 Edit field

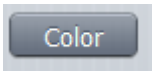



## 9 Client area



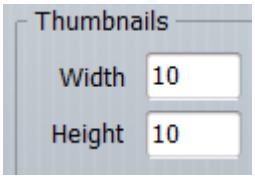

## 10 Drop down

## 11 Color button



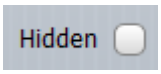

## 12 Import button


## 13 Client area

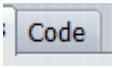



## 14 Export button


## 15 Hidden check

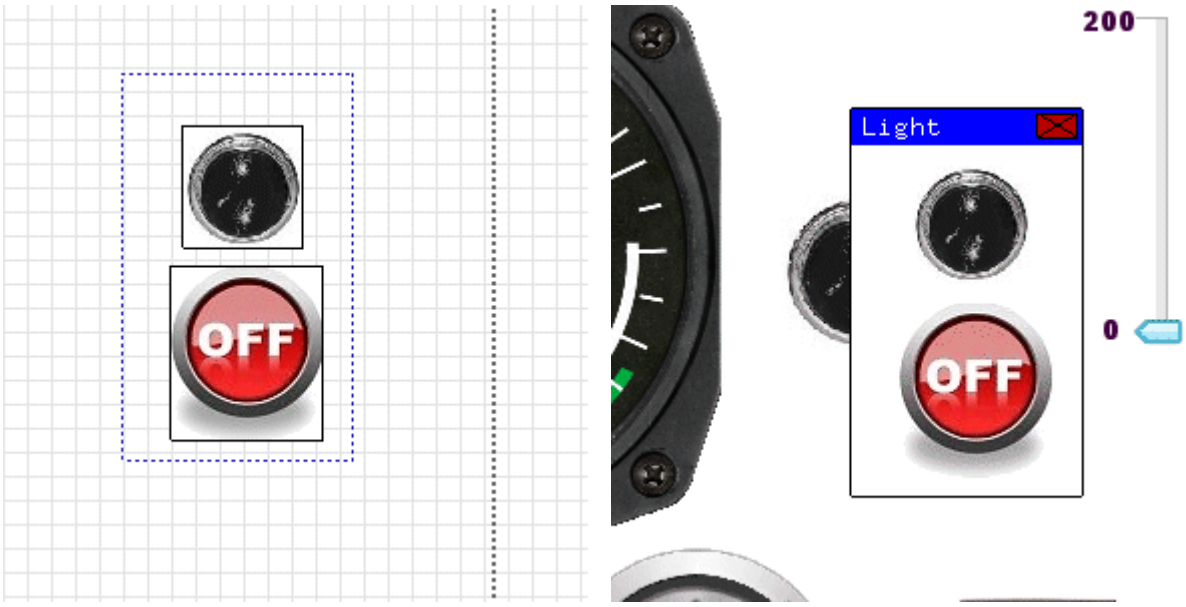



## 16 Code tab



# Windowing

Each HMI panel can hold any number of overlaying windows which can popup and hide on request, using code functions.
These windows behave like HMI sub panels.
Each of them can have any size smaller than the HMI panel they belong too.
They also can be repositioned by the user mouse on the title bar, or closed using the x button.




## • How to Use

Let's create a little window that will **popup** with a button on the Base Window (panel) and will **close** with a button on the popup window itself.

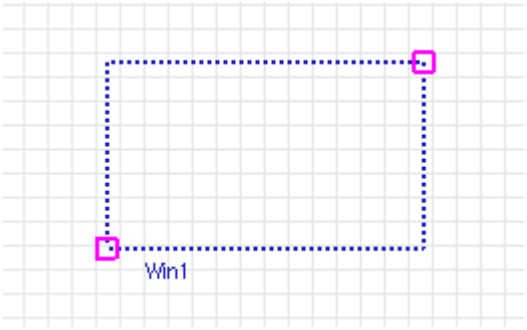
First, create a simple Push Button on the Base Window.

Load the following textures:
/Data/Hmi/Simple/button_up.gif
/Data/Hmi/Simple/button_down.gif

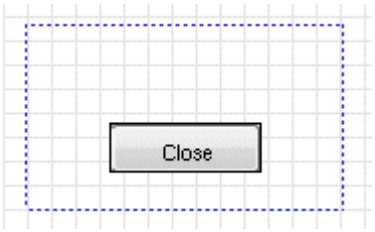
Set Open as a Caption.


Now, click on the + tab to add a new window for the panel:


Then resize the window:




Add now the close Push Button (same as above but with Close caption):




On the Runtime code of the push button, add the following code:

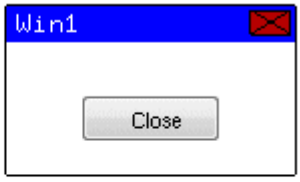
```
H:closeWindow("Win1");
```

Now, click on Base Window tab and on the Runtime code of the Open push button, add the following code:

```
H:openWindow("Win1");
```

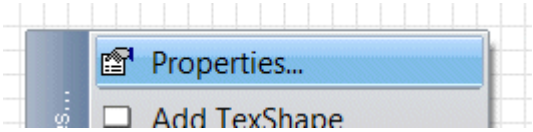
Compile and run:

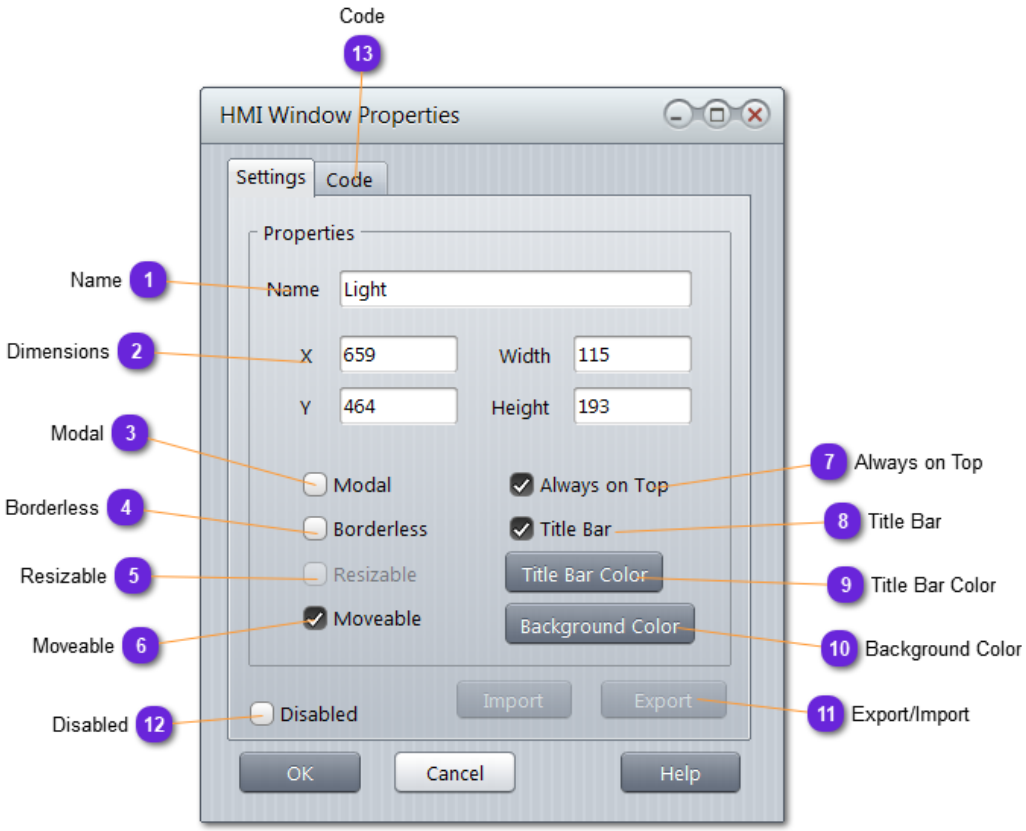
## Windowing




# Properties

To popup the window properties, when a window tab is selected, right click the mouse and select Properties.



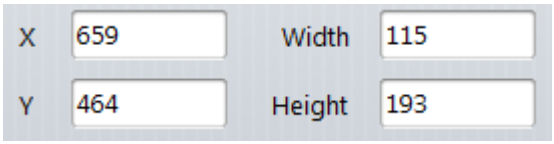




## 1 Name


Name of the window.
This name will be used in the close and open function. Must be unique.

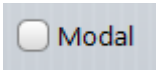

*Use `H:closeWindow("Win1");` and `H:closeWindow("Win1");` from anywhere in the code to show or hide the named window.*

## 2 Dimensions




X and Y are the lower left corner of the window. (0,0) is the lower left corner of the HMI design panel.
Width and Height are in pixels.

## 3 Modal



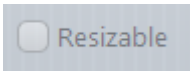

If checked, when the window is shown, only all other modeless windows and the base panel are not selectable with the mouse. Only the modal window has focus.
If unchecked, the window sprites will be selectable as well as any other sprites in other modeless windows and base panel.

## 4 Borderless


If checked, the window will appear without border and without title.


*It is not possible to move a borderless window with the mouse as the handle is the title bar.*

## 5 Resizable



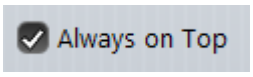

Window is resizable at runtime.
*Not available yet.*

## 6 Moveable


When checked, the title bar can be selected with the mouse to move the window over the HMI panel, at runtime.


*Window must have borders. Mouse button shall be down during the slide. Too quick motion will lose the grab.*

## 7 Always on Top

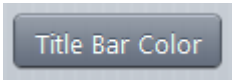



When checked, the window will always remain top of other modeless windows.

## 8 Title Bar

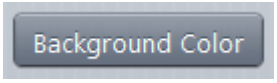

If checked, the window will be displayed with a title bar displaying the name of the window.
The title bar can be grab to move the window over the panel at runtime.

## 9 Title Bar Color




Select here the preferred color for the title bar. Default is blue.

## 10 Background Color




Select here the preferred color for the background of the window, at runtime. Useful for borderless windows.

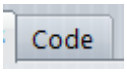
## 11 Export/Import


*Not available yet.*

## 12 Disabled


When checked, the window will not be generated, including its Sprites.

