Triana User Guide

The Triana Team

Contents

List of Figures

List of Tables

Chapter 1

Overview

1.1 Introduction

Triana is a graphical environment that allows you to create powerful computer programs and to use them, with a minimum of effort and no programming. Using Triana, you simply assemble your program from a set of building-blocks that you drag into a work-space window and connect up using your mouse. With a click of the mouse the program will perform whatever operations you want. You can tell Triana to execute your program just once or continuously, as long as data is available to it.

Since Triana is written in pure Java, it will run on almost any computer, and you can share your work with colleagues who work on different kinds of computers. Triana is a revolutionary new way to expand the number of things you can do on your computer. It allows you to create serious and complex programs without dealing with programming languages, compilers, debuggers, and error codes.

- Use Triana on a wide variety of data: numerical data, either taken from an experiment or generated by Triana; audio data; images; even text files.
- Triana comes with a wide variety of built-in tools. There is an extensive signal-analysis toolkit, an image-manipulation toolkit, a desk-top publishing toolkit, and many more. Most Triana tools will show you a parameter window, so you can adjust the way they work. Parameters can typically be changed dynamically, without interrupting the flow of data.
- Triana will display your data, either in a text-editor window or in a versatile

graph-display window. The grapher will display several curves at once and will allow you to zoom in on interesting features and mark them.

- Triana is particularly good at automating repetitive tasks, such as performing a find-and-replace on all the text files in a particular directory, or continuously monitoring the spectrum of data that comes from an experiment that runs for days or even years. If you run an experiment, Triana can help you save the cost of buying extra expensive oscilloscopes or spectrum analyzers: just let your laptop or lab PC do the job. If you maintain a web site you will find you can automate many of your tedious updating tasks. If you are a teacher, you can simplify the maintenance of student records and the grading of papers. If you regularly create reports, Triana will allow you to feed updated data directly into the finished document, no matter how it is formatted.
- Triana is a wonderful assistant in the classroom or teaching laboratory. It can help you to make demonstrations of analysis techniques, to generate simulated experimental data, to display graphs of complicated functions.
- If the tools supplied with Triana do not do what you need, Triana contains a wizard that helps you to create new ones, with parameter windows. Or you can use tools that are available on another web site, directly over the Internet, without having to copy them to your computer.
- Triana includes many features to ensure that your programs should work as you want them to. It checks data types and tells you if you have connected up tools that are not compatible. It gives you clear error messages if problems arise during the run and offers advice on how to avoid the situation. You can easily insert display units to monitor intermediate results and track down more subtle errors. Triana traps serious errors so the program will not crash; at worst you need only restart the calculation.

Triana is being developed by scientists at Cardiff University in the UK. They are working within the GEO600 gravitational wave experiment, a major physics collaboration between scientists from Germany, Britain, and other countries. GEO600 will generate many terabytes of numerical data each year, and Triana is designed to make it possible for scientists in the project to examine this data in a simple and versatile way. Triana is in use within GEO600 and in other major centers of research in the USA and Europe. The release version of Triana will take the same principles of easy operation and versatility, and extend them to a variety of types: image, sound, text, and numerical data.

But you don't have to be a rocket scientist to use Triana. Like the internet, email,

the web, and web browsers, Triana is another tool initially created for scientists that has very much wider use. Triana opens programming to people who don't know a programming language.

Chapter 2

Getting Started

2.1 Download, Installation and Configuration

Step 1 - Download
Download the latest Triana release from `http://www.trianacode.org`.

Step 2 - Set Environment Variable
Set an environment variable for your system, `$TRIANA` for Unix like systems or `%TRIANA%` for Windows, to point the top level `triana` directory, the location that you saved and unpacked the download to.

e.g. (from the command prompt)

`set TRIANA=C:\triana`	*Windows*
`setenv TRIANA=/home/user/triana`	*unix - tcsh*
`export TRIANA=/home/user/triana`	*unix - bash*

Step 3 - Build Triana
Run the Triana build script (`buildTriana`), which is located in the `triana/bin` directory.

e.g. (from the command prompt)

`C:\triana\bin\buildTriana.bat`	*Windows*
`/home/user/triana/bin/buildTriana`	*unix*

Step 4 - Run Triana
Run Triana using the `triana` script located in the `triana/bin` directory.

e.g. (from the command prompt)

`C:\triana\bin\triana.bat`	*Windows*
`/home/user/triana/bin/triana`	*unix*

2.2 The Triana User Interface

The Triana User Interface, figure 2.1 consists of a number of main components.

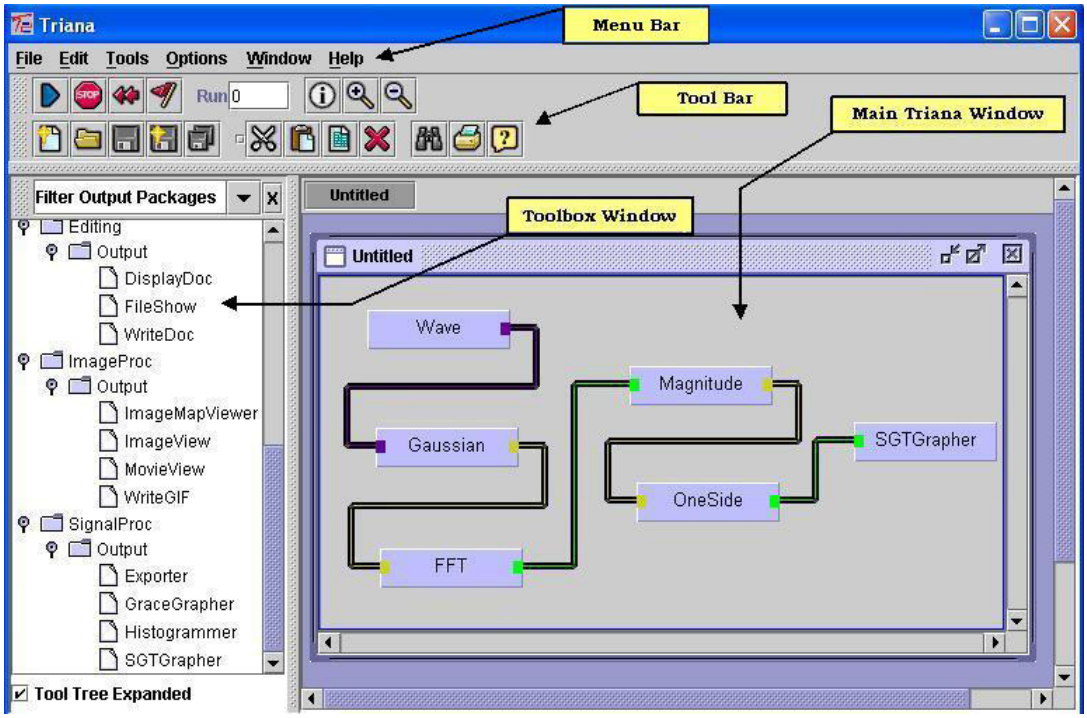


Figure 2.1: MainTriana User Interface

2.2.1 The Toolbox Window

The *Toolbox Window* allows you to explore the Triana toolboxes and navigate through the toolbox hierarchies. It is easy to view the toolbox contents simply by double clicking the toolbox name or the corresponding pointers. Contextual menus are available by "right clicking" on toolboxes or tools and a search entry allows for easy location of tools. The Triana *Toolbox Window* figure 2.2 consists

of a tree component with branches representing toolboxes and sub-toolboxes, leaf nodes representing tools or components.

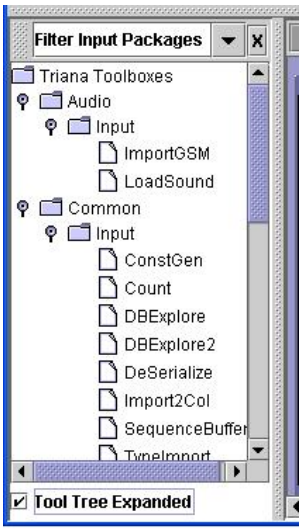


Figure 2.2: Toolbox Window

There are a number of different icons that represent different types of tools, these are can be seen in figure 2.3

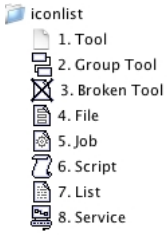


Figure 2.3: Tool Tree Icons

The icons represent different tools with toolboxes, the top "folder" icon represents a toolbox.

1. Standard tool.
2. Group tool, a number of other tools combined and saved as a new group.
3. Broken tool, a tool that cannot be instantiated on a main window. Typically because it has not been compiled.
4. File, a special tool that represents a *file* object, either locally or remotely.

5. Job, a special tool for representing a *job* object for job submission purposes.
6. Script, a special type of group tool that can act on other tools.
7. List, a representation of a list of objects.
8. Service, a proxy component to a remote service such as a web service.

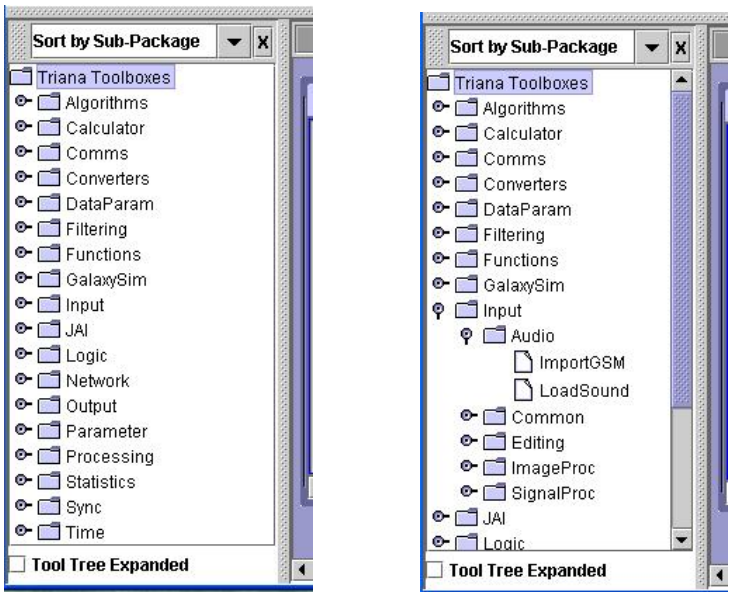
Searching

The combined search control at the top of the toolbox window has multiple functions. In text search mode, typing part or all of a component's name and hitting enter will search all of the toolbox paths for matches and display any matches in place of the normal tool tree.

The search control is also a "drop down list" with six items which are all tool filters for displaying a filtered view of the toolboxes:

- **All Packages (default)** returns the tool tree view to its default.
- **Sub-Packages** shows all the sub-packages of all the Triana toolboxes in alphabetical order as shown in figure 2.4(a). When clicked on a particular sub-package it points to the corresponding toolboxes (main packages that contains this sub-package). For example in the figure 2.4(b), we can see that the Input sub-package points to all the toolboxes that contain Input sub-package, such as the Input sub-package of the Audio toolbox/package.

