Lua Scripting

Building Better Behaviors in VR-Forces

VR-Forces developers and end users can write new tasks and set data requests using the Lua scripting language.

Lua is a programming language. It is widely used in the video game industry as the preferred way to extend game engines. Its popularity stems from its light-weight, fast nature and its ability to bind to C++ toolkits, such as VR-Forces. To learn more about the language please visit: http://www.lua.org/

What You Can Do with Lua in VR-Forces

You can write Lua scripts that define tasks and set data requests (sets) for simulation objects in VR-Forces. Prior to the availability of Lua scripting, all tasks and sets were developed in C++. Adding new tasks required C++ programming knowledge, a C++ development environment, and a VR-Forces development license. Additionally, for tasks or sets that required user input, developers had to design and implement a dialog box for receiving that input.

Using Lua scripting, anyone with sufficient programming skill can add a new task or set. VR-Forces builds a dialog box automatically. The scripts can be added to the Task and Set menus with all C++ tasks and the end user will not be able to tell the difference between a C++ and a Lua task.

Types of Scripts

VR-Forces supports the following types of scripts:

- Task. Scripted tasks can be listed on the Task menu and used as independent tasks or in plans just like C++ tasks.
- Reactive task. A reactive task monitors the simulation and gets activated when the conditions set in the script become true. Reactive tasks are enabled or disabled for simulation objects through the GUI.
- Background process. A background process runs continuously from the time a simulation object is created. There is no user interface for assigning background processes to simulation objects. Their use is based entirely on the scripting process.
- Set data request. Set data request scripts change simulation object state without interrupting the current task.

Benefits of Using Lua to Write Tasks

Use of Lua makes the task creation process significantly easier for several reasons:

- The process of developing a new task is more approachable. Task developers do not need to configure a large and complicated C++ environment. For example, you do not need to have a compiler installed when writing or maintaining a task. If your team has developed a custom task and you discover a small issue with it in the production system/lab, you can edit the task quickly through VR-Forces without having to set up (or go back to) your development environment.
- MAK developers use Lua scripting extensively to add new higher-level tasks. (Our developers enjoy the benefits of Lua scripting just as much as we believe that customers will.) We expect that some customers may want these tasks to work differently. Because the tasks are written in Lua and because the Lua scripts are distributed as part of simulation objects sets, customers can edit them to meet their specific needs. In essence, the tasks will be "open source" and highly accessible to you – much more accessible than the C++ API.
- Scripted tasking offers an approach to task development that is not possible with a compiled language. To develop a task in C++ you must:
  - Shut down VR-Forces.
  - Open a compiler.
  - Write the task, perhaps as a plug-in.
  - Compile and link the plug-in (which may take many minutes).
  - Start VR-Forces.
  - Open a scenario and test out the new task.
  - If the task does not work as expected, the whole process starts again.

  This process may seem tedious to you; it certainly seems that way to us. Because Lua is a run-time interpreted scripting language you can develop tasks while running VR-Forces and simultaneously developing a test scenario. You still need to write and test the task, but there is no requirement to open and close applications or compile anything. You can test your work in real time by just replaying your scenario. This approach may save you up to 70% of your time.

Experienced Programmers and Savvy Users can Write Scripts

Lua is a programming language and you must understand general programming concepts and Lua syntax to use it. It is much simpler than C++, but still requires a decent understanding of C-like syntax. We believe it will be much more accessible to programmers who may not be C++ experts.

Script Creation Basics

Let's take a high-level look at the steps in creating a scripted task. As an example we will imagine a task to refuel an aircraft. This example task would likely be assigned to a tanker object.

First, you set up metadata, which includes the task's name, its parameters, and whether or not it should be on the Task menu (Figure 1).