## 13 Code




User can add its own code on this part, for the different phases of the windows life:

```
case INIT: {  // simulation launch/start
} break;

case ON_OPEN: { // on window show
} break;

case ON_CLOSE: { // on window hide
} break;

case CLEAN: { // simulation stop/exit
} break;
```

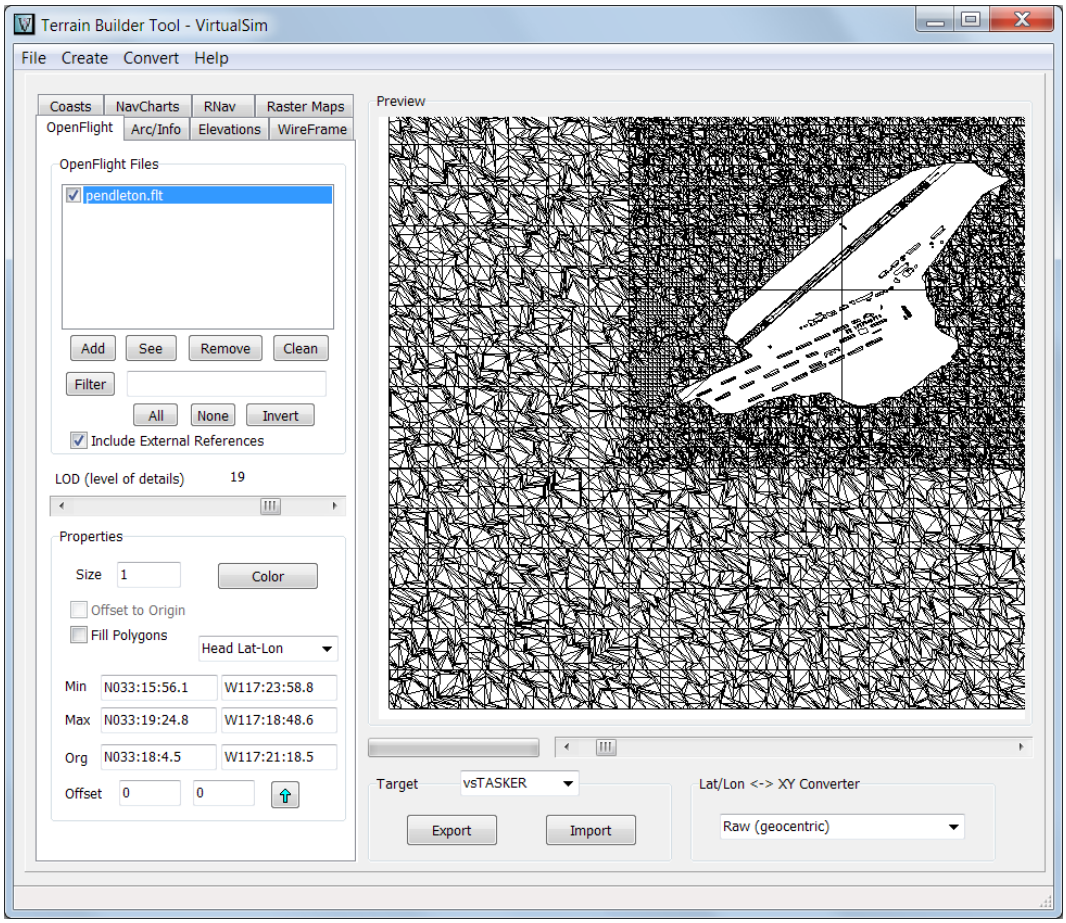
# Utilities

Utilities are software or packages provided with vsTASKER.

# TerrainBuilder

This utility tool must be used to convert standard terrain formats to proprietary vsTASKER terrain formats.
Some format can now be converted on the fly by vsTASKER but conversion in advance offers more capabilities and reduces loading time and memory footprint.

## Panel





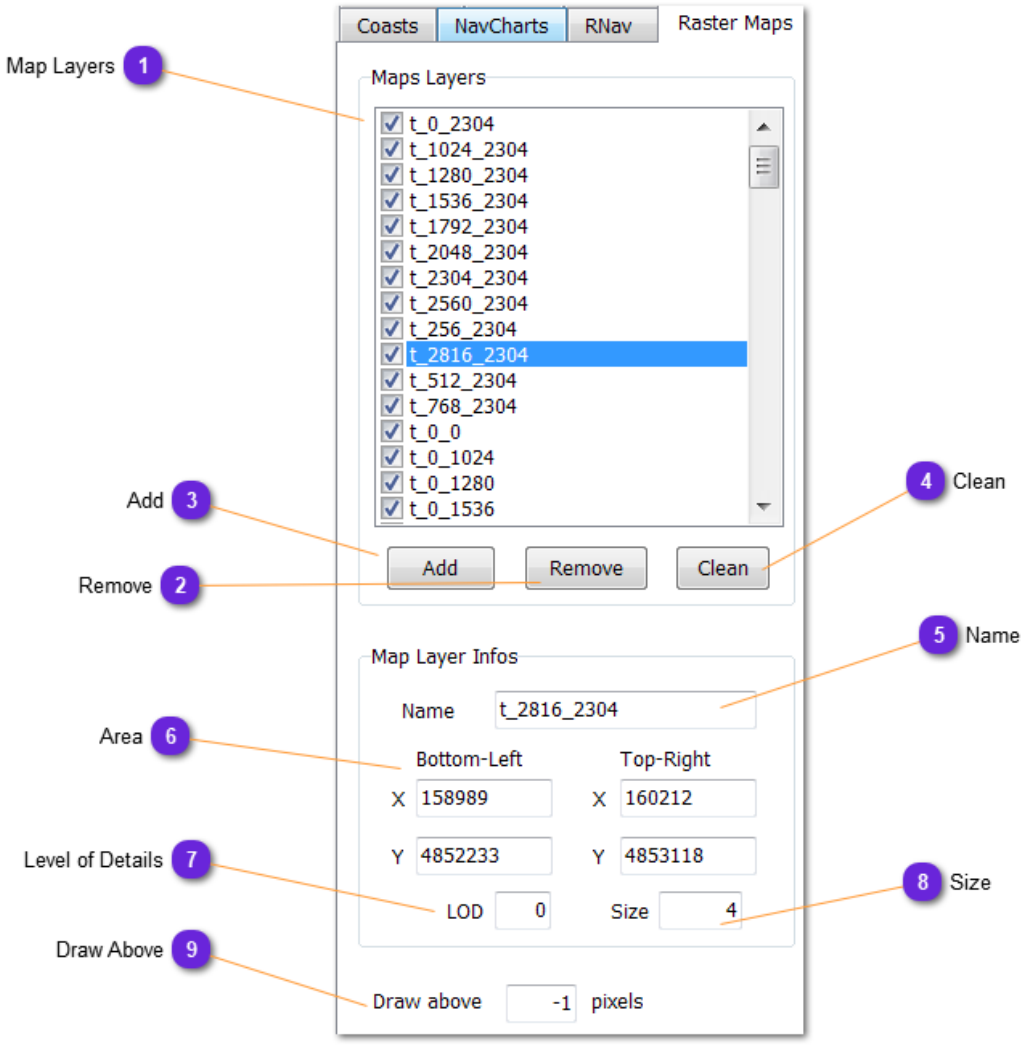
**Target**:

**Export**:

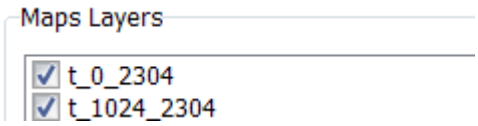
### Import:

### Lat/Lon <-> XY Converter:

# Raster Maps



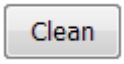

## 1 Map Layers



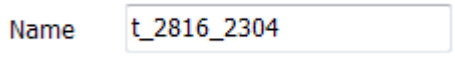

## 2 Remove

## 3 Add

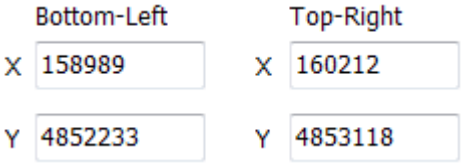

## 4 Clean



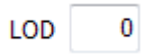

## 5 Name




## 6 Area




## 7 Level of Details




## 8 Size


## 9 Draw Above

# Tiled Maps

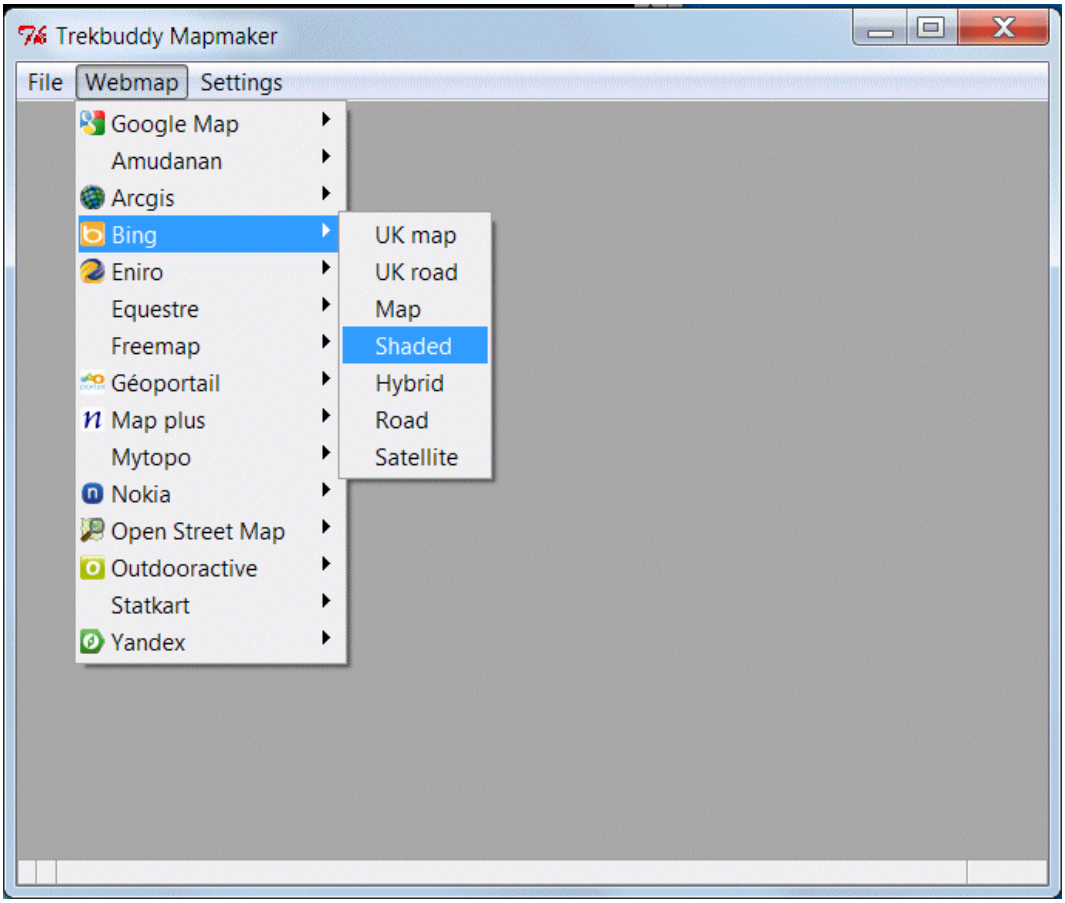
vsTASKER supports tiled map at different resolutions that must be combined using vsTerrainBuilder, from the output of TrekBuddy MapMaker utility.

in vsTASKER tool directory, extract the following file **TBmm UI-2.0.7b.zip** to wherever you want.