- **All Tools** expands all toolboxes and sub-toolboxes so that all the tools or components are shown.
- **Input Tools** is a special form of **Sub-Packages** filter that shows all of the tools that are contained in any sub-toolbox called *Input*. This is a fast way of finding all tools that are capable of generating data for the start of a workflow.
- **Output Tools** is the corresponding filter that displays only tools from *Output* toolboxes. These are the tools that are designed to display or output the results of a workflow.
- **Data Type Tools** when selected gives you a popup menu as shown in figure 2.5 below. In this menu select the *Triana Data Types* by which to filter the toolbox view. Select the required data types from the menu and press the "Ok" button. Triana will filter the tools which contains those selected data types. Select **All Tools** to view the tree.



(a) Sub-packages (b) Expanded

Figure 2.4: Sorting by Sub-packages

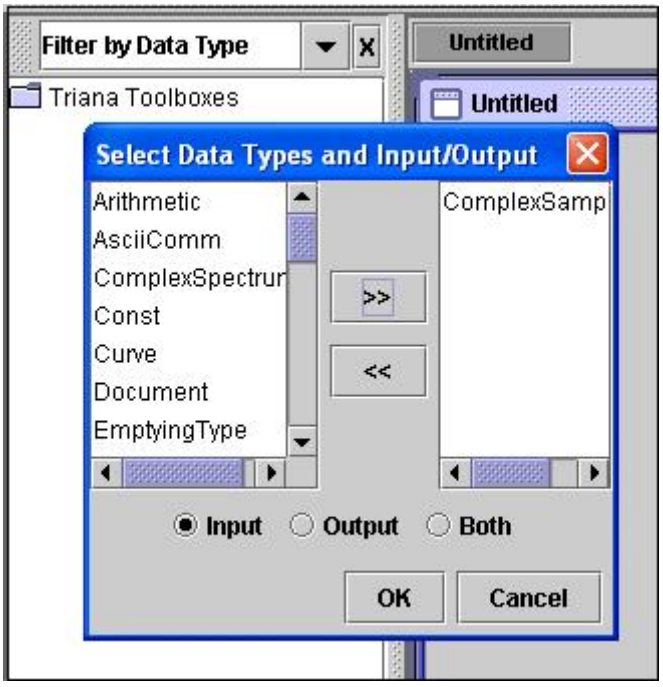


Figure 2.5: Filter Toolbox by Data Types

Toolbox Contextual Menus

The contextual menus for tools and toolboxes are accessed by "right clicking" (*control* clicking on Mac OS X) on the tool or toolbox icon in the toolbox window. The menus vary dependant upon the item chosen:

- All tools and toolboxes have a common set of editing functions in the menu, see table 2.1, these are explained in more detail in the section on editing toolboxes on page **??**.

Menu Item	Description
Cut	Cut the selected tools to the clipboard.
Copy	Copy the selected tools to the clipboard.
Paste	Paste the tools in the clipboard to the current toolbox.
Delete	Cut the selected tasks.
Rename	Rename the selected task.

Table 2.1: Common Contextual Tool Menu Items

- The pop-up menu for a single tool has in addition to the common editing functions the following menu items in table 2.2, see section 4.2 for a detailed description.

Menu Item	Description
Edit Tool Description	Edit the tool tip shown when the mouse is over this tool.
Edit Source	Edit the source code for this tool in the default text editor.
Edit HTML Help	Edit the help file in the default editor.
Edit XML Definition	Edit the XML tool definition file in the default editor.
Compile	Bring up the compile wizard for this tool.
Help	Show the help for this tool.

Table 2.2: Contextual Tool Menu Items

- Group tool menus are similar to a single tool menu except that the *Edit Source* menu item is removed and instead there is a *Open* menu item that will open the group in a new main Triana window.
- There are some specialist menus for *File*, *Job* and *List* tools that are discussed in chapter 3

2.2.2 Main Menus and Tool Bars

You can use Menus and Toolbar to give Triana instructions about what you want to do. The *Tool Bar* is a collection of frequently used commands or options that appear as two rows of tools.

The Main Menu

The main menu system, outlined in table 2.3, displays a list of commands.

Menu Group	Description
File	This is a standard menu group and most of the commands are provided in the Toolbar.
Edit	This menu group common commands like for selecting, grouping and ungrouping tasks.
Run	Contains start, stop and reset functions for constructed workflows.
Tools	This menu group allows you to make you own tools and also allows you to compile and generate XML file of you new tools.
Services	If Triana is running in a distributed mode (see chapter 3) then this menu provides the functions for discovering and creating remote services.
Options	Allows for configuration of Triana options (see section 2.6) such as *auto connect*, if you select this, your tasks dragged on the main triana window will be automatically connected by the cable.
Window	This adjusts the view of the main Triana windows or workspaces. *Cascade* or *tile* open Triana windows so that you can view them all.
Help	Under this menu group you can find comprehensive guidance for Triana. It includes reference to Triana's class packages, you can also search for usage and functions of all the Triana tools via find tools command and other tutorials to get started with Triana.

Table 2.3: Menu Commands

Some of the commands have images which indicate that they can be accessed

from the tool bar and short cut keys[1] which can be accessed anywhere. Many of the commands available from the menu can be also called with the help of popup menus directly from the main Triana Window (see section 2.2.3).

The Tool Bars

The main toolbar contains buttons with images, hovering the mouse over a tool bar item will display its function. The buttons are grouped by functionality and are outlined in table 2.4 looking left to right and top to bottom:

Group	Description
File group	*New*, *Open*, *Save*, *Save As*: these buttons work with taskgraphs to enable work to be saved to file and opened again.
Edit group	*Copy*, *Cut*, *Paste*, *Select All*, *Delete*: these buttons enable editing of tools in a taskgraph.
Help group	*Find*, *Print*, *Options*, *Help*: these buttons enable help searching, screen shot printing, options and help functions.
Run group	*Run*, *Run Recorded*, *Stop*, *Reset*, *Flush*: these buttons enable the execution and halting of the front most Triana Main Window.
Task group	*Show Properties*, *Group*, *Ungroup*: Show properties displays the parameter panel user interface for the tool in the main Triana window, group and ungroup combine all selected tools into a compound tool and back again.
Zoom group	*Zoom In*, *Zoom Out*: these buttons change the size of the task-graph in the main Triana window making everything smaller or larger for easier viewing.

Table 2.4: Tool Bar Commands

2.2.3 The Main Triana Window

The *Main Triana Window* is a workspace for constructing programs by dragging and dropping the tools from the toolboxes. Tasks are the components that can be graphically connected to create a particular data flow algorithm. The connection between tasks is made by dragging a cable from the output node (right-hand

[1]short cut keys are platform specific keyboard combinations that can be accessed without using the menu system. Cut, Copy and Paste are good example that use the standard keys for the operating system (*control C* for copy on Windows

side) of the sending task to the input node (left-hand side) of the receiving task. Once a network has been created it can be executed.

Main Triana Window Contextual Menus

Like most user interface components in Triana the main window for constructing workflows has contextual pop-up menus. These are accessed by "right clicking" (*control* clicking on Mac OS X) on the window. The menus vary dependant on where the mouse is clicked:

- Clicking on a tool icon will display the contextual tool menu contents in table 2.5:

Menu Item	Description
Properties	Display the parameter panel interface.
Node Editor	Display the node editor. Page 21
Run Continuously	Run this unit in continuous mode.
Auto Save History	Automatically save the history for the selected unit.
Save History	Save the history for the selected unit.
Create Service	Create a service from the selected unit.
Run Script	Run a script on the selected unit.
Cut	Cut the selected tasks to the clipboard. Page 14
Copy	Copy the selected tasks to the clipboard. Page 14
Delete	Cut the selected tasks. Page 14
Paste Into	Paste the tasks in the clipboard to the current window. Page 14
Rename	Rename the selected unit.
Help	Display the help file for the selected task. Page 28

Table 2.5: Main Window Tool Menu

- Clicking on multiple selected tools, section 2.4, will show a similar menu but with some additional items added and many of the previous ones removed, table 2.6.
- Clicking on a *group* tool, section 2.4, will show a similar menu but with some additional items added and some previous ones removed, table 2.7.
- Clicking on the window background will display the final menu, table 2.8

Menu Item	Description
Group	Turn the selected tasks into a new group task. Page 18
Cut	Cut the selected tasks to the clipboard. Page 14
Copy	Copy the selected tasks to the clipboard. Page 14
Paste Into	Paste the tasks into the tool tree. Page 14
Delete	Cut the selected tasks. Page 14

Table 2.6: Main Window Multiple Tool Menu

Menu Item	Description
View Group	Display the contents of this group in a new main Triana window. Page 18
Properties	Display the parameter panel interface.
Control Properties	Display the control properties dialog.
Ungroup	Return this group to its constituent parts. Page 18
Create Service	Create a service from the selected tasks.
Run Script	Run a script on the selected tasks.
Cut	Cut the selected tasks to the clipboard. Page 14
Copy	Copy the selected tasks to the clipboard. Page 14
Delete	Cut the selected tasks. Page 14
Paste Into	Paste the selected tasks into the tool tree. Page 14
Rename	Rename the selected unit.

Table 2.7: Main Window Group Tool Menu

Menu Item	Description
Group Editor	Display the group editor dialog. Page 18
Resolve Group Nodes	Checks for unconnected nodes in a group.
Save	Saves the current workflow.
Select All	Selects all tasks in the window. Page 14
Paste	Paste the tasks in the clipboard to the current window. Page 14

Table 2.8: Main Window Background Menu

Editing Workflows

A workflow on a main Triana window is not fixed. Just as if it were a document in a text editor the workflow, the connections and the components within it can all be edited. *Cut*, *Copy* and *Paste* commands all work and connections can be deleted and redrawn.

Selecting Tasks Clicking on the task once with the mouse in the main Triana window will highlight the particular icon. This means that it is selected. Multiple tasks can be selected by holding down the *control key* (*command* on OS X) and selecting the tasks or by "rubber banding"[2] the required tasks. You can do a number of things with selected tasks, for example:

- they can be deleted
- they can be moved
- they can be copied and pasted into the workspace (for multiple repetition of particular group of tasks)
- they can be copied and pasted into the tool tree (for creating stored copies of the task and its current parameters)
- they can be grouped into a composite task (group task)

Copying Tasks You can copy tasks by selecting the tasks you wish to copy and then choose the *Copy* option from either the main Triana window's popup menu (right-click on one of the tool icons) or the *Copy* option on the *Edit* menu. **TIP :** if you select several tasks for copying from the Main Triana window then the tasks will get copied but NOT the connections. If you wish to copy a selection of tasks along with the connections then make a group out of the tasks first and then copy the group.

Pasting Tasks Once a copy operation has taken place selecting *Paste* will paste a copy of the tasks to the current window. The *Paste* option can be found either in the window's popup menu (right-click on one of the tool icons) or by the *Paste* option on the *Edit* menu. If the tools are pasted into the tool tree, a stored version of the tool with the current parameters will be created. This new tool will appear in the tool tree and can be reused at a later date.

