RTC:Stage User Guide

Geoffrey Biggs
geoffrey.biggs@aist.go.jp

November 4, 2010

1 Introduction

RTC:Stage is an RT Component for the OpenRTM-aist middleware. It provides access to a simulated world being run by the Stage[1] simulator. All models in the world can be controlled and their data utilised by other components. The models that can be accessed outside of the component can be filtered to provide a smaller, more manageable component when using large simulations. Additional model proxies can be created and added to the component through a simple plugin system.

A key feature of the component is that its available ports change to match the world being simulated. If the simulated world contains two robots, separate ports will be available for accessing each robot. If multiple lasers are present in the world, multiple laser interfaces will be created. See the Section 5.1 for details on port naming.

This software is developed at the National Institute of Advanced Industrial Science and Technology. Approval number H22PRO-1167. The development was financially supported by the New Energy and Industrial Technology Development Organisation Project for Strategic Development of Advanced Robotics Elemental Technologies.

2 Requirements

RTC:Stage requires the C++ version of OpenRTM-aist-1.0.0 and the latest version of Stage from the Git repository[2]. At the time of writing, the most recent release of Stage does not contain several necessary API functions used by the component.

RTC:Stage uses the CMake build system[3]. You will need at least version 2.6 to be able to build the component.

RTC:Stage requires libltdl, part of GNU's libtool[4], for loading plugins.

Stage itself does not run on Windows. The component is therefore only available on Linux and MacOS X.

3 Installation

Follow these steps to install RTC:Stage:

1. Download the source, either from the repository or a source archive, and extract it somewhere.

   ```
   tar -xvzf rtcstage-1.0.0.tar.gz
   ```

[1] http://playerstage.sourceforge.net/
[2] http://github.com/rtv/Stage
[3] http://www.cmake.org/
[4] http://www.gnu.org/software/libtool/

2. Change to the directory containing the extracted source.

   ```
   cd rtcstage-1.0.0
   ```

3. Create a directory called "build":

   ```
   mkdir build
   ```

4. Change to that directory.

   ```
   cd build
   ```

5. Run cmake.

   ```
   cmake ../
   ```

6. If no errors occurred, run make.

   ```
   make
   ```

7. Finally, install the component. Ensure the necessary permissions to install into the chosen prefix are available.

   ```
   make install
   ```

8. The install destination can be changed by executing ccmake and changing the variable `CMAKE_INSTALL_PREFIX`.

   ```
   ccmake ../
   ```

The component is now ready for use. See the next section for instructions on configuring the component.

RTC:Stage can be launched in stand-alone mode by executing the `rtcstage_standalone` executable (installed into `${prefix}/bin`). Alternatively, `librtcstage.so` can be loaded into a manager, using the initialisation function `rtcstage_init`. This shared object can be found in `${prefix}/lib` or `${prefix}/lib64`.

4 Configuration

RTC:Stage has a distinct start-up process that is necessary to dynamically create ports to match the simulated world. This start-up process means that the configuration values must be specified in a configuration file, rather than through RTSystemEditor or similar tools.

To change the component's configuration, provide a file similar to the one shown below.

```
configuration.active_config: simple

conf.simple.world_file: /usr/local/share/stage/worlds/simple.world
conf.simple.gui_x: 640
conf.simple.gui_y: 480
conf.simple.limit_models:
```

It is possible to specify more than one configuration set in the file. The set used when the component initialises is specified on the first line. Give the file a suitable name, such as "stage.conf". Then add to rtc.conf the following line:

```
Simulation.RTC_Stage.config_file: stage.conf
```

The available configuration parameters are described in Table 1.

5 Ports

The ports provided by the component are dynamically created when the component is *initialised*. They are created to match the models available in the world.

Parameter	Effect
world_file	The path of the world file to load.
gui_x	The width of the window to display the simulation in. This option currently has no effect.
gui_y	The height of the window to display the simulation in. This option currently has no effect.
limit_models	The list of model filters. See Section 6.1 for details.
plugins	A list of paths to proxy plugins that should be loaded. See Section 6.2 for details.

Table 1: Available configuration parameters.

Model type	Model name	Port name
Robot	r0	r0_vel_control
Laser	r0.laser:0	r0_laser_0_ranges
Camera	r0.camera:1	r0_camera_1_image

Table 2: Examples of the port naming scheme.

5.1 Naming

The port names reflect the world, indicating which models they provide access to. For example, if the world contains a robot named "r0," a set of ports will be created providing access to its velocity control, odometry output, geometry service, and so on. These ports will all begin with the prefix "r0_". See Table 2 for examples of how the port names are created. Note that the special characters "." and ":" are replaced by underscores ("_").

6 Model proxies

The RTC:Stage component uses model proxies to provide access to the models contained in the simulated world. Each instance of a model in the world corresponds directly to an instance of a model proxy in the component. Several proxies are provided with the component. These cover the most popular models supported by Stage. They are described in Table 3.

In the event that the user wishes to use a model from Stage for which no proxy is provided, a plugin proxy can be written. See Section 6.2 for details.