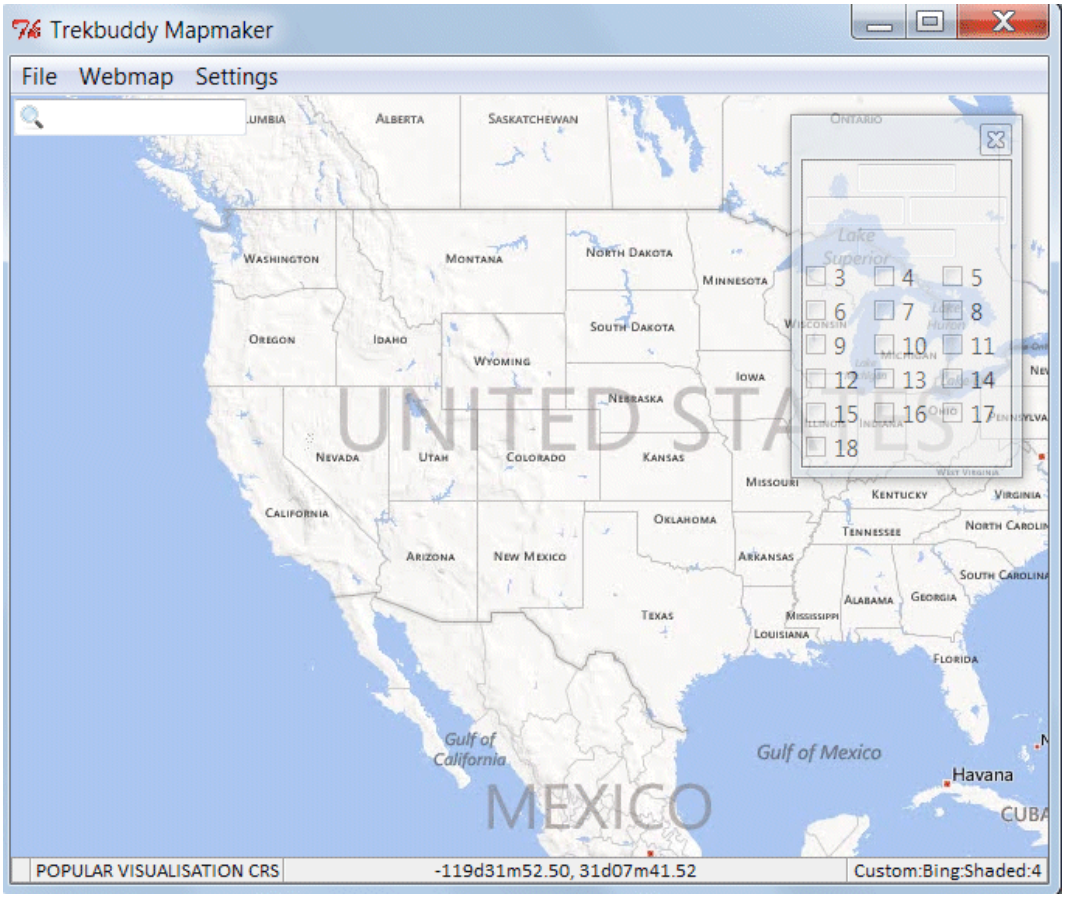
Open the folder then run **Mapmaker.exe**

If you want to extract one area with several levels of details (you can add more but the concept is still the same), select one of the map provider in the Webmap menu (note that Google Map is not available since they decide to change their API to forbid such extraction and Trekbuddy Mapmaker has not being updated so far to adapt).

We select the Bing map provider, like below.

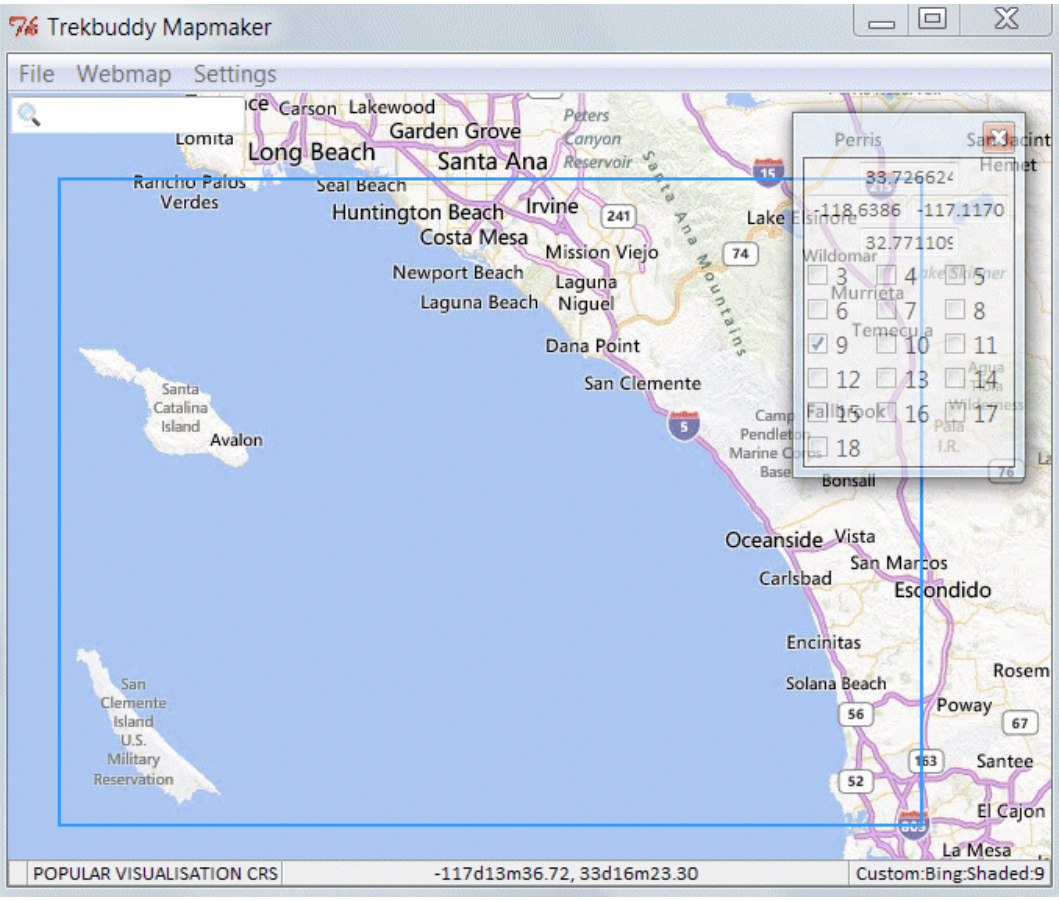



Move the map using the left mouse button.
Zoom using the mouse wheel.




We zoom (using the mouse wheel) to get close to the Camp Pendleton in California. Below is the first largest view we want.

## Tiled Maps

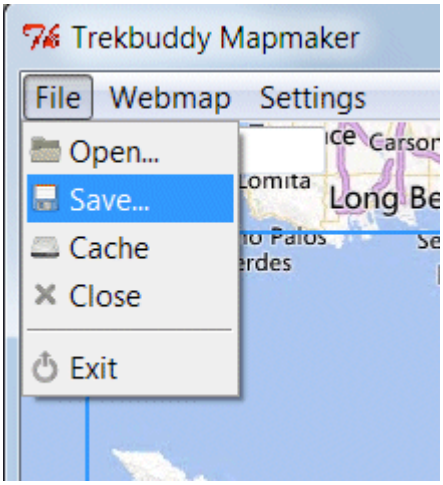



We define the area to be extracted using the left mouse button (blue rectangular area).
The area can be resized using the mouse by approaching the corners.

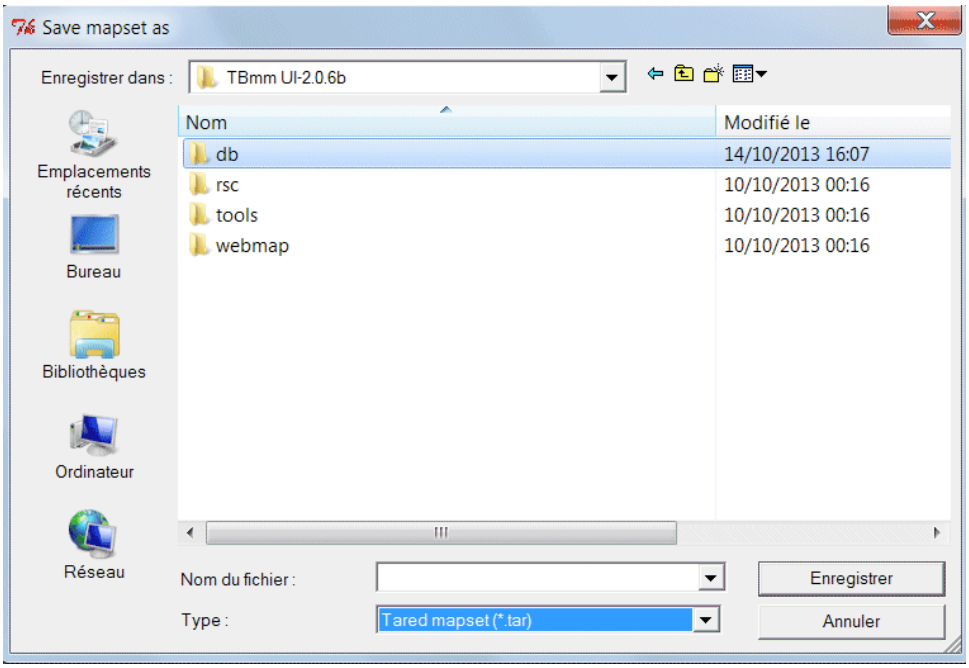
You can see that the current zoom is **9** (bottom right value of the window).
So we will check the box **9** in the zoom extractor (transparent) window.

Trekbuddy does not allow multiple zoom level extractions so, we will do the operation several time.
Remind that the number of tiles depends on the zoom level and the area size. It is advisory not to extract too many tiles because, you might be restricted from the provider (Several map providers limits 10000 tiles a day per IP) but you will also need a lot of disk space. vsTASKER has a smart way of loading/unloading maps according to the region viewed and the limitation will not be on its side.