[2]rubber banding is the term given to dragging the mouse on a screen area which creates a thin box line around all of the items within the given area

Deleting Tasks You can delete tasks by selecting the tasks you wish to delete and then choose the *Delete* option from either the window's popup menu (right-click on one of the tool icons) or the *Delete* option on the *Edit* menu. Alternatively, to delete any single task use the Task's popup window (right-click on the Task's icon) and select the *Delete* option.

Moving Tasks You can move multiple tasks by selecting the tasks you wish to move and then whilst holding down the *control* or *command* key drag the tasks to where you want them to be on the window.

2.3 Your First Workflow

This section will demonstrate how to create and run the simple taskgraph shown in figure 2.6.

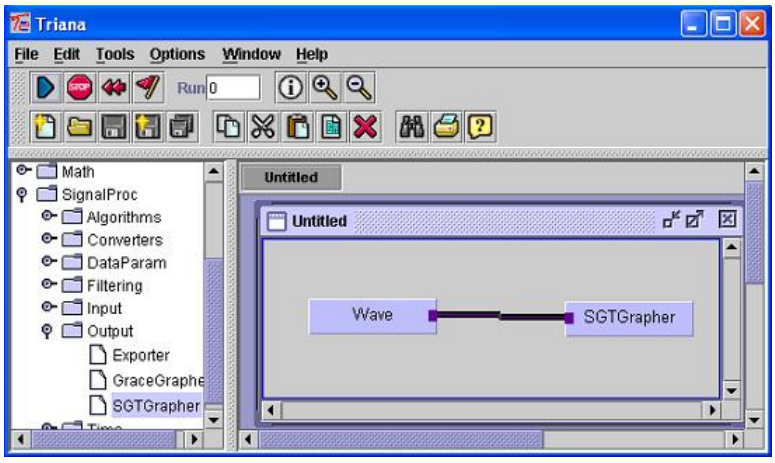


Figure 2.6: Simple Taskgraph

Lets take a closer look on how we make this connection by creating this network step by step.

1. In the tool tree, navigate to the *SignalProc* toolbox, open that toolbox by "double clicking" the toolbox or using the open branch control by the side of the toolbox, you should see all the sub-toolboxes within it. Now navigate to the *Input* toolbox and open that to reveal the tools.

2. Now select the *Wave* task and drag it from toolbox tree window and drop it on the left hand side of the main Triana window. After dropping the wave task in the Triana workspace it should look like figure 2.7

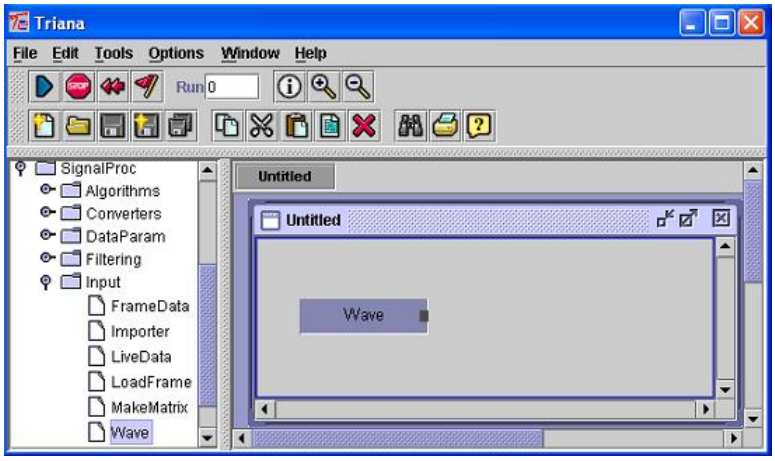


Figure 2.7: Instantiated Wave Tool

Note: you can move the tasks around the main Triana window in exactly the same way i.e. by dragging them to the new location.

3. Similarly, put the *SGTGrapher* task, a graph displayer, on the main Triana window. This task can be found in the *Output* toolbox within the *SignalProc* toolbox. Henceforwards written as *SignalProc/Output*. You should see the main window look as figure 2.8.

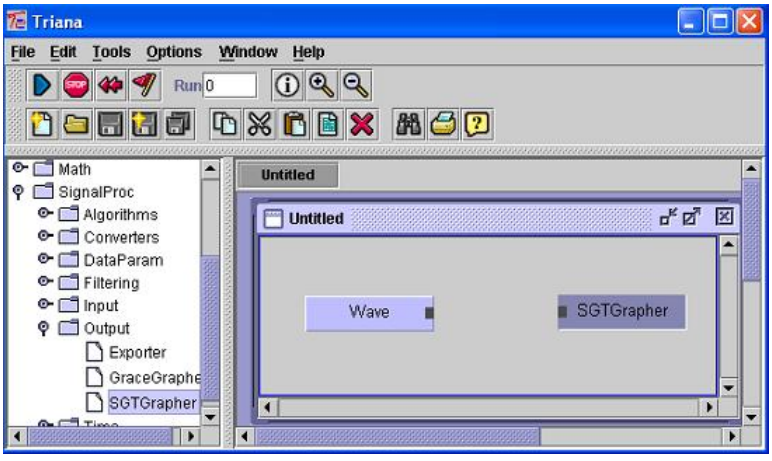


Figure 2.8: Instantiated Wave and Graph Tool

4. Now, connect the *Wave* task and the *SGTGrapher* task by dragging the cable from the output node of the Wave task and join it to the input node of the SGTGrapher. Which will give you the complete workflow shown in the first figure, figure 2.6

5. Press the *Run* tool bar button or the menu item *Run* under the *Run* menu. You should see small blue "execution indicators" flash on and then off each of the tasks. If you have a very fast computer these may flash on and off almost too quickly to see.

6. Display the *SGTGrapher* parameter panel user interface either by "double clicking" the tool in the main Triana window or using the contextual menu on the tool and selecting the *properties* item. The contextual menu is a pop-up menu that is accessed by "right clicking" on *Windows*^{TM} or Linux operating systems and "Command Clicking" on *Apple OS X*^{TM}, see section 2.2.3. The interface for *SGTGrapher* should look like figure 2.9.

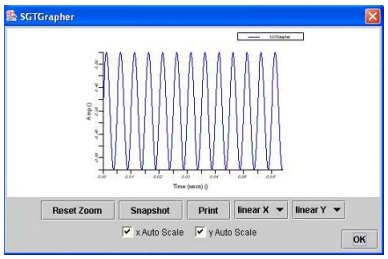


Figure 2.9: Grapher Interface

7. The form of the graph can be changed by changing the input parameter in the *Wave* task. Display the parameter panel interface by "double clicking" or using the *Properties* contextual menu on the Wave icon in the main Triana window. You should see the panel in figure 2.10.

8. Change the frequency to 800 hertz and click apply. Then, click on the run button and view the wave type in the SGTGrapher display, by repeating steps 5 and 6. You can view the graph display change as you change the frequency by moving the scroller, clicking apply and thenrun again.

Note: It is important that the *Apply* button is pressed after parameter values have been changed or the changes will not be set. An alternative is to select the *Auto Commit* check box at the bottom of a parameter panel. Checking this option forces changes to parameters to be set automatically and immediately this may have unexpected results for some workflows. Clicking the *OK* button applies all parameter changes and closes the parameter panel.

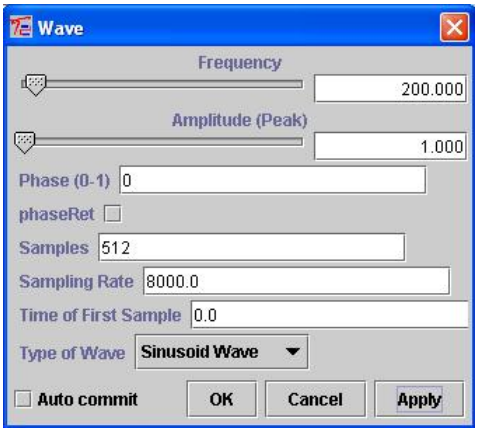


Figure 2.10: Grapher Interface

2.4 Grouping Tasks

In Triana, tasks can be grouped and then saved to toolbox files for later use. Grouped tasks are flexible because they appear as any other Triana task and can be used in the same way. Therefore, grouped tasks can be used to build new tasks from existing components without needing to write any java code!

Let us group the tasks from the network that was created in section 2.3, page 15, *"Your First Workflow"*. To demonstrate how we do this we shall convert the taskgraph in figure 2.6, page 15 into a group task called *WaveView*.

Lets take a closer look on how we do this step by step.

1. First Step is to select the tasks that are to be grouped. There are 3 ways different ways to select:
   (a) Groups of tasks can be selected by "rubber banding" the tasks. Click on the back ground of the main Triana window above and left of the tasks and drag a rubber banding area over the tasks. When the tasks are selected their colour turns slightly darker.
   (b) Bring up the popup menu of the main Triana window by clicking once on the empty space of the main Triana window with the right mouse button. And choose *Select All* option.
   (c) Select *Edit* from the Menu bar and choose *Select All* option.
2. Group the selected tasks either by selecting:
   (a) *Edit* option of the menu bar and choose the *Group* option.

(b) Press the Group tasks button in the tool bar.

(c) Use the contextual menu on one of the selected tools by "right clicking" or "command clicking" on the tool icon and selecting the *Group* option.

A *New Group* box will appear where you can enter any name for the new group. See figure 2.11

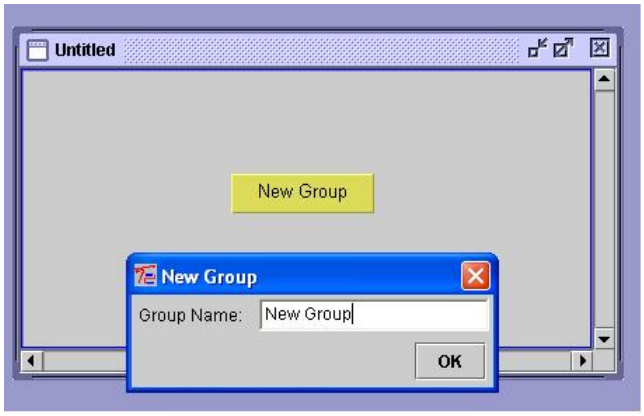


Figure 2.11: New Group Box

3. Enter the name *WaveView* and click OK. A new main Triana window will be created containing our new group. See figure 2.12 The name of the

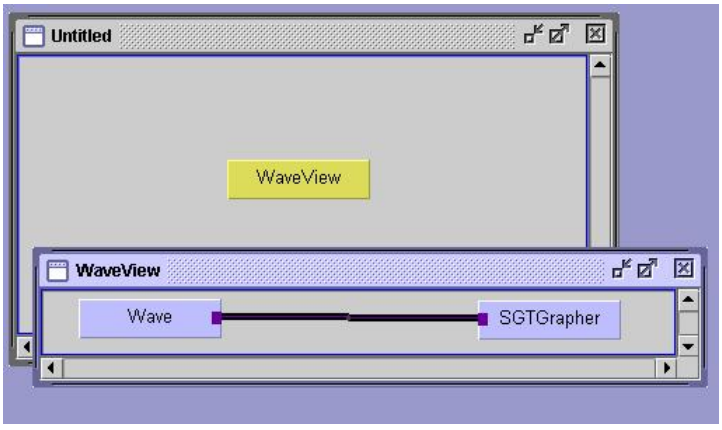


Figure 2.12: New Group Box

New Group is now changed to WaveView. Another window is showing the tasks which were grouped to make the new task WaveView.