6.1 Filtering models

When using a large simulation, the number of proxied models, and so the number of ports provided by the component, may become unmanageable. To counter this, the user can specify a set of model name filters in the component's configuration. Only those models who's names match one of the filters will have proxies created.

The filter format is a list of strings separated by commas. Each string is a filter. A model's name must match at least one filter for a proxy to be created. The wild card "*" can be used to specify flexible filters. The filter format is shown in Table 4.

For example, consider a simulation containing two robots, "r0" and "r1." "r0" has a laser scanner and a camera, while "r1" has two laser scanners. The simulated component would provide proxies for the following models:

- r0
- r0.camera:0
- r0.laser:0

Proxy	Ports	Data type	Port description
Actuator	vel_control	TimedDouble	Velocity control of the actuator.
	pos_control	TimedDouble	Position control of the actuator.
	state	ActArrayState	Current status of the actuator.
	current_vel	TimedDouble	Current velocity of the actuator.
	svc	GetGeometry2D	Get the pose and size of the actuator.
Camera	control	TimedPoint2D	Control over pan and tilt.
	image	CameraImage	Colour image captured by the camera, in RGBA.
	depth	CameraImage	Depth image captured by the camera, in 8-bit.
	svc	GetGeometry2D	Get the pose and size of the camera.
Fiducial	fiducials	Fiducials	List of currently-detected fiducials.
	svc	GetGeometry2D	Get the pose and size of the fiducial sensor.
Gripper	state	GripperState	Status of the gripper.
	svc	GetGeometry2D	Get the pose and size of the gripper.
		GripperControl	Open and close the gripper.
Laser	ranges	RangeData	Range values measured by the laser.
	intensities	IntensityData	Intensity values measured by the laser.
	svc	GetGeometry2D	Get the pose and size of the laser sensor.
Position	vel_control	TimedVelocity2D	Velocity control of the robot.
	pose_control	TimedPose2D	Pose control of the robot.
	current_vel	TimedVelocity2D	Current velocity of the robot.
	odometry	TimedPose2D	Value of the robot's odometry sensor.
	svc	GetGeometry2D	Get the pose and size of the robot.
		SetOdometry2D	Set the value of the odometry sensor.

Table 3: The proxies provided with RTC:Stage.

- r1
- r1.laser:0
- r1.laser:1

Without any filters, this would produce an instance of RTC:Stage with a large number of ports. If the user is only interested in a subset of the available models, specifying an appropriate set of filters will restrict the number of proxies created. Table 5 shows examples of which models will be proxied for different filter strings.

6.2 Proxy plugins

The Stage simulator supports writing model plugins. These provide additional functionality in the simulation, allowing new device types to be simulated easily without modifying Stage itself. Many robot developers may wish to implement new devices in this way for their work. Such models are not supported by default in RTC:Stage. To provide support, a proxy plugin that matches the model plugin must be created. Plugins can also be created for models built into Stage, such as the `ModelPosition` model, and new proxies can be created that over-ride the proxies included in RTC:Stage.

A proxy plugin provides an implementation of the `ModelProxy` interface. It must implement the abstract methods of this interface, and is responsible for adding any relevant ports to the RTC:Stage component. It is also responsible for moving data between these ports and the simulation.

In addition, proxy plugins must export two symbols:

Filter	Effect
`filter`	Match the entire model name.
`*filter`	Match at the end of the model name.
`filter*`	Match at the beginning of the model name.
`*filter*`	Match anywhere in the model name.
`filter1*,filter2*`	Two filters.

Table 4: The available filter formats.

Filter string	Created proxies
`r0`	r0
`r0*`	r0, r0.camera:0, r0.laser:0
`*camera:0`	r0.camera:0
`*:0`	r0.camera:0, r0.laser:0, r1.laser:0
`*laser*`	r0.laser:0, r1.laser:0, r1.laser:1
`r0.laser*,r1.laser*`	r0.laser:0, r1.laser:0, r1.laser:1
`r1,*laser:0`	r0.laser:0, r1, r1.laser:0, r1.laser:1

Table 5: The result of various filters on the created model proxies.

- `GetProxyType` - Returns the model type the plugin is for.
- `ProxyFactory` - Constructs an instance of the proxy.

To compile a proxy plugin, use the `BUILD_PROXY_PLUGIN` CMake macro, available in the `RTCStagePlugin` CMake file.

See the example plugins for more details on creating proxy plugins. Generally, copying an example and modifying it to meet the new model will allow rapid development.

6.3 Example plugins

Two example plugins are included with the component. They are installed with the component in `${prefix}/share/rtcstage/examples/`, where `${prefix}` is the location in which RTC:Stage was installed. The plugins can be compiled using CMake, e.g.:

1. `cd ${prefix}/share/rtcstage/examples/blobfinder/`
2. `mkdir build`
3. `cd build`
4. `cmake ../`
5. `make`

6.3.1 Blobfinder proxy

This example plugin provides a proxy to Stage's blob finder sensor model. It demonstrates using your own IDL in a plugin to provide the necessary data types.

6.3.2 Position proxy

This plugin demonstrates replacing the default position model proxy with a custom proxy. By loading the proxy provided by this plugin, the default position proxy is over-ridden in the component. The new proxy provides an alternate interface to the model using different data types.