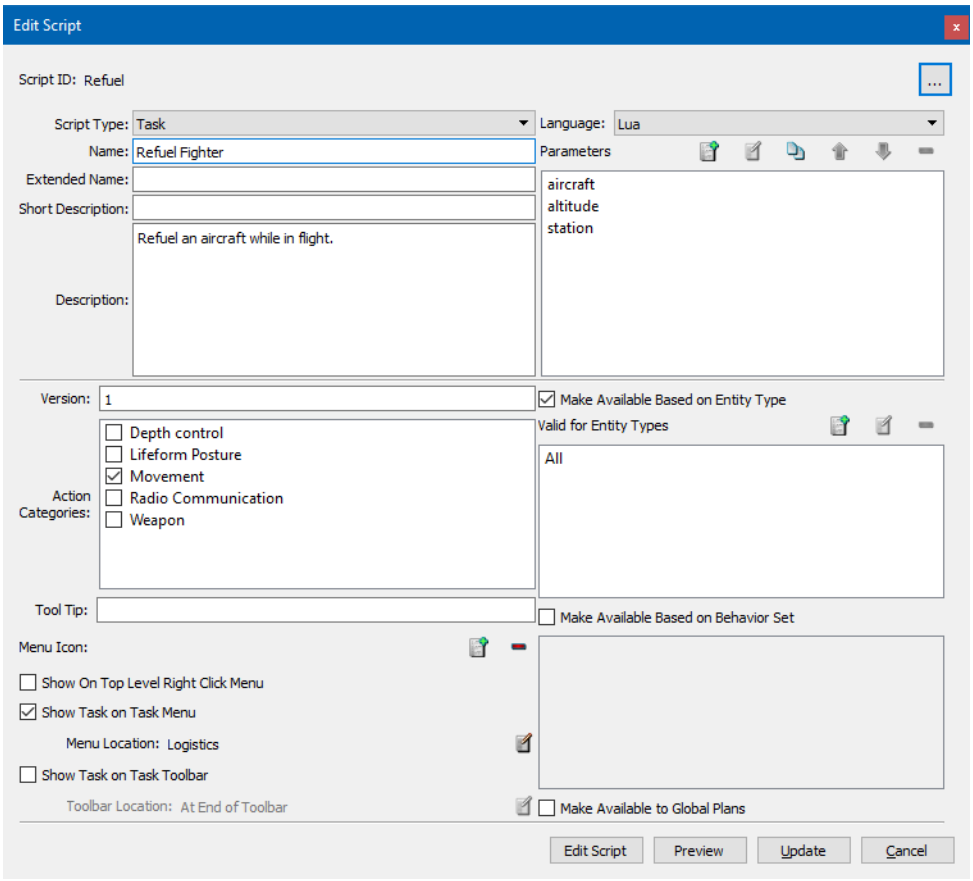

Figure 1 Script metadata

As part of specifying metadata, you define parameters (Figure 2).

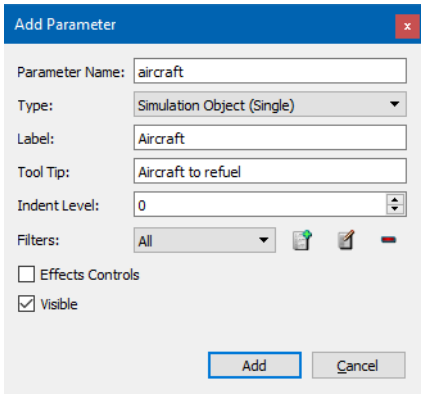

Figure 2 Parameter definition

Then you write the script. (This sample code is not written in Lua, but it represents the type of script you might write.) It is an extremely simple task that uses a sequential coding style, which is only suited for very basic tasks. More common LUA tasks will be tick() based and require a number of functions and likely much more code. Assume the following parameters:

- this - the tanker executing the task.
- aircraft - The aircraft being refueled.
- station - a location where the tanker orbits which includes Altitude

```
LastPosition = getLocation(aircraft);
-- Get into position. We need to be 1k meters flying north
FlyAltitudeHeading(1000,0));
SetOrderedSpeed(300); //300kph
-- tell the aircraft to tail me at 200m behind
SendRadioTask(aircraft, FlyFormation(this, 200, 180));

-- Assume it takes 1 min to refuel
Wait(60);
setAttributes(aircraft, "Fuel", 100); // set fuel to 100
Fuel = getAttributes(this,"Fuel");
Fuel = Fuel – X; //decrease my own fuel by X
setAttributes(this,"Fuel", Fuel);

// detach
currentLocation = this->location();

// Aircraft drops down and orbits were I left off
SendRadioTask(aircraft, HoldStation(LastPosition);

// Tanker Pops up and heads to station
HoldStation(station);
```

After you save the new task, you can access from the task menu (Figure 3).