Note: You can close the *WaveView* window showing the individual tasks, to reopen it select the WaveView task and right click the mouse and choose View Group.

4. Run the new group task by making sure its main Triana window is the front most and running in the normal manner. To see the result, double click or use the contextual menu on *WaveView* to bring up the parameter panel. The panel for group tasks presents each of the contained task's panels in a tabbed interface. Select each tab to see the user interface for each task. This can be seen in figure 2.13

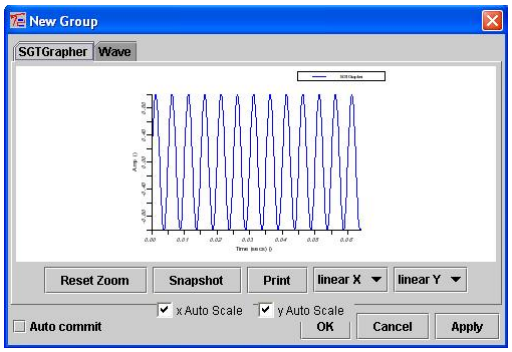


Figure 2.13: New Group Tool Parameter Window

2.5 Using Parameters Within Triana

Note: This section assumes that you have read and understood, section 2.3 on creating and running your first workflow.

Triana makes a distinction between *parameters* and *data*. Most components and workflows in Triana are data driven with producer components generating data objects that fow through communication channels to the consumer component next along in the workflow. A parameter is a special type of data message that is normally used for communication between a task and its parameter panel. In a distributed computing setting the user interface for a component may not be running on the same computer as the task so the communication between them in Triana is decoupled and based around messages.

Parameters can also be used by users to pass settings or information that are not strictly a generated data results from one component to another. In chapter 4 we will look at parameters in more detail and how they can be used by

unit programmers. This section will cover the use of parameters from a users perspective.

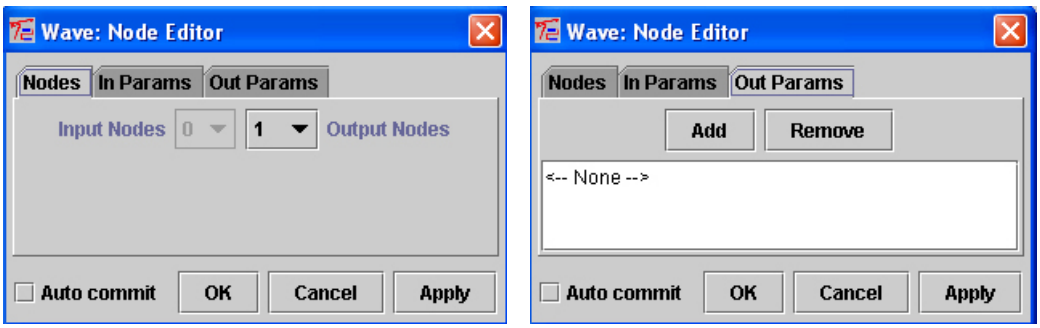
Parameters can be input or output from every task within Triana. Triana automatically detects all the parameters which a task uses and makes them available to the user. For example, let's demonstrate this by creating a network in which one *Wave* task controls another *Wave* task's frequency.

Note: In Triana parameters are output *after* the process function has been run (i.e. output after the data from the task) and parameters are input *before* the process function has been run. This means that you can send a parameter from a calculation in your process function of one task and the receiving task will receive this before it receives its data to process. This is extremely useful, for example, for re-constructing data sets when you need to know how beforehand how many you will want to concatenate.

2.5.1 The Node Editor

The Node editor in Triana is the mechanism by which the user can change the input and outputs to and from a task, subject to constraints placed by the task designer (see chapter 4). The editor is displayed by using the selecting the *Node Editor* option from a task's pop-up menu, page 12.

The user interface consists of three three tabbed windows, which can be see in figure 2.14. The first tab, figure 2.14(a), allows the user to change the number



(a) editing data nodes (b) editing output parameters

Figure 2.14: Node Editor

of input and output nodes for the task. In this case we can see that for the

Wave: Node Editor changing the number of input nodes has been disabled. This constraint has been set by the writer of the *Wave* tool. The user can change the number of outputs by selecting the desired value from the drop down list. Once *Apply* or *OK* has been pressed then the number of outputs on the task will be changed to reflect the new number.

Note: In Triana if the number of output nodes from a task is increased this means that when the task outputs its data, multiple copies of the data object are sent. One copy from each output node.

The second and third tabs, the third is shown in figure 2.14(b) allow the user to control a task's input and output parameters. By default no task parameters are output from or used as input to another task. However all of the parameters that the task programmer has used in the user interface, for instance, are available to use as parameters.

To add a parameter as an output to the task, select the *Out Params* tab, press the *Add* button and choose one of the listed parameters. See figure 2.15. Here the

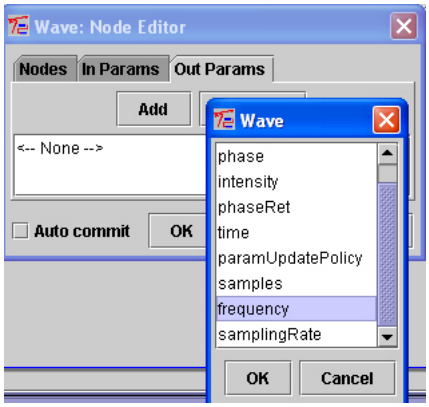


Figure 2.15: Parameter List Interface

user has selected the *frequency* output parameter for the *Wave* task. Adding an input parameter is exactly the same process using the *In Params* tab.

Note: As in the case of changing the number of input and output nodes. Access to parameters is constrained by the unit programmer. If a parameter does not appear in the list then it has not been made externally accessible. See chapter 4.

Node Editor Shortcuts

There is a short cut for increasing and decreasing the number of input and output data nodes for any task on a main Triana window. If you hover the mouse over the a task icon and the *Show Node Increase/Decrease Icons* setting is set, see page 26, then you should see + and − symbols flash on. Clicking on the left hand side symbols will increase or decrease the number of input nodes, clicking on the right hand side will do the same for output nodes.

Note: As with the *Node Editor* long hand method of changing the number of nodes this is only enabled where the unit programmer has specified it. For many tasks it does not make sense to allow multiple input nodes for instance.

2.5.2 An Example with Parameters

In this tutorial we will create the following network (the *Wave* tasks can be found in *SignalProc/Input* and the *Grapher* is in *SignalProc/Output*). The connection between the two *Wave* tasks is a parameter connection (see figure 2.16). The first *Wave* task is outputting a parameter and the second *Wave1* task is receiving this parameter. In this case the parameter is the frequency of the waveform.

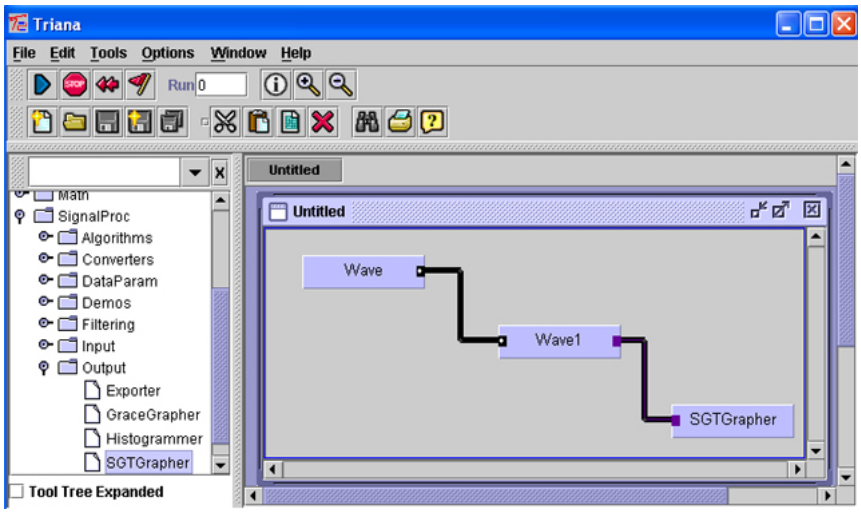


Figure 2.16: Parameter Connection

Let's take a closer look on how we make this connection by creating this network step by step.

1. Instantiate two instances of the *SignalProc/Input/Wave* tool onto a new main window by either dragging the tool from the toolbox twice or dragging one and then *copy* and *pasting*. See pages 15 and 14 for more information.
2. Select the first of the two *Wave* tasks and bring up the *Node Editor* by selecting the option from the pop-up menu, see the previous section on page 21.
3. In the first tab, *Nodes*, set the output node count to 0. We don't need to output any data from this task for this example.
4. Change to the third tab, *Out Params*. Click on *Add* button, this will allow you to choose the parameter which will be output from the first parameter output node on the *Wave* task. Select the *frequency* parameter and click *OK*. This will added a parameter node to the output of the *Wave* task. Parameter nodes are recognised by a white circle in the centre of the node as can be seen in figure 2.17.

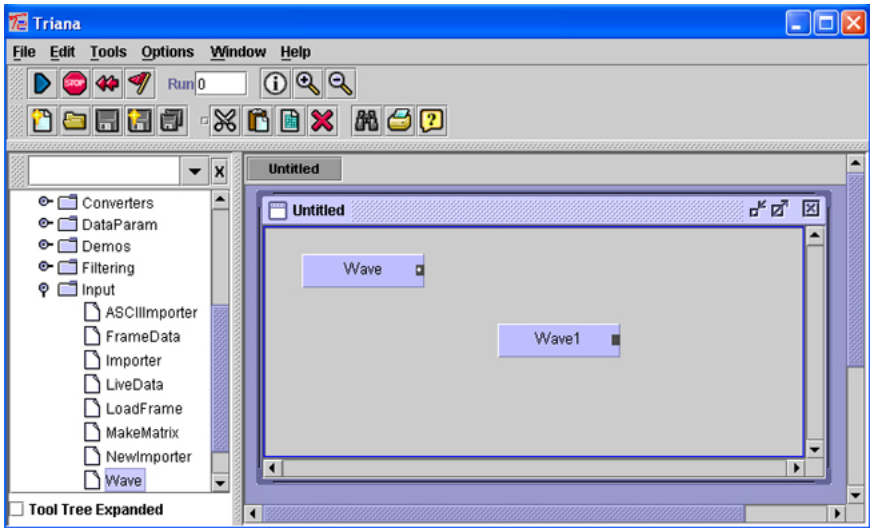


Figure 2.17: Parameter Node

5. Now, do the same for the *Wave1* task but in this case add an input parameter node. Go to the second tab *In Params* of the *Wave1 Node Editor*, click on *Add*, select *frequency* and click ok).

   Note: Choosingfrequency as an input node parameter in the *Wave1* task means that whatever we plug into this node will be controlling *Wave1*'s frequency. There is no type checking mechanism on parameters.

6. Now, make a connection between the output node of *Wave* and the input node of *Wave1* by dragging a cable between the nodes. Add a *SGTGrapher* task (in *SignalProc/Output* toolbox) and connect it to the output from *Wave1*. You should now have the workflow we first looked at in figure 2.16.

7. Now, show both parameter windows for *Wave* and *Wave1* (by double-clicking the task's icons) and then press the run the workflow. Move the scroller for the frequency in *Wave* task thereby changing its frequency and when you click on apply, you will notice the scroller for *Wave1*'s frequency also changes. *Wave* is remotely controlling *Wave1*'s frequency! Now when you run the workflow again you will notice that the frequency on the graph display will also change accordingly.

2.6 Configuring Triana Options

Triana is highly configurable and remembers settings from session to session. Window placement, window size and "zoom" level, see page 11, are saved when Triana quits and are recalled when Triana is restarted.

In addition to these automatic configurations some settings can be manually set. Under the *Options* main menu item there are two items. The first *Debug Window* displays the message logging window for Triana. For most users this can be ignored unless you are curious, it displays logging and error messages and is used mainly by tool programmers for debugging. See chapter 4. The second item *Triana Options* displays the options configuration window.

2.6.1 Options Configuration Window

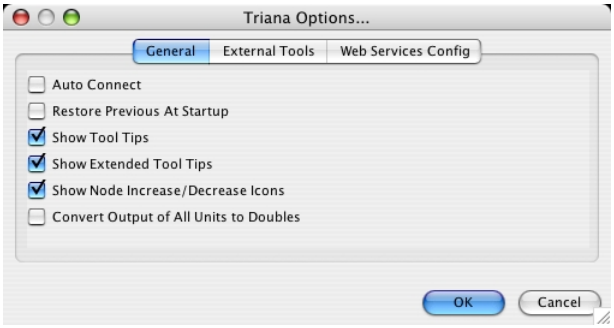


Figure 2.18: Options Window - general settings

The *Triana Options* window consists of three tabbed panes.

General Options

The general options settings which can be seen on the first tab of the *Triana Options* window, figure 2.18, has a set of "On/Off" check box settings.

Option	Description
Auto Connect	Automatically connects a new task dropped on a main window to a task already on the window. There is a smart algorithm behind this that attempts to connect to the correct task according to placement on the window.
Restore Previous At Startup	Automatically save and restore the working windows. Any main Triana windows with workflows will be saved at shutdown and restored the next time Triana is started.
Show Tool Tips	Display a short tool tip when the mouse is over a tool or task either in the toolbox window or on a main window.
Show Extended Tool Tips	Show a more detailed tool tip, includes package information, directory and more.
Show Node Increase/Decrease Icons	Show or hide the *Node Editor* short cuts.
Convert Output of All Units to Doubles	Triana's built in data types use *float* by default, this option forces them to use *double* instead. See chapter 4

Table 2.9: General Options

External Tools

The external tools options which can be seen in the second tab of the *Triana Options* window, figure 2.19, has a series of text fields with browse buttons and some additional controls. For most users the defaults set here should be suitable and there is no need to change them. If you are using Triana to develop tasks then some of these settings may be of interest.

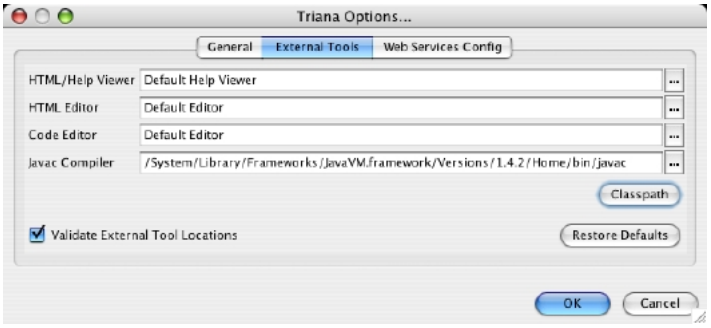


Figure 2.19: Options Window - external tools

HTML/Help View The tool used to display HTML and help files within Triana. The default is the internal HTML renderer, this can be changed by typing the path to a browser in the text field or pressing the browse button (...) and navigating to the executable.

HTML Editor The tool used to edit HTML and help files. The default is the built in text editor, this can be changed by typing or browsing to the path of another text editor.

Code Editor The tool used to edit unit source code. The default is the built in text editor, this can be changed by typing or browsing to the path of another text editor.

Javac Compiler The Java compiler used to compile unit source code within Triana. The default should be the system Javac compiler, this can be changed to a different compiler by the standard method.

Validate External Tool Locations A check box that when selected will check that any path in the external tool fields is a valid location. The path must point to a real program. This should be left checked unless there is a specific reason why not, i.e. you have deliberately added a non-existent compiler. If an invalid path is detected it must be corrected before the settings can be saved unless the box is unchecked.

Classpath A button which brings up the *Select Path* window, see figure 2.20. This window allows the user to configure the classpath used with the compiler inside Triana to compile a unit. It is only used by users wanting to compile their own units, see chapter 4. It provides the ability to add or remove and change the order of specific items on the classpath. There are options to automatically add all tool box paths, which will add every item on every classpath for every tool box to the list, and an item to *retain the classpath for future*. This saves the classpath when Triana has finished.

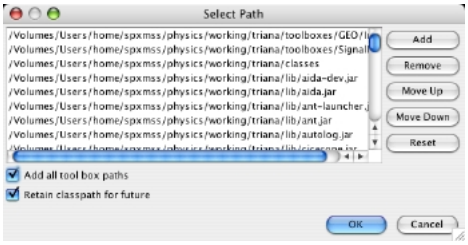


Figure 2.20: Selecting the Unit Classpath

Note: See the section on writing your own tools for more information on units and classpaths, specifically this classpath is for compilation only. For the run time classpath it is necessary to add any required libraries to the Triana classpath.

Restore Defaults A button that will restore all the Triana options to their factory settings.

Selecting *OK* will save all settings and close the window, returning to the normal Triana user interface.

2.7 Getting Help

Triana units are self documenting. You can access help on a particular Triana task by selecting the task in either the tool box window or a main Triana window and selecting *Help* by: pressing the *F1* function key; selecting from the contextual pop-up menu; selecting the tool bar button; or selecting *Help* from the *Help* main menu.

Chapter 3

Distributed Computing with Triana

In the previous chapters, the majority of discussion has concerned creating workflows using local Triana units, units where the processing is done on the same machine as the Triana application. However, unless the local machine happens to be a massive super-computer, such an approach is very limiting in terms of the computing power available to workflows. In an era of service orientated computing, such an approach also prevents workflows from employing any of the distributed services that are increasingly becoming available.

In this chapter we look at the facilities for utilizing distributed resources within Triana workflows. We give an overview of TrianaŠs architecture relating to distributed computing in Section 3.1, in particular noting the differences between the grid-oriented components and service-oriented components. In sections that follow sections we outline the current distributed computing bindings within Triana, namely Web Services (Section 3.2), P2PS (Section 3.3) and the GridLab GAT (Section 3.4).

3.1 Overview

Distributed components within Triana can be split into two categories, which we refer to as grid-oriented components and service-oriented components:

Service-Oriented - Service oriented components, such as web services and P2PS services, are network accessible components that exist on remote

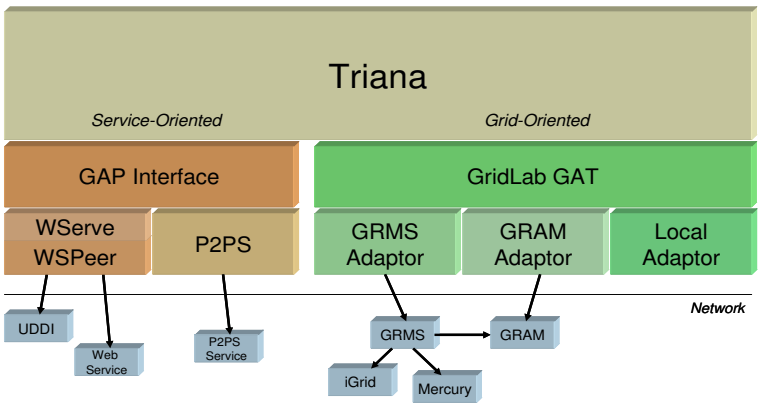


Figure 3.1: Distributed Component Middleware within Triana.

computing resources. These components can be discovered by Triana and then invoked from within a Triana workflow

Grid-Oriented - Grid oriented components represent jobs launched by Triana on a remote machine using a grid resource manager, e.g. GRAM[1] or GRMS[2]. Unlike service components, these jobs do not have network accessible interfaces so communication is only available via input/output files.

Triana uses different interfaces to access these two categories of distributed component. For service-oriented components the misnamed GAP Interface (Grid Application Prototype Interface) is used, while for grid-oriented components the GridLab GAT (Grid Application Toolkit) API is used. Both these interfaces have multiple bindings which allow different service/grid middleware to be employed without amending the Triana application code (as shown in Figure ref-fig:distarch). For service-oriented components, web services and P2PS services can currently be invoked, while for grid-oriented components, job submission can currently be done using GRMS, GRAM or the local adaptor. Both the GAP Interface and the Gridlab GAT allow new bindings to be plugged in when they become available. For example, when a Condor GAT adaptor becomes available Triana will automatically be able to utilize Condor job submission.

We outline the process for invoking web services in Section 3.2 and for P2PS services in Section 3.3, however, given that both invocations are done via the GAP Interface, there is a lot of commonality between the processes. We outline job submission via the GridLab GAT in Section 3.4; this covers using GRMS,

[1]see www.globus.org
[2]see www.gridlab.org

GRAM and Local adaptors.

3.2 Web Services

An important current paradigm in distributed computing is web services. Web services are remote software components accessible using standard network protocols and with defined XML-based interfaces. Already a large number of web service components are available via the Internet, and increasingly legacy applications are being wrapped in web service interfaces.

In Triana terms, web services function exactly as locally available tools. A web service receives input data, performs some operation on that data, and returns results. Due to this similarity, Triana allows the user use web services within a workflow as if they are standard tools. Once a web service has been discovered or imported (see Sections 3.2.2 and 3.2.3), it appears as a tool in the tool tree alongside the other tools and can be connected into a Triana workflow in exactly the same manner of other tools.

As well as discovery and importing web services, Triana allows the user to deploy a workflow subsection as a web service for other users (including other Triana users) to access. We discuss this more in Section 3.2.7.

3.2.1 Web Service Configuration

Web service configuration can be done using the *Services→Configure* menu option, and the selecting *Configure* next to the web service option. It should be noted that Triana automatically loads the last used web service configuration and therefore this configuration step needs only be done if you are changing the web service options, e.g. using a different UDDI repository. It should also be noted that once web services features (e.g. discovery) have been utilized the configuration cannot be changed within the current running Triana. This will be indicated by a disabled *Configure* option.

The web service configuration dialog (see Figure 3.2) is split into two panels, *Basic* and *Advanced*. The *Basic* panel enables the web service HTTP port and the UDDI repository to be set. The HTTP port is the port used by services created within Triana (see Section 3.2.7) only. If using this feature then this port must be open on the local firewall otherwise users will not be able to access the services. For simply invoking external web services not port need to be opened.

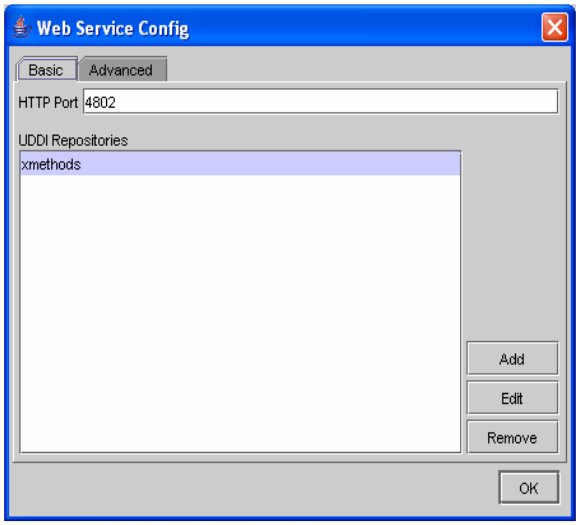


Figure 3.2: Web Service Configuration Dialog.

The UDDI repository is used by Triana to discover services (see Section 3.2.2. By default the XMethods (`www.xmethods.net`) is set, but additional repositories can be added. Note that when adding a UDDI repository only the inquiry address is required, the publish address is only required if deploying services within Triana, and a username/password are only needed if required by the UDDI instance.

To publish in some UDDI repositories trust stores are required. If this is the case then the location of the trust store file should be specified in the `config.xml` which is located in the `/.wspeer/admin` directory. Dummy trust store files can be found in the `triana/system` directory, these work for some UDDI instances.

The advanced panel allows additional properties to be configured, such as proxy addresses and security. Information on these properties may be found on the WSPeer website (`www.wspeer.org`).

3.2.2 Discovering Web Services

Once that a UDDI repository has been configured, web services can be discovered very simply using the *Services→Discover Services* menu option. This option displays a dialog prompting for the name of the service to search for. This can either be the exact name or can include % as a wildcard character. For example,

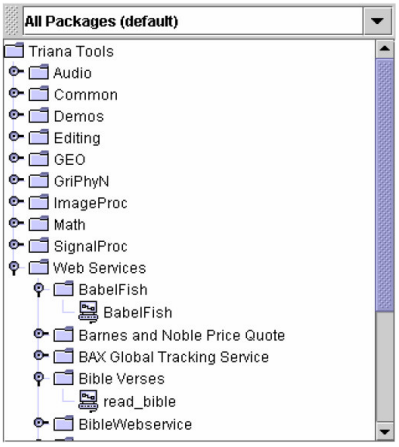


Figure 3.3: Web Service tools in the Tool Tree.

'C%' will search for all web services beginning with the letter C.

When a web service is discovered, tools representing each of the operations on that web service are inserted into the *Web Services* package in the tool tree, alongside the existing locally available tools, as shown in Figure 3.3.

3.2.3 Importing Web Services

If a web service is not listed in a UDDI repository, then it can be imported directly into Triana from its WSDL description. This is done through selecting the *Services→Import Service* menu option, which causes a dialog requesting the service location to be displayed. This service location is the full http address of the WSDL document specifying the web service to be imported[3]. The web service at the service location will be imported into the the *Web Services* package in the tool tree.

3.2.4 Conncecting Web Services

Once a web service has been discovered/imported and appears in the user's tool tree, it can be instantiated by dragging the tool onto the main workspace (in the same way locally available tools are instantiated). Web services tasks appear in red on the workspace.

[3]For example, an web service interface to Altavista's BabelFish language is located at `http://www.xmethods.net/sd/2001/BabelFishService.wsdl`

Each input node on an instantiated web service task represents an element of the input message for that operation, and each output node represents an output message element. Information on the required input/output types for a web service is displayed when the mouse is hovered over the task.

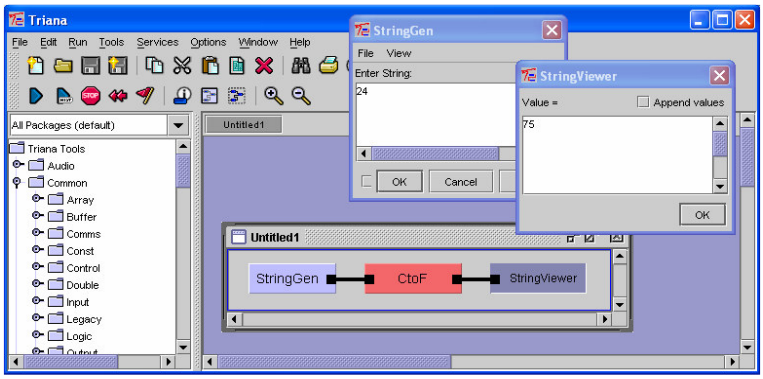


Figure 3.4: Using StringGen and StringViewer to provide input to and display the output from a temperature conversion web service.

Locally available tools can be used to provide the input to and display the output from web services. Two useful local tools are Common.String.StringGen and Common.String.StringViewer, which are used to generate string input and display string output respectively. These tools can also be used to input/output standard numerical data types (int, double etc.) as Triana automatically converts to/from the required type. In Figure 3.4 we show StringGen and StringViewer being used as input and output for a temperature conversion service. Other local tools can also be used as long as their output/input type is compatible with the type required by the web service, or alternatively the output from one web service can be directly piped to another. We look at handling complex data types in Section 3.2.6.

A workflow containing web services is executed as for a standard Triana workflow (i.e. by pressing the run button).

3.2.5 Bible Translation Example

In this section we demonstrate the creation of a simple bible translation workflow using third-party web services. This example uses the XMethods UDDI repository, so the following UDDI inquiry and publish addresses should be specified in the configuration (see Section 3.2.1):

Inquiry - `http://uddi.xmethods.net/inquire`
Publish - `https://uddi.xmethods.net/publish`

The two web services we wish to use are BabelFish[4], an interface to AltaVista's Babelfish service, and BibleVerses, a web service for extracting verses from the bible. The easiest way to import these web services is using the *Services→Disover Services* menu option, and then specifying 'B%' in the Discover Service dialog (this queries the UDDI for all tools beginning with B).

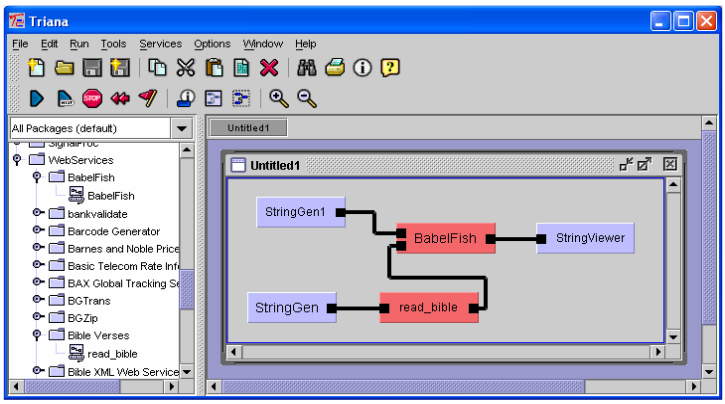


Figure 3.5: A simple bible translation workflow.

Once the BabelFish and BibleVerses web services have been discovered, they will appear as tools in the Web Services toolbox on the tool tree. Each of these services should be dragged onto the workspace, along with local StringGen and StringViewer tools, to create the workflow shown in Figure 3.5. StringGen and StringViewer are in the Common.Input and Common.Output packages respectively.

In this workflow StringGen provides the input for BibleVerses. If we double-click on StringGen and enter 'Genesis 1:1-7' in the input dialog, then this input will cause BibleVerses to extract the first seven verses from the bible (Genesis chapter 1, verses 1 to 7). The output from BibleVerses is used to provide the second input for BabelFish, which is the text that is translated. The first input to BabelFish is provided by StringGen1. This is the languages that the text should be translated from/to. Using StringGen1 to specify 'en_fr' indicates we wish to translate from English to French. The output from BabelFish is sent to StringViewer.

[4]Online documentation for BabelFish and BibleVerses can be found at `www.xmethods.net`.

Pressing the play icon on the tool bar will run the bible translation workflow we have created, and hopefully the 'Genesis 1:1-7' extract from the bible, translated into French, will be displayed in StringViewer (double-click on StringView view the result).

3.2.6 Complex Data Types

In Triana there are two ways to handle web services that require complex data types: use the dynamic web service type generator and viewer, or generate static type classes and create custom tools.

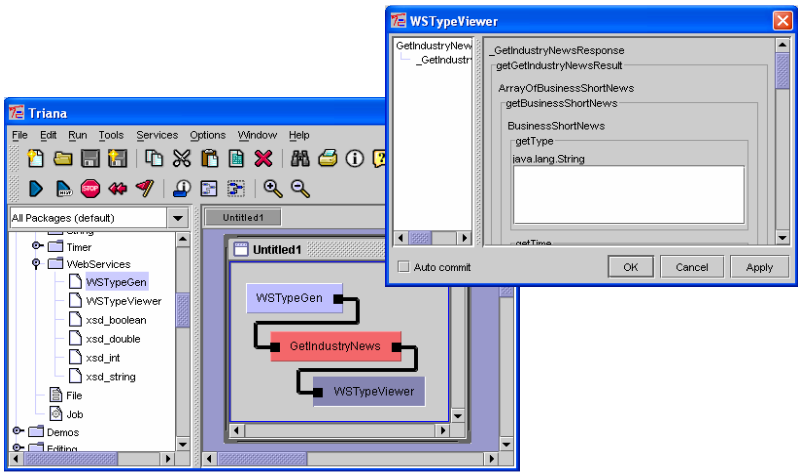


Figure 3.6: Generating/viewing complex data types using WSTypeGen and WSTypeViewer.

The input to a web service that requires a complex type can be automatically generated using the WSTypeGen tool, which resides in the Common.WebServices toolbox. Similarly, the complex output from a web service can be viewed using the WSTypeViewer tool, which is also found in the Common.WebServices toolbox. To use these two tools simply connect them to the web service task (as shown in Figure 3.6). The act of connecting them to the web service will cause a form for inputting/viewing the complex type to be dynamically generated. This form can be accessed by either double-clicking on the WSTypeGen/WSTypeViewer task, or by right-clicking and selecting *Properties* from the pop-up list.

Although WSTypeGen and WSTypeViewer are useful for testing web services that use complex types, for more long-term solutions it is generally required

that static classes are generated for the complex types. Static type classes can easily be generated using a utility such as Apache's WSDL2Java[5]. Utilities such as WSDL2Java parse the WSDL description of the web service and generate a set of Java classes for the web service and the types used within that service. It is easiest if these classes are created in the same location as the tools that will access them (same Java base directory).

Once static classes have been created for the complex types, custom Triana tools for populating the types with data or converting between types must be created. These tools should be able to access the complex type classes in the same way as for any Java class. We describe creating custom Triana tools in section 4.2.

3.2.7 Deploying Web Services

As well as using external web services, Triana provides the mechanism for deploying user workflows as fully functions web services. These deployed web services can either be used within a local Triana workflow, allowing some of the processing to be handled by a remote resource, or by third-party users. It should be noted that third-party users *do not* have to use Triana to access these web services.

In order to deploy web services Triana must first be configured with a UDDI to which you have both publish and inquiry access (see Section 3.2.1). This includes configuring the trust key store if required. Secondly, Triana launcher services must be run on the machine(s) that will host the deployed services. To do this Triana should be installed on those machines and the following command executed from the command-line:

```
TrianaService -ws
```

A web service can be created from either a single task or a group of tasks (see Section **??**). To deploy the task/group task as a web service right-click on the task and select *Create Service*. This will cause a dialog to appear with a list of the locations where the web service can be hosted (as shown in Figure 3.7). This list will include the available launcher services (see above) along with a 'Local Service' option. Creating a local service means that the web service is hosted within the local Triana as opposed to on a remote site. Note that it can take a while for Triana to discover the available launcher services from UDDI; they automatically appear in the list once discovered.

[5]See `http://ws.apache.org/axis/java/user-guide.html`

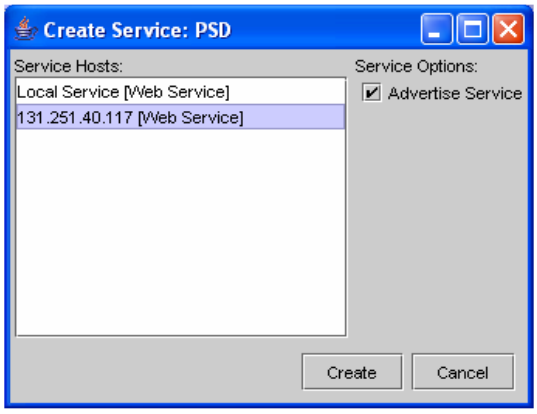


Figure 3.7: Dialog for deploying a task/group task as a web service.

Once a host has been chosen, Triana will attempt to deploy a web service on that host. While this process is taking place, the task in the workflow will appear with red stripes. The task becoming completely red indicates that the deployment has been successful and that the web service is ready to be used. As mentioned before, web services deployed using Triana can be used either within Triana or by third-party users.

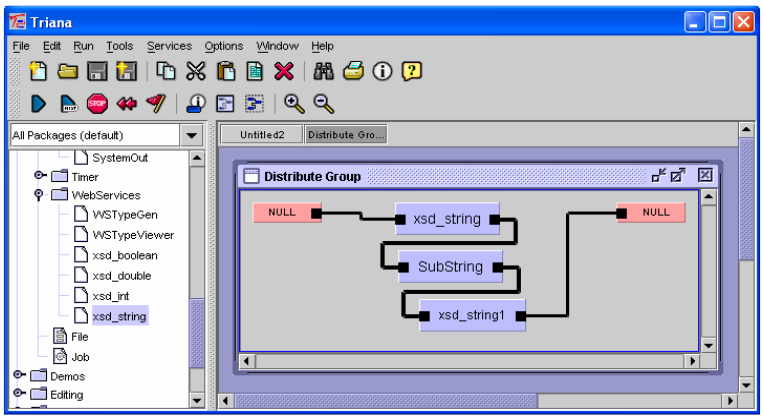


Figure 3.8: Using xsd tools to ensure standard input/output XML types are used when deploying a web service.

The input and output data types for deployed web services are automatically determined by Triana. If the input/output types from the task being distributed coincide with a standard XML type (e.g. java.lang.String, java.lang.Integer etc.) then that standard XML type is assigned. For non-standard types Triana assigns a string as the input/output type and assumes that the data received by the web

service will have been serialized to a string using JSX. Such web services can be seamlessly handled by Triana but will be difficult for third-party users to use. It is recommended that standard types are used whenever possible, and that the tools xsd_string, xsd_double etc. are appended to the start of the workflow subsection being distributed to ensure the correct standard type is used, as illustrated in Figure 3.8. These tools can be found in the Common.WebServices toolbox.

3.3 P2PS

P2PS (Peer-to-Peer Simplified) is a lightweight peer-to-peer infrastructure for dynamic service discovery and pipe-based communication. As, alongside web services, P2PS is one of the bindings to the GAP Interface (see Section 3.1) it can be used within Triana to discovery and communicate with remote services. It can also be used to dynamically deploy Triana workflow subsections as standalone services running on remote machines.

Unlike Web Services, where much of the benefit comes from utilizing existing remote services, the lightweight and dynamic nature of P2PS means the ability to deploy workflow subsections as remote services is particularly beneficial. For example, this mechanism can be used to distribute processing units to nodes in a cluster in a high-throughput task-farming application. We discuss deploying P2PS services further in Section 3.3.2. We also outline discovering existing P2PS services and connecting them into Triana workflows in Sections 3.3.3 and 3.3.4. However, first we describe configuring P2PS peers (Section 3.3.1).

3.3.1 P2PS Configuration

To configure P2PS select the *Services→Configure* menu option, and then the *Configure* option next to P2PS. Note that once P2PS has been used within the current running Triana it cannot be reconfigured and this option will be disabled. For discovering and communicating with services within your local subnet the default configuration should be sufficient.

For bridging between multiple subnets a rendezvous connection is required. The simplest way to do this is to enable Triana as a rendezvous peer and then have at least one service in the other subnet(s) enabled as a rendezvous peer pointing to Triana's local rendezvous port (see the Discovery Panel in the P2PS Configuration Dialog).

There are multiple different options that can be configured within P2PS, such as the transport protocols used and rendezvous policies, however discussion of these are beyond the scope of this manual. More information can be found at `www.p2psimplified.org`.

3.3.2 Deploying P2PS Services

As with web services (see Section 3.2.7), individual/group tasks within a Triana workflow can be deployed as P2PS services running on a remote machine. The mechanism to do this is exactly the same a for web services. Before deployment, first Triana launcher services must be started and configured on each remote machine that will host a P2PS service. This is done by installing Triana and the running the command:

```
TrianaService Űp2ps
```

To deploy an individual/group task, right-click on the task and select *Create Service*. This will cause a dialog to appear listing the locations where the remote service can be hosted. This list will include the locations where Triana launcher services are available plus a 'Local Service' option. The 'Local Service' option indicates that the remote service will be hosted within the current running Triana. Note that it can take a while for Triana to discover the available launcher services; they automatically appear in the list once discovered.

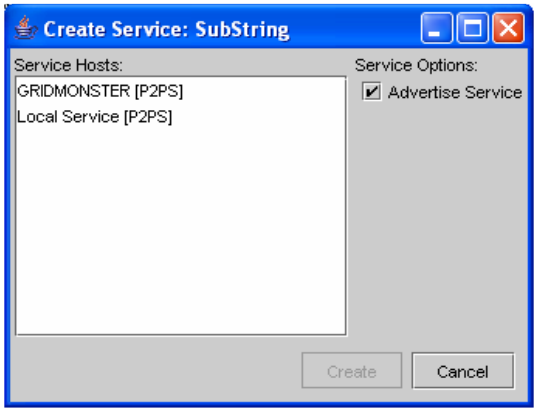


Figure 3.9: Dialog for deploying a task/group task as a P2PS service.

Once a host has been chosen, Triana will attempt to deploy a P2PS service on that host (as shown in Figure 3.9). While this process is taking place, the task in the workflow will appear with purple stripes. The task becoming completely

purple indicates that the deployment has been successful and that the P2PS service is ready to be used. P2PS services deployed using Triana can be used either within Triana or by third-party users.

Unlike Web Services, there is not a standard set of data types defined for P2PS. All data sent to and received from P2PS services deployed using Triana is serialized using JSX. This serialization should be seamless to Triana users and is only a consideration for third-party software using Triana deployed services.

3.3.3 Discovering P2PS Services

Existing P2PS services can be discovered very simply using the *Services→Discover Services* menu option. This option displays a dialog prompting for the name of the service to search for. With P2PS this search name has match exactly (ignoring case) the service name required.

When a P2PS service is discovered, a tool representing the operation provided by that service will appear in the tool tree alongside existing locally available tools. The P2PS service tool will appear in the same package as the original task used to create the service, however it will be displayed with a remote service icon (shown in Figure 2.3).

3.3.4 Connecting P2PS Services

Once a P2PS service has been discovered/imported and appears in the user's tool tree, it can be instantiated by dragging the tool onto the main workspace (in the same way locally available tools are instantiated). P2PS services tasks appear as purple on the workspace. P2PS service tasks can be connected to local tasks and other remote service tasks in exactly the same way as standard Triana tasks.

A workflow containing P2PS services is executed as for a standard Triana workflow (i.e. by pressing the run button).

3.4 GridLab GAT

The GridLab GAT[6] is a generic and flexible API for accessing Grid services, such as job submission and file movement. It has an adaptor based pluggable architecture (see Figure 3.1 that allows different service bindings to be utilized. For example, a GRAM adaptor allows job submission using Globus GRAM, while a GRMS adaptor allows job submission using GridLab GRMS. New adaptors can be developed and plugged in without changing application using the GAT (i.e. Triana).

Triana uses the GridLab GAT to allow job submission with a workflow, and also to transfer input files to and output files from a remote job. We give an example GAT workflow involving staging files and running a remote job in Section 3.4.1, and then descriptions of job submission and file transfer in Sections 3.4.3 and 3.4.4 respectively. We outline configuring the GridLab GAT in Section 3.4.2.

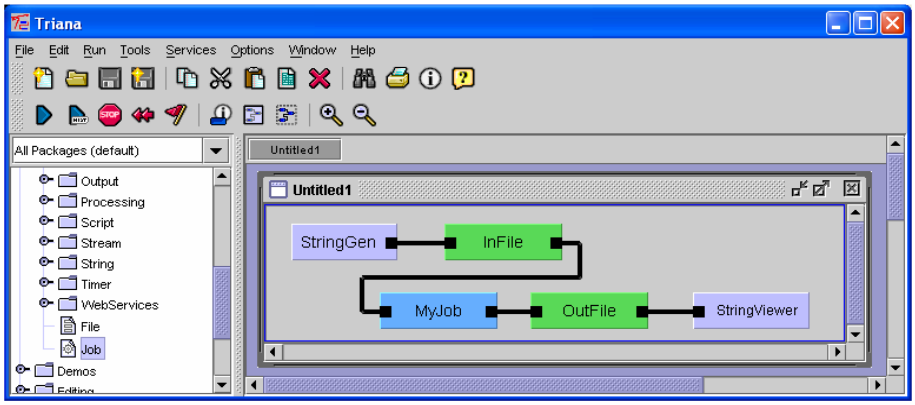


Figure 3.10: Example GAT workflow.

3.4.1 Example GAT Worfklow

In Figure reffig:gatexample, we show an simple example workflow for running a Grid job and transferring files to/from that job. This example workflow involves the following steps:

- Input generated by StringGen is written to the file InFile.

[6] see www.gridlab.org

- InFile is pre-staged at the location where MyJob is to be run.
- MyJob is run
- OutFile is post-staged from the location where MyJob was run
- OutFile is read and the output is viewed in StringViewer

The job (MyJob) run in this example could be any command-line application, and the input/output data (InFile/OutFile) can be any type of data. For example, MyJob could be a heavyweight numerical solver, InFile the parameter file specifying the parameters of the solver and OutFile the results from the solver.

Courtesy of using the GridLab GAT, this example workflow is the same regardless of whether MyJob is executed on the local machine or a remote grid resource (such as a super-computer). We discuss submitting jobs to grid resources further in Section 3.4.3.

3.4.2 GAT Configuration

The architecture of the GridLab Gat allows different adaptors to be plugged in for job submission/file transfer, and depending on the binding you intend to utilize different configuration is required. Currently the available adaptors for job submission are GRAM, GRMS and local, the configuration they require is as follows:

GRAM - Grid proxy initialized with valid Grid credential (e.g. using `grid-proxy-init` in Globus COG Kit[7]). Contact your local Grid administrator concerning acquiring a Grid certificate.

GRMS - Grid proxy initialized with valid Grid credential (same as GRAM).

Local - No configuration required.

The currently available adaptors for file transfer are GridFTP, FTP, HTTP and local, the configuration they require is as follows:

GridFTP - Grid proxy initialized with valid Grid credential (same as GRAM job submission).

FTP - Need to specify `username` and `password` in GAT configuration.

HTTP - No configuration required.

[7] See www.globus.org

Local - No configuration required.

3.4.3 Job Submission

Job submission in a Triana workflow is represented by a special *Job* component, a default version of which can be found in the Common toolbox. Dragging the default Job component into a workflow creates an empty job submission task (which by default is medium blue). Right-clicking on this task and the selecting *Job Properties* allows the submission details to be set.

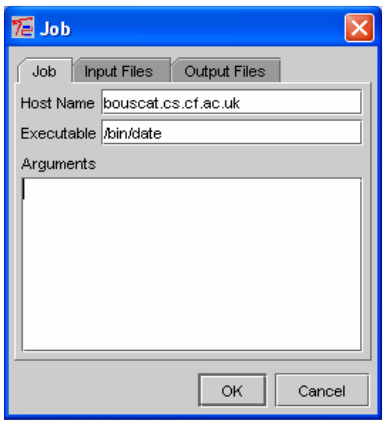


Figure 3.11: Job Properties Dialog.

The main panel of the Job Properties dialog (shown in Figure 3.11) allows the following submission properties to be set:

Host Name - Host name of the machine the job will execute on. Leave blank if running on the local machine or delegating to a Grid resource broker.

Executable - The job executable URI (e.g. /bin/date).

Arguments - The arguments for the job (if required).

To launch a basic job simply set the host name, executable and arguments in the Job Properties dialog, and then run the Triana workflow as normal. The job will be submitted to the appropriate resource manager (GRAM, GRMS, Local etc.) and executed on the specified host.

As well as detailing the job to be executed, the Job Properties dialog also allows the pre-staged (input) and post-staged (output) files for the job to be specified.

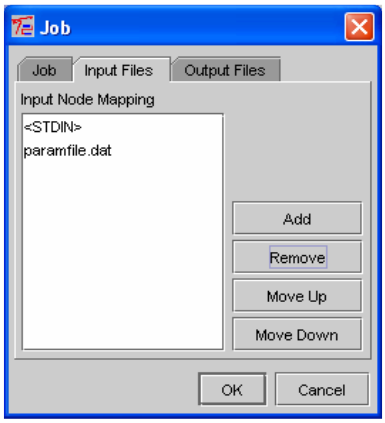


Figure 3.12: Inputs Panel in the Job Properties Dialog.

This is done using the *Inputs Panel* and the *Outputs Panel* in the Job Properties dialog.

The *Inputs* Panel in the Job Properties Dialog (see Figure 3.12) lists the files that are pre-staged in the job execution directory before the job is run. This list also represents the mapping between the input nodes on the job submission task and the pre-staged files, e.g. if the first item on the list is param.txt then the file at node 1 is pre-staged in the running directory as param.txt. Instead of the actual pre-staged file name, the tag <STDIN> indicates the file is piped to the standard input for the job. The tag <PRE_STAGED> indicates that the input file is pre-staged under its original name.

The Outputs Panel in the Job Properties Dialog is exactly the same as the Inputs Panel except that it specifies the files that are post staged-after job execution, and their mapping to the job task output nodes. As with the Inputs Panel, the actual name of the file in the job execution directory can be specified, or the tags <STDIN> and <STDERR> can be used to specify the post-staging of the standard input and standard error respectively. The tag <POST_STAGED> indicates the name of the file connected to the relevant output node is post-staged under its original name.

The number of input/output nodes on a job submission task is automatically set equal to the number of pre-staged and post-staged files. Connecting file components to these input and output nodes specifies the source location of the pre/post-staged files. This can be seen in the simple example shown in Figure 3.10. In this example, InFile is connected to the input node of MyJob, indicating that InFile is pre-staged before job execution, and OutFile is connected to the output node of MyJob, indicating that OutFile is post-staged after job

execution. We discuss file transfer using the GridLab GAT further in the next section.

3.4.4 File Transfer

As with job submission, files in a Triana workflow are represented by a special *File* component. A default version of the file component can be found in the Common toolbox (by default instances of this component are green). The file transfer operation associated with a file task instance depends on where it is connected into the workflow. For example:

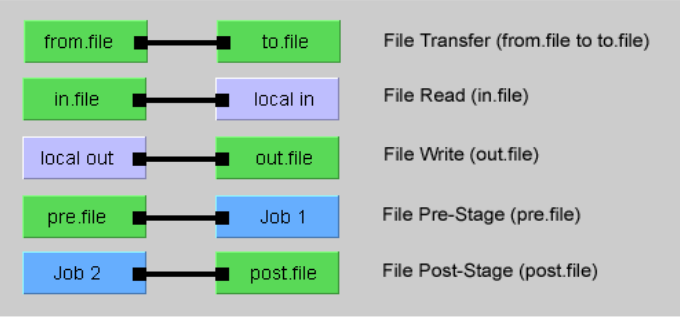


Figure 3.13: File Transfer Operations.

- Connection between two file tasks indicates a file transfer from the left file to the right file.
- Connection from a file task to a local task indicates a file read operation, with the input data being processed by the local task.
- Connection from a local task to a file task indicates a file write operation using the output data from the local task.
- Connection from a file task to a job submission task indicates pre-staging the file before job execution. The mapping between the file task and the file in the job execution directory is specified in Inputs Panel of the Job Properties dialog (see Section 3.4.3).
- Connection from a job submission task to a file task indicates post-staging the file after job execution. The mapping between the file in the job execution directory and the file task is specified in Outputs Panel of the Job Properties dialog (see Section 3.4.3).

Illustrations for each of these file transfer operations are shown in figure 3.13.

File tasks can be instantiated by dragging a file component (either the default file component in the Common toolbox or a custom file saved into the toolboxes) onto the workspace. Once instantiated, the file represented by the file task can be set through right-clicking on the task and selecting *Set File*. The URI of any file that can be handled by available GAT Adaptors (see Section 3.4.2) can be set. For example:

temp/myfile.dat - Local file in temp directory (relative to running directory).

/temp/myfile.dat - Local file in temp directory (relative to root directory).

file:///temp/myfile.dat - Same as temp/myfile.dat.

file:////temp/myfile.dat - Same as /temp/myfile.dat.

file://bouscat.cs.cf.ac.uk/temp/myfile.dat - File on remote machine in temp directory (relative to user home on remote machine).

file://bouscat.cs.cf.ac.uk/temp/myfile.dat - File on remote machine in temp directory (relative to root directory on remote machine).

file://bouscat.cs.cf.ac.uk//temp/myfile.dat - File on remote machine in temp directory (relative to root directory on remote machine).

ftp://bouscat.cs.cf.ac.uk/temp/myfile.dat - FTP accessible file on remote machine.

http://bouscat.cs.cf.ac.uk/temp/myfile.dat - HTTP accessible file on remote machine.

gsiftp://bouscat.cs.cf.ac.uk/temp/myfile.dat - GSI FTP accessible file on remote machine.

As can be seen from some of the examples above, it is valid to either express the protocol used to access the file explicitly (e.g. HTTP, FTP) or to leave it to the GAT to determine which protocol to use (e.g. when file scheme is specified).

Chapter 4

Extending Triana

Triana has been developed with extensibility in mind. From relatively simple extensions, such as writing your first unit, to major extensions, such as building alternative workflow language readers and writers or new GAP bindings, Triana has been designed to be extended.

This chapter deals with some of the possible extensions to Triana. The content here is often more technical, from a programming perspective, than the rest of this manual. A understanding of some technical matter such as the Java programming language is a prerequisite. This chapter is not intended to be a Java programming guide, there are many fine resources for that.

4.1 CVS Access

If you intend to do anything more than simple unit programming in Triana it is recommended that you use a developers version of the system from our CVS repository.

This section describes the steps in correctly checking out the source code for the core Triana and Triana toolboxes from the cvs repository and building from the source code. If you don't know how to use a CVS client or don't have a CVS account, then it is recommended that you follow the instructions in section 2.1.

4.1.1 Conventions

Note: These instructions assume you have the necessary user accounts and passwords. It is aimed at command line cvs users. WinCVS or similar users should be able to follow the instructions by using the repository and module names within their particular CVS Client.

- `Text written this font` should be typed as command line input. Commands should be entered without line breaks unless explicitly instructed. (This includes line breaks due to book formatting)
- *Text written in this font* and surrounded by < ... > should be replaced by the users appropriate details.
- **Text in this font** signifies the unix command line prompt, the text following it will be the command to type.

4.1.2 CVSRoot and Passwords

The "CVSROOT" you should you will depend on whether you have read only "pserver" access or write "ext" access. This document assumes "pserver", if you don't know then ask the person who gave you your user name and password.

For the Core Triana and default Toolbxes repositories there is anonymous "pserver" access, use the username <*anonymous*> with no password, just hit return at the password prompt.

4.1.3 Tagged, Stable and Unstable Versions

Please see the web site `http://www.trianacode.org` for the current branch tag names in the repositories. If you don't know what a tag is please see a good text book such as the CVS Book.

4.1.4 A Word About Directory Structure

For administration reasons Triana is split into a number of different packages which are stored in various CVS repositories. Some of these are project specific

and not public so don't assume that because something is listed in this document that you will be able to get the source code from the CVS server.

The public packages are:

1. Core Triana. The Triana Environment itself.
2. Toolboxes. The default toolboxes that come with Triana.
3. Toolboxes-dev. The unstable toolboxes that developers are currently working with. Use at you own risk.

The project specific packages