Once selected, you can extract the layer for the selected zoom level:



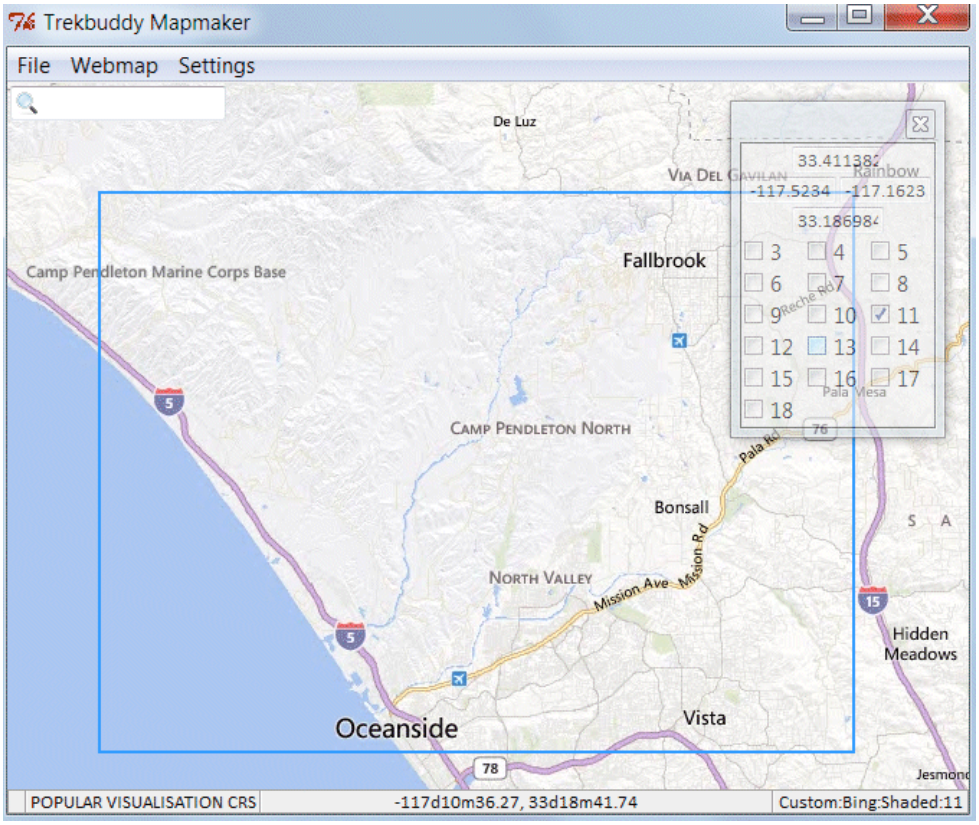

Select Tared mapset and create a sub directory in db/pendleton



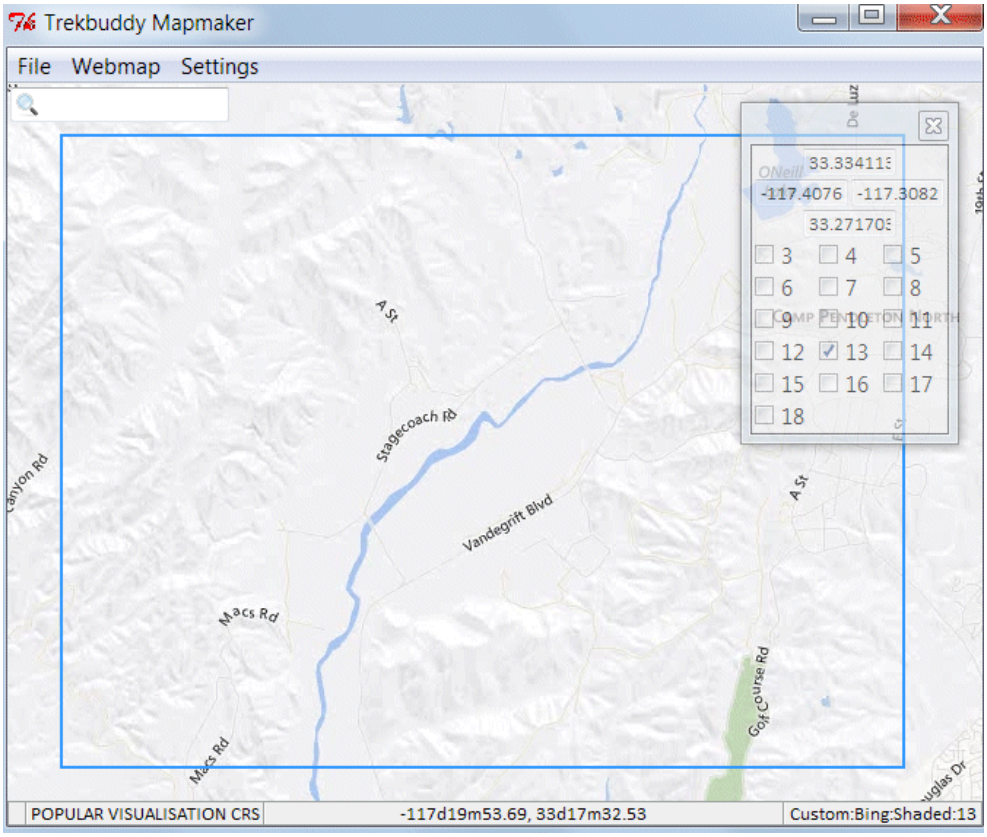

Give a name of the layer that includes the zoom level, here, pendleton09
**Save**
If the number of tiles to be saved in the tar set is not too high, you can also increase the zoom level without resizing down the area. Note that the number of tiles will be multiplied by 4.

Press ESC to get rid of the selection zone and zoom more towards Camp Pendleton (you can also uncheck the 9 box in the zoom window and check the **10** one then save again the pendleton10 layer, this is really up to you).

## Tiled Maps




Here, I will continue extracting the pendleton11 tared mapset.



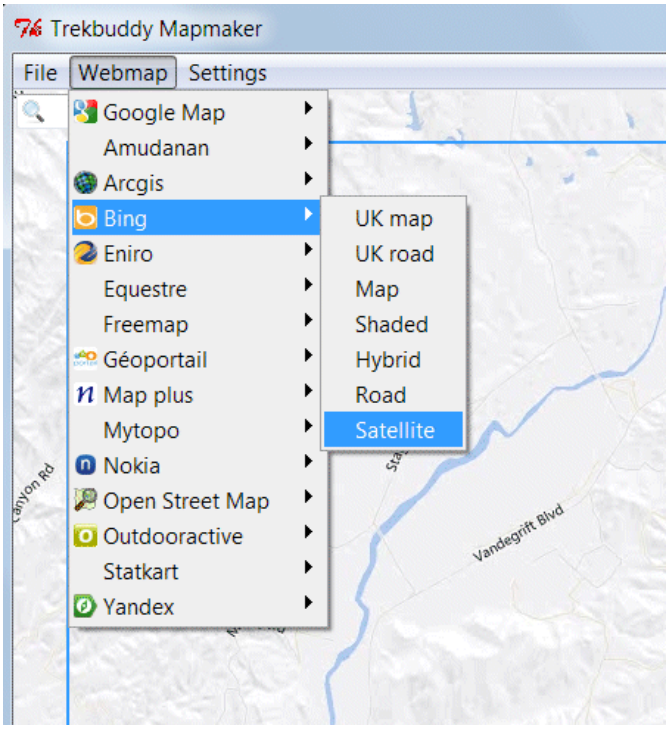

Until you get close enough to the area of interest.

Then, extract the map layer at zoom **13** and change the source to Satellite.

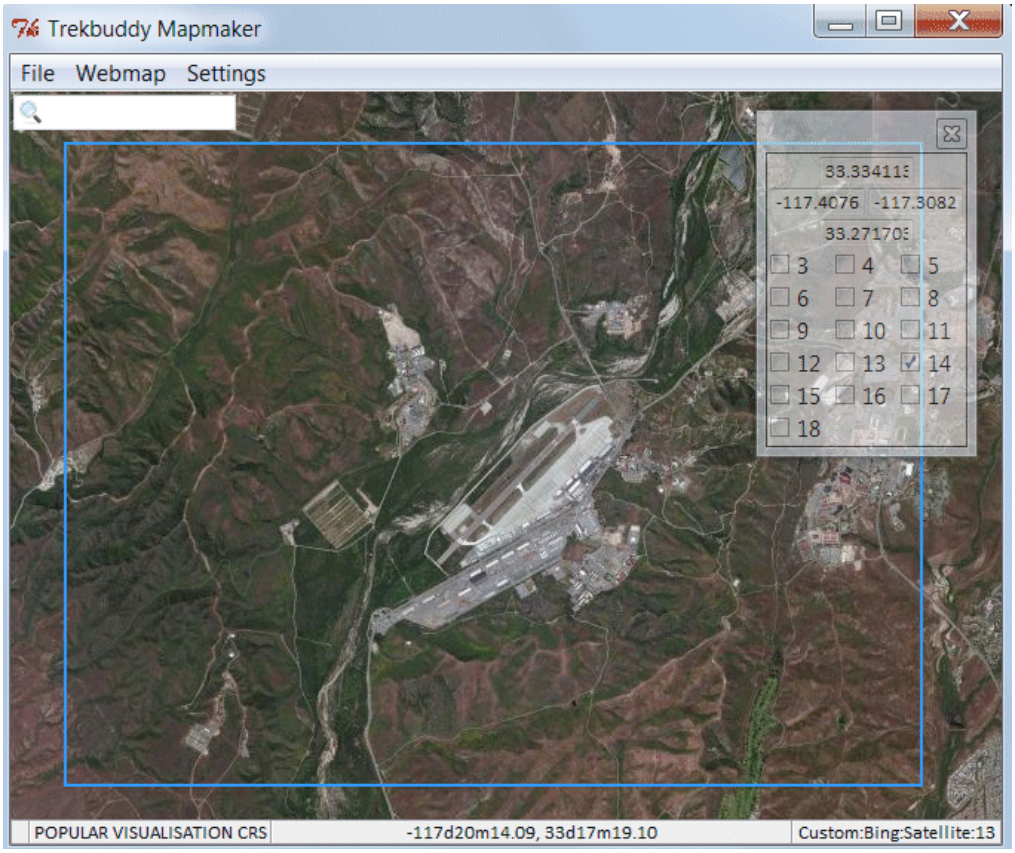
*Zoom levels do not need to be consecutive.*

Also, you can extract more than one area at a specific zoom level and gather them into the same database. For i.e, two cities of a map can have high resolution for their downtown center but the area between them can remain in low resolution (less interest).

### Tiled Maps



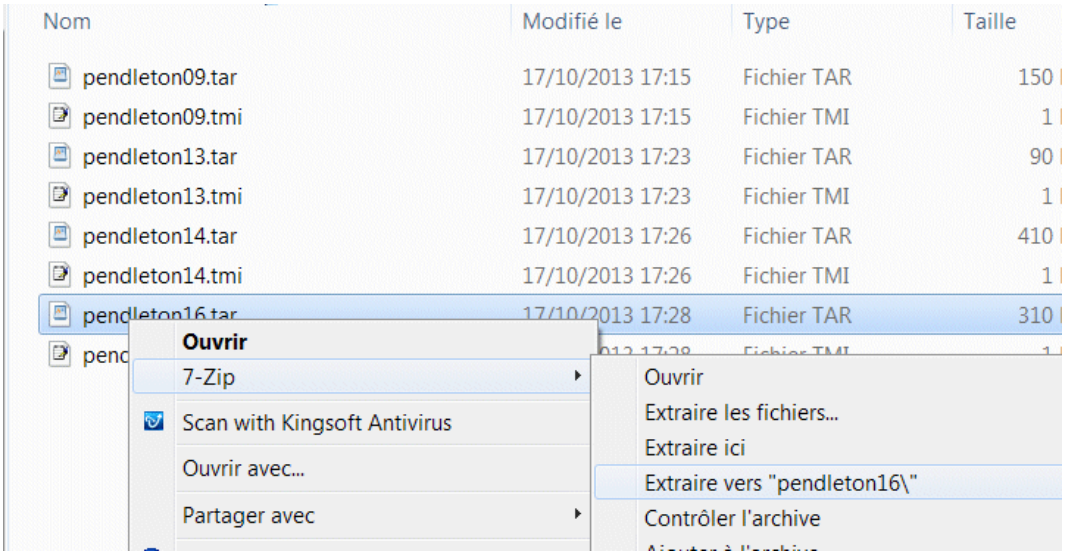

Then extract the satellite layer at zoom **14** and continue zooming in until you get to the resolution you want for your simulation.




Below is the zone extracted at zoom **16**

### Tiled Maps


Once all the layers have been extracted and saved, time to **unzip** them in order to load them all into vsTerrainBuilder

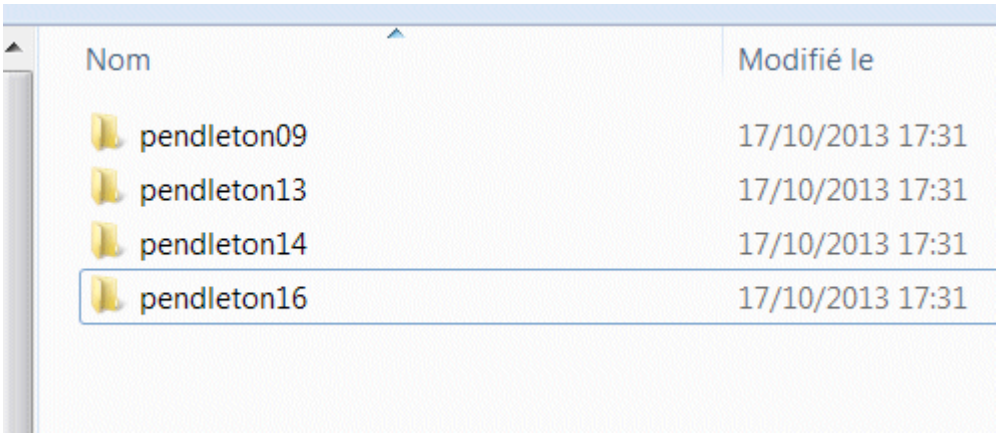



Unzip tared files into separate directories (we need to separate them because they all embed a /set directory with same filenames for image tiles. vsTerrainBuilder will

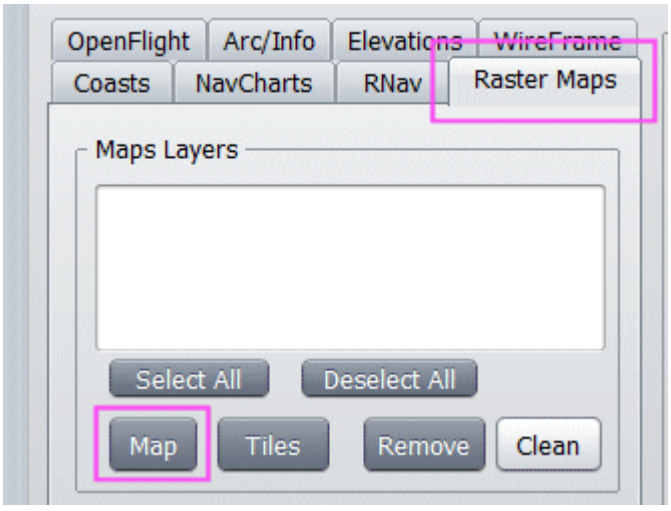
be able to rename them according to the zoom level but until then, they all must be kept separated).

Once untared, you can remove the .tar and .tmi files and keep only the directories, like below:

*Sometimes, Trekbuddy software stops to run. Just delete the directory (save the / db extracted map files) and unzip again the TBmm UI-2.0.7b.zip file.*




Now, it is time to open vsTerrainBuilder

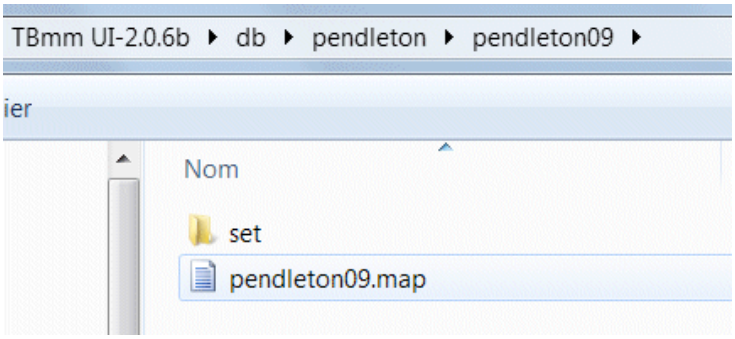



Select Raster Maps panel and click on **Map** button
You will now load all layers extracted by Trekbuddy from the **smaller zoom** value up to the **higher one**.

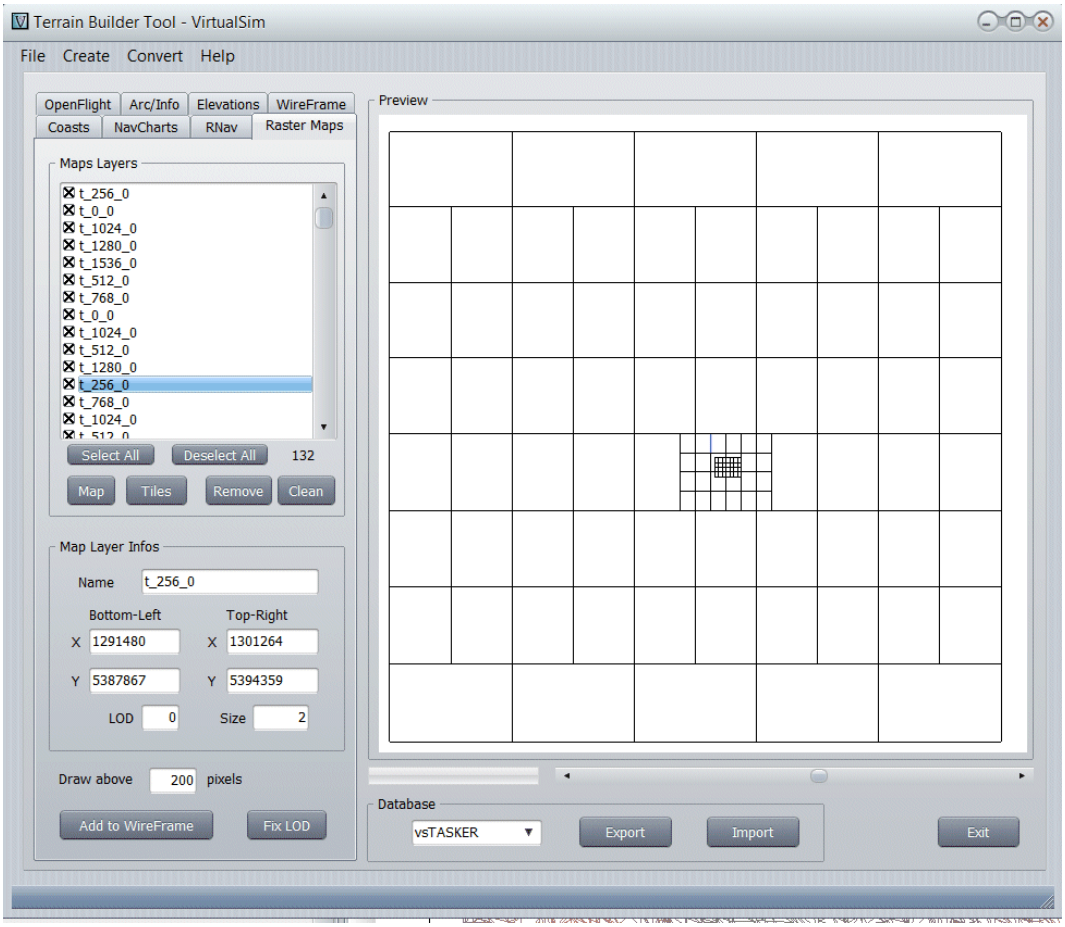
*Do not use Tiles button, reserved for map files from **OziExplorer** format*

### Tiled Maps

Add the .map file generated by Trekbuddy

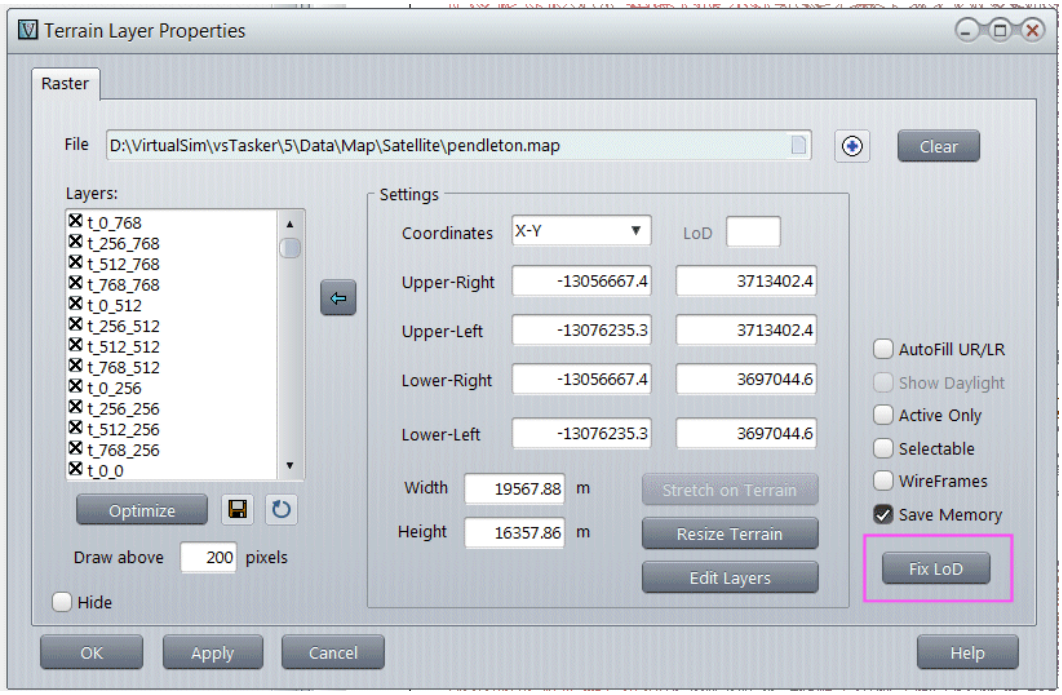



Then, continue with all the remaining layers: pendleton13, pendleton14, pendleton16
vsTerrainBuilder should look like below:



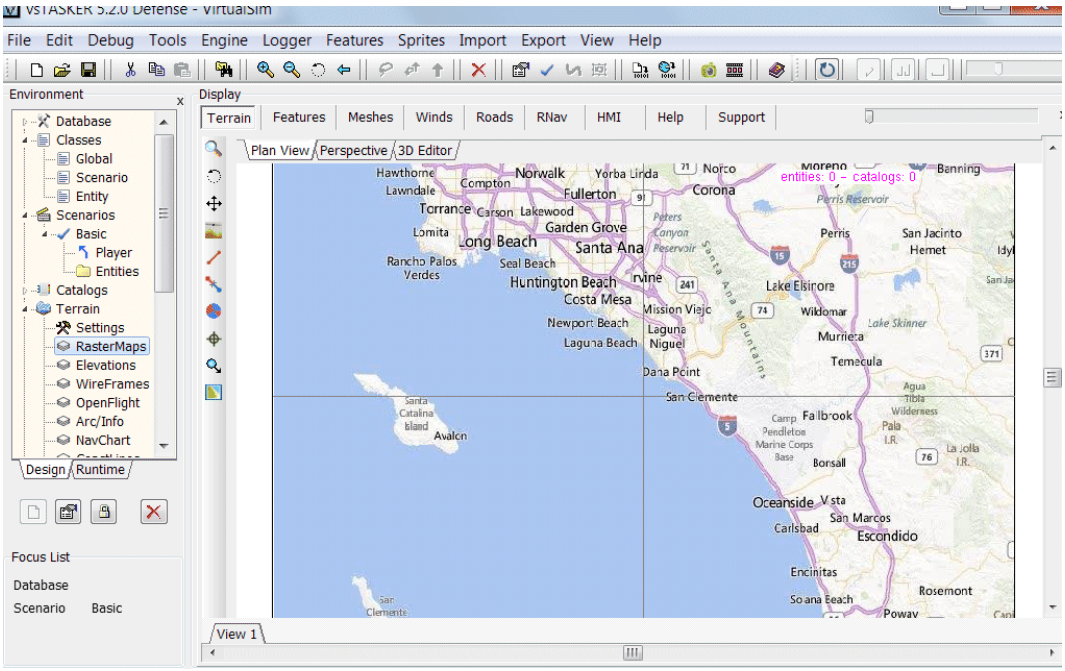

Now export the database to /data/map directory using the **Export** button. Give the name Pendleton to the database file.
Close vsTerrainBuilder and open vsTASKER.

Select Terrain::RasterMaps layer and load the Pendleton .map file (the one vsTerrainBuilder has exported and **not** the Trekbuddy one!)

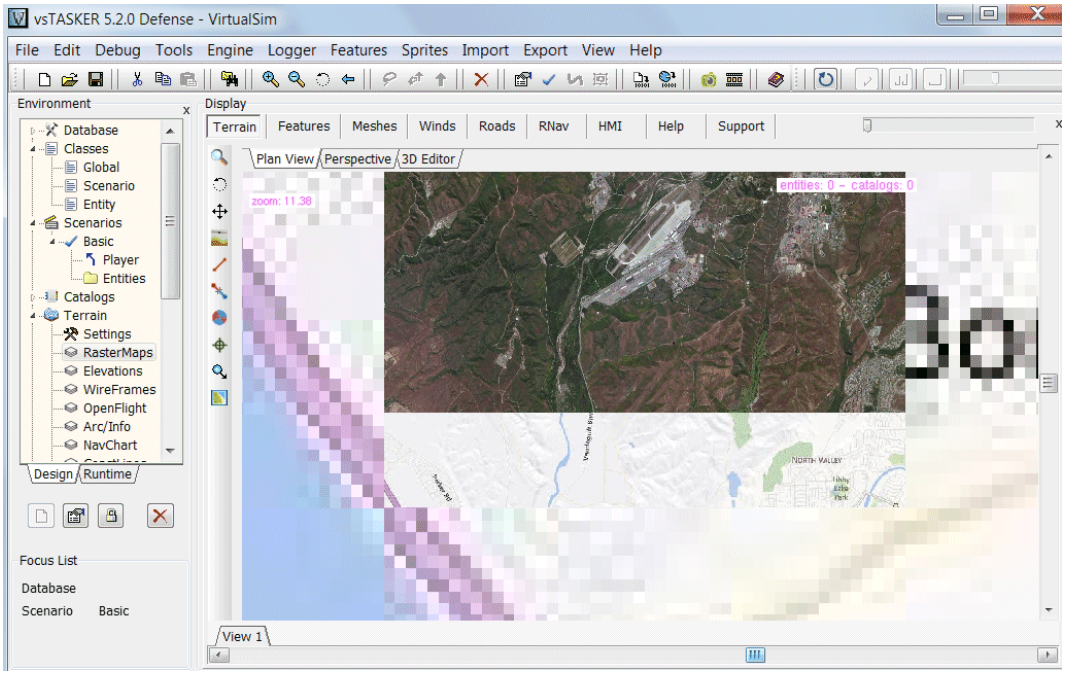



You can use Fix LoD if the database is not displaying correctly (zoom level mixed up) but then, do not forget to resave the map file with the button (left, near **Optimize** button)
You can leave or set 200 pixels value for the maximum size a tile can be drawn before being replaced by a higher resolution sets. Between 150 and 250 is a good bet.
Click on Resize Terrain to move the terrain automatically around the loaded map area, then OK.

## Tiled Maps

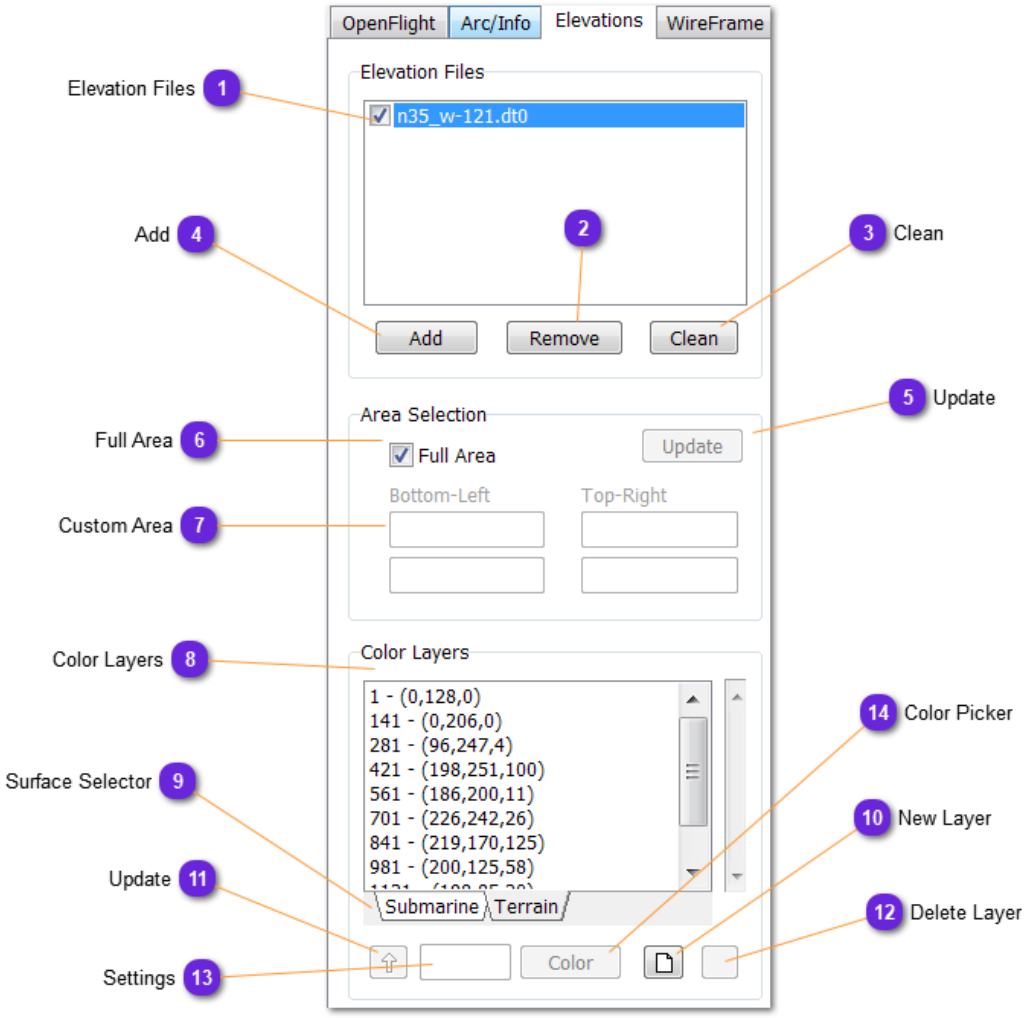



Zooming using the mouse wheel will automatically show/hide the higher resolution tiles imported into the map database.

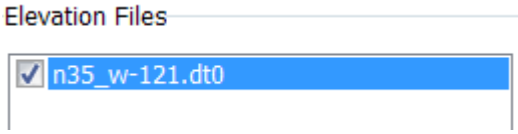



*It is good to keep the original untared TrekBuddy files as you might use them again to add more zoom level or zoom areas. You will need to go through the whole process from the vsTerrainBuilder tool without having to go back to TreckBuddy extractions.*

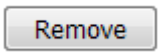
# Elevations



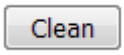

## 1 Elevation Files



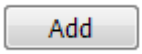

## 2 Remove



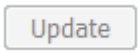
## 3 Clean



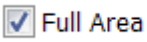

## 4 Add




## 5 Update

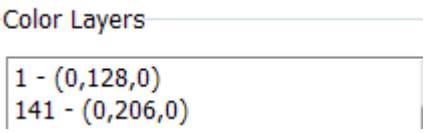



## 6 Full Area

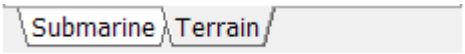



## 7 Custom Area


## 8 Color Layers




## 9 Surface Selector



## 10 New Layer

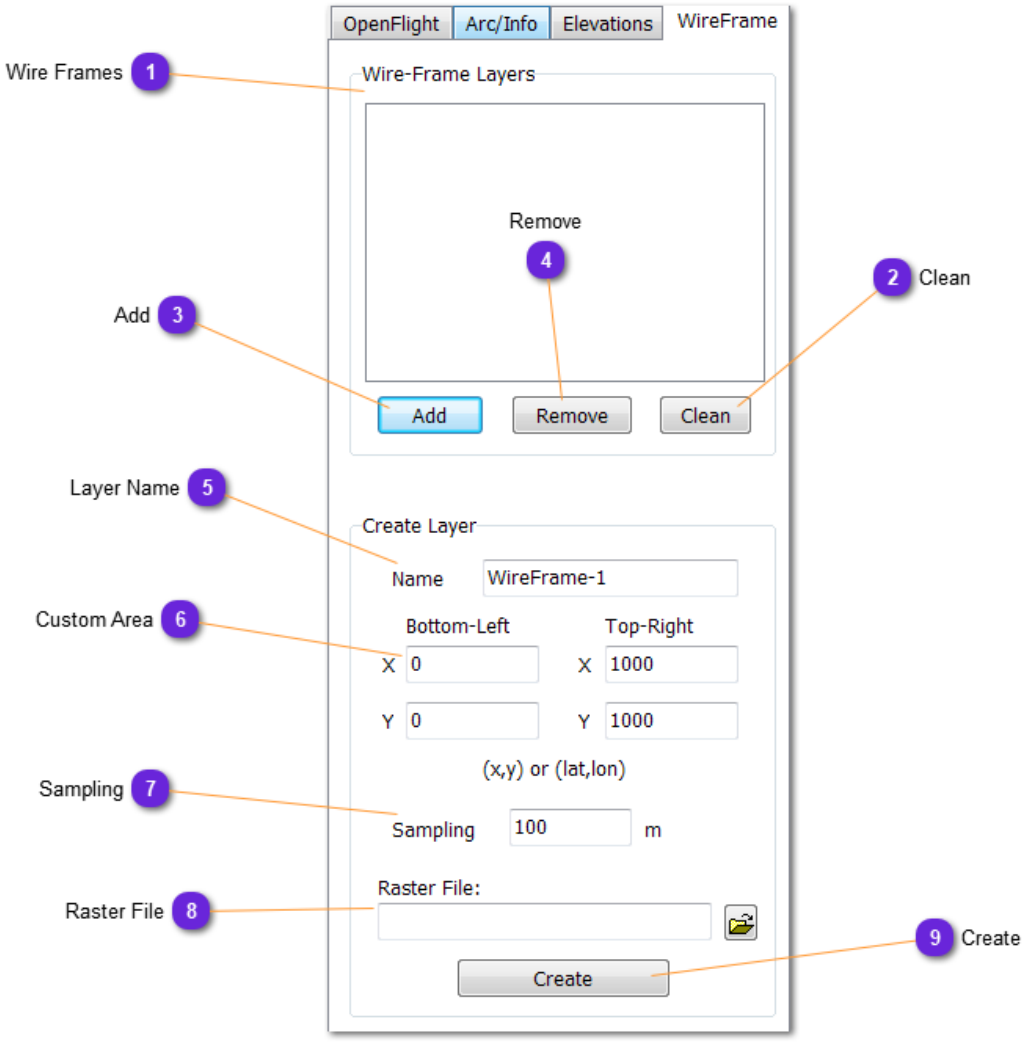
## 11 Update

## 12 Delete Layer

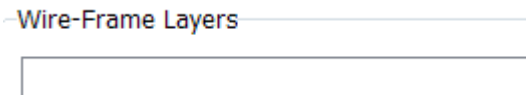
## 13 Settings

## 14 Color Picker

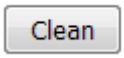
# WireFrame




## 1 Wire Frames

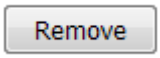



## 2 Clean

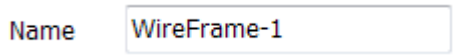


## 3 Add


## 4 Remove




## 5 Layer Name

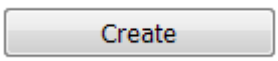



## 6 Custom Area


## 7 Sampling


## 8 Raster File


## 9 Create



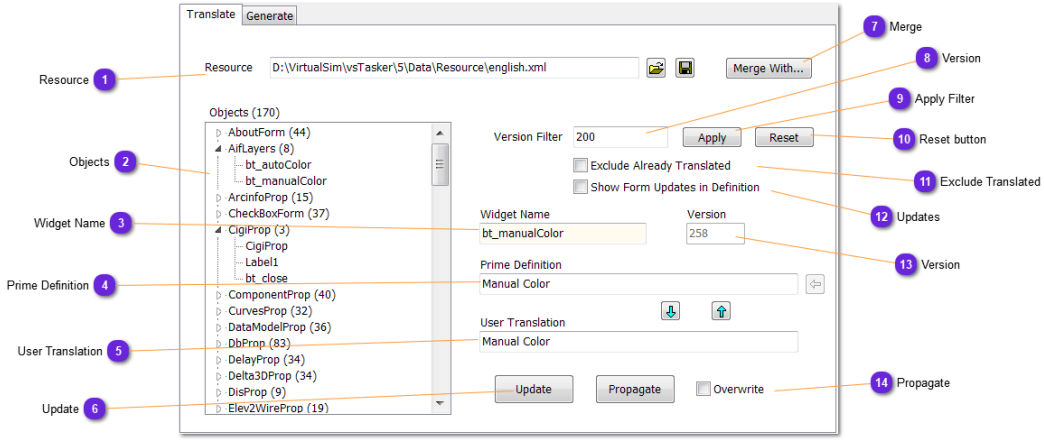
# Translation Tool

vsTranslate.exe (in root directory) is a utility that maintains a relationship between each widget id description pertaining of any CodeGear C++ Builder windows and their translated counterpart.
User can either change the label of any displayed text (label) of any vsTASKER GUI window or translate the GUI into his own language.

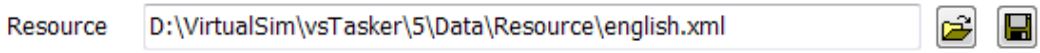
Translated database files are stored in /data/resource with .xml extension type. vsTASKER loads a file at startup for translation.

User can change the default file (english) from the menu Tools::Preference::Environment, then Forms panel.

# Translate Pane




## 1 Resource

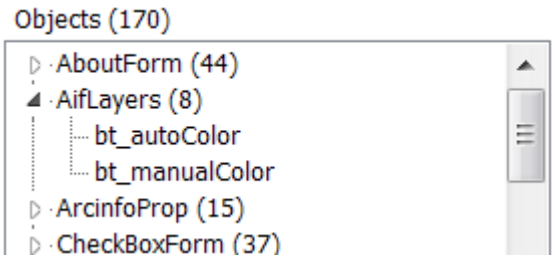



Open the translation file located in \Data\Resource. english.xml is the default file for vsTASKER.

Do not forget the save the database using  into the file after any change using the Update, Propagate or Automatic buttons.


*New versions of vsTASKER will erase the existing translation databases. If you have made some changes in the provided databases, do not forget to backup them before installing the new version ones.*

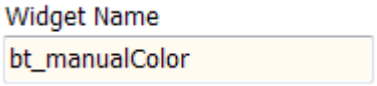
## 2 Objects




List all application windows found during the generation process. Each window contains a number of labeled widgets.

To modify the definition of any of them, expand the list and select each entry.

## 3 Widget Name




Name of the widget as defined in the GUI SDK at design time.
This value is read-only.

## 4 Prime Definition


Label or caption associated with the widget in the GUI SDK, at design time.
It is normally in English and must be considered as the prime meaning of the widget.

When the field is reddish, use the button replace the current Prime definition with the new one defined in the GUI SDK.
This can happen after Generate for widgets that have been revisited by the designer or have their meaning changed.

## 5 User Translation


Translation of the Prime definition.
Use the language as expected in the resource file.
When it exists, vsTASKER will always use the User Translation text over the Prime definition.

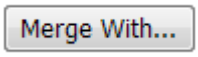
Use to copy the Prime definition locally. This can be useful when the label is long or if the wording is similar.

Use to copy the translation in the Prime definition field.

## 6 Update


Update write the change (in Prime or User fields) into the memory.

## 7 Merge

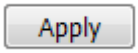



Will merge the current translation database with a given one.
All new objects from the given one will be added to current.
All new widgets from similar objects will be added.

## 8 Version


The Version filter indicates the minimum value for the widget version stamp filtering.
Putting 200 will only display widgets stamped from 200 up to `LAST_VERSION` value.
Older widgets will be ignored.
To display all widgets, put 0 in the field.

## 9 Apply Filter

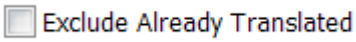



Apply the filter to the database and update the Objects list.

## 10 Reset button

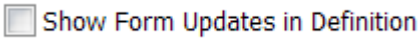

Will put the database version stamp as the filter and rebuild the Objects list.
Doing so, only new/updated objects and widgets since the last database save will be listed for translation or review.

## 11 Exclude Translated




When checked, all translated widgets (those with User translation fields differing from Widget name) will be skipped.
Checking this box is useful to finish or update a translation database when a new version of vsTASKER is received.

## 12 Updates




When checked, only the entries whose Prime definition differs from the GUI SDK labeling will be listed.
To get the conflicting labeling (as defined in the GUI SDK by the user), click on  right of the Prime definition field.

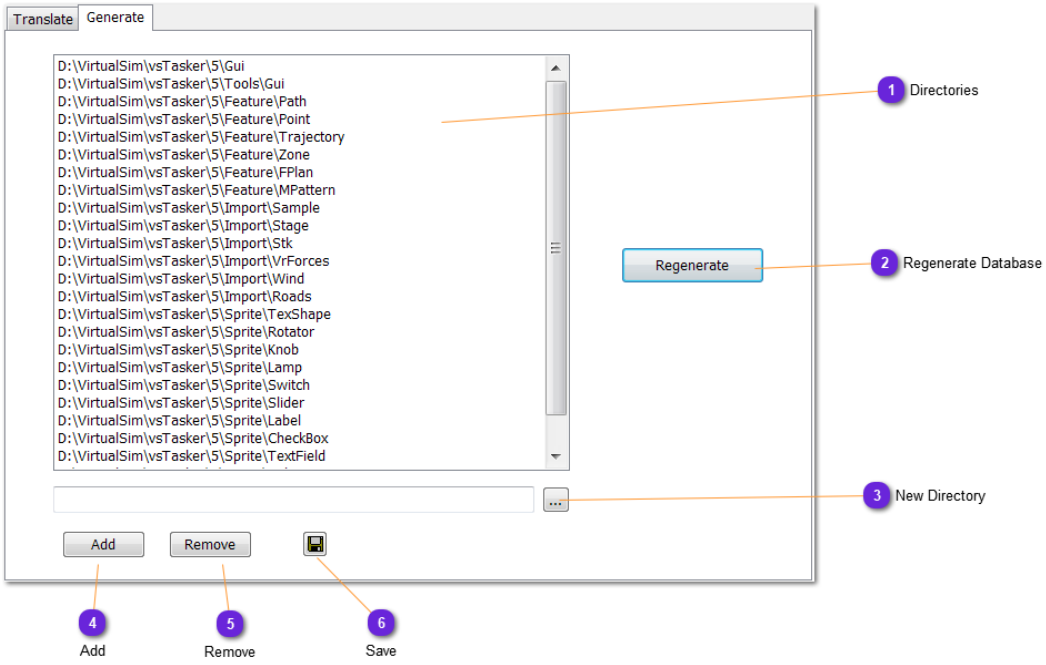
## 13 Version


Value of the `VERSION` (as defined in include/macros.h) when the item was created in the GUI SDK.
The smaller the value, the older the widget.

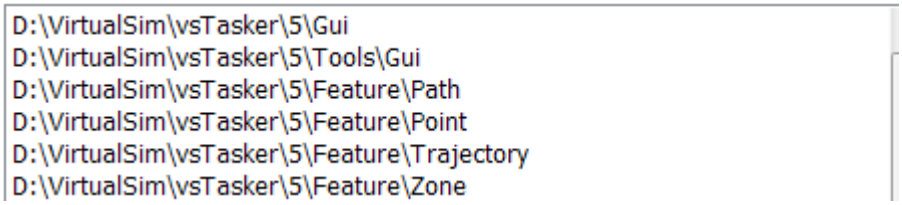
## 14 Propagate


Does as Update but propagates the translation to all widgets having the same Prime definition.
Check Overwrite to force the propagation to all widgets and not only previously translated ones.

# Generate Pane



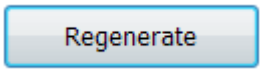

## 1 Directories




Lists all the directories that will be parsed to get the .dfm files (GUI definition files from CodeGear C++ Builder) to generate or update a translation database.

Any entry of the list can be selected for removal.

## 2 Regenerate Database



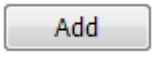

Click this button to force the engine to visit all listed directory and build or update a translation database.
If a database has been loaded (see Translate pane), the engine will update the database. If no database is loaded, it will be created from scratch.

Use this button each time new windows (using Translate() function into their constructor) is created or new widgets added in any existing window of the GUI SDK.
Database must be updated and saved to prevent crash at startup (or window popup).

## 3 New Directory


Use this button to add a new directory to be parsed by the engine.
This directory must contain .dfm files.

## 4 Add




When a new directory has been selected (3), this button adds it to the list (if not already there)

## 5 Remove

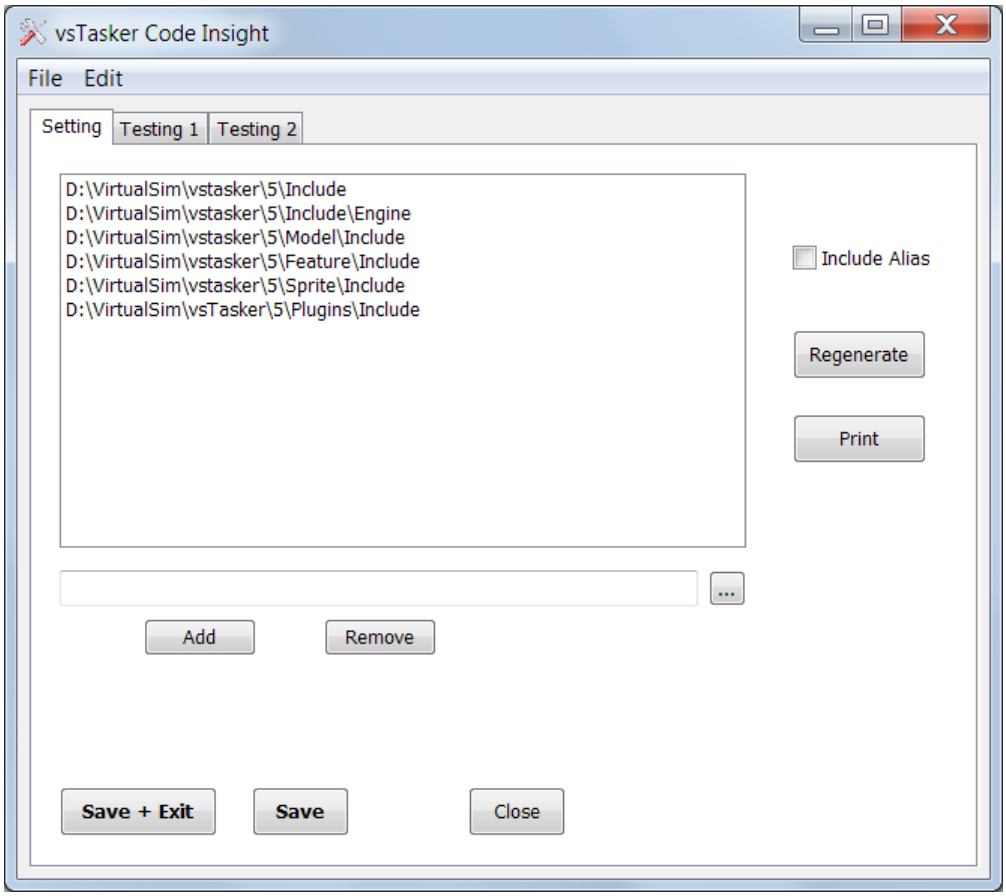

If a directory is selected in the list, use this button to remove the entry from the list.

## 6 Save

When the directory list has been modified (add or remove), this button will save it into the data/resource directory for later retrieval.

# Code Insight

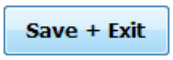
vsCodeInsight.exe (in root directory) is a utility that rebuilds on demand the completion database, by scanning all header files (.h) in specified directories. vsTASKER default database has been built using the provided header files. Whenever the user add his own header files, either in components, sprites or features, it might be useful to update the completion database.



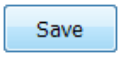

Regenerate button **rebuilds** locally the entire database by parsing all header (.h) files listed in each directory from the list.

*As sub-directories are ignored, so it is important to list all directories containing header files needed in the database using the ... button for selection then the Add button to insert the directory in the list.*

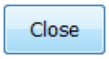
To **remove** one entry, just select it in the list then use Remove button.

Use 

to **save** in /Data/Resource the regenerated database and the directory listing, then to **exit** the utility.



to **save** in /Data/Resource the regenerated database and the directory listing.



will **exit** the utility **without saving** neither the database nor the directory listing.

# Copyright



**IMPORTANT - READ CAREFULLY**

This license statement and limited warranty constitutes a legal agreement ("License Agreement") between you ("Licensee", either as an individual or a single entity) and VirtualSim Inc. ("Vendor"), for the software product vsTASKER ("Software") of which VirtualSim Inc. is the copyright holder.

BY INSTALLING, COPYING, OR OTHERWISE USING THE SOFTWARE, YOU AGREE TO BE BOUND BY ALL OF THE TERMS AND CONDITIONS OF THE LICENSE AGREEMENT.
Upon your acceptance of the terms and conditions of the License Agreement, VirtualSim Inc. grants you the right to use the Software in the manner provided below.
If you do not accept the terms and conditions of the License Agreement, you are to promptly delete each and any copy of the Software from your computer(s).
The Vendor reserves the right to license the same Software to other individuals or entities under a different license agreement.

After accepting this license agreement, the Licensee is permitted to use the Software under the terms of this agreement for no more than the number of days allowed by the Evaluation license, without payment to the Vendor.
The Permanent license bears the name of the licensed person or entity, the computer LAN card number and is not transferable to any other party or computer. Pricing and availability is subject to change without prior notice. The Licensee can consult the most recent pricing information at sales@virtualsim.com.

The Software is provided "as is". In no event shall the Vendor or any of his affiliates be liable for any consequential, special, incidental or indirect damages of any kind arising out of the delivery, performance or use of this Software, to the maximum extent permitted by applicable law. While the Software has been developed with great care, it is not possible to warrant that the Software is error free. The Software is not designed or intended to be used in any activity that may cause personal injury, death or any other severe damage or loss.

When errors are found in the Software, the Vendor will release a new version of the Software that no longer contains those errors a reasonable amount of time after the Vendor is given an accurate description of those errors. Which amount of time is reasonable will depend on the complexity and severity of the errors. The Vendor will mention the release at http://www.virtualsim.com and, at the Vendor's option, directly contact the Licensee to announce the new release.
New releases are free to download for Licensees under valid Maintenance contract. Licensees that want to download the new release and who are not under Maintenance contract might pay either the full price of the new release or the Maintenance fees he would have paid from the time he bought the product until the date of the new release issue, whichever comes cheaper.

You must not attempt to reverse compile, modify, translate or disassemble the Software in whole or in part. You must not run the Software under a debugger or similar tool allowing you to inspect the inner workings of the Software.
The Software remains the exclusive property of the Vendor. Any Licensee which fully complies with the terms in this license agreement may use it according to the terms of this license agreement. You must not give copies of the Software or your license key to other persons or entities. You must not transfer the Software or your license key to another person or entity.

## Copyright