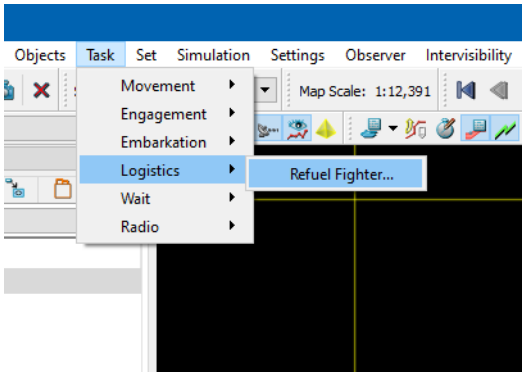

Figure 3 New task

Scripted Tasks Can Access Other Tasks

Within a scripted task, you can assign other scripted tasks and C++ tasks to the simulation object executing the task and to other simulation objects. Essentially, a task can now be a "mini-plan", which can be called from the entity's real plan. Or, you can start to address more complex tasks, like refueling multiple planes.

Assume the following parameters:

- this - the tanker refueling the aircraft.
- FighterList - a list of aircraft to refuel.
- Station - a location where the tanker orbits which includes altitude.

```
For Fighter in FighterList do; refuelFighter(this, Fighter, Station);
done;
```

When assigned, this task will refuel each fighter, sending each fighter back to their original location while the tanker starts the fueling at the Station point.

Using Frame-Based Architecture (the tick() Function)

LUA scripting supports sequential task creation and a tick() (for frame based) architecture. In fact, we believe most scripts will be written using the tick() method.

The sequential approach is highly limiting, but easy to do. Tasks execute a series of commands in sequential order:

```
execute A
execute B
execute C
```

This provides a logic that is extremely easy to follow, but produces a task which is very inflexible. What happens if the entity is attacked between A and B? What if the task is to refuel a plane that subsequently crashes before step 10 of a 20 step process? Should the tanker then continue with all the unfinished steps even though the plane is not present?

The tick() based approach allows developers to implement state machine-based tasks in which the existing condition of the world is examined before a switch to a new state is made. This means tasks are a bit more complicated to write, but also significantly more robust and powerful.

VR-Forces supports both of these approaches.

Units Can Control Subordinates

You can use scripted tasks to define the behavior of a unit. You would write a script for a specific entity in a leadership position. To this end, you would not be creating tasks for a unit but rather a single entity who could then be assigned subordinates that form a unit. The goal would be for a commander to issue subtasks to subordinates as part of execution for a larger task. For example, a team leader may task one subordinate to provide overwatch while tasking a second subordinate to advance to a specific position. When the movement is finished, the advanced subordinate provides overwatch while the first team member advances to the next position.

Using Scripts as an Alternative to Plans

Scripts can be as complex as you want. Since they have access to all supplied tasks and sets and since Lua fully supports variables and looping and conditional structures, a scripted task could take the place of writing a plan for any given simulation object. The robustness of Lua would provide planning options not available in the VR-Forces plan language.

While we plan to use Lua tasking to create robust tasks that can be used in multiple scenarios, you can also create a complicated plan specific to a scenario. Essentially, you would create a task called "Joe's Task" and then assign that entity only that task at the beginning of the scenario.

Scripted Tasks are Portable

By default, new tasks are specific to a scenario. You can promote them to be available to all scenarios that use a simulation model set. You can also export and import scripted tasks. This allows you to share a package of scripted tasks among multiple VR-Forces installations. You can also easily migrate them to new versions of VR-Forces.

Lua Scripting Does Not Replace the C++ API

The Lua scripting interface does not replace the C++ API. It supplements it. You can continue to use the C++ API for task development. Previously written C++ tasks will continue to work.

When you write new tasks, you can choose which API to use. How you write your tasks will depend on how complicated you want your task to be. With the C++ API, you have access to the full VR-Forces toolkit API. The VR-Forces C++ API is extensive and can be intimidating. It gives you the ultimate flexibility, which is frequently more than you need.

Writing in C++ means you can do more, but at the expense of a much more complex environment that requires you to recompile and relink every time you make a change.

You can also combine the two APIs. You can write a new task in C++ and then use the automatic dialog box creation feature of scripted tasks to build the dialog box for your new task.

MAK Developers Use Lua Scripting and C++ for New Tasks

Many of the new tasks added by MAK developers in recent releases of VR-Forces were written as Lua scripted tasks. We continue to use the C++ API for "primitive" or "low-level" tasks, which are computationally expensive, while we use Lua for higher level tasks. An example of a low level task would be Go to Position X, or Fire Weapon. A higher level task for Lua would be something like Provide Close Air Support at position X using approach Y.

Lua Scripting is Built-into VR-Forces

The Lua scripting feature is a part of the standard VR-Forces product. There are no additional files to install. You do not need to pay extra for it. You do not need a developer's license to use it. We include a text editor for writing scripts. However, you can use the editor of your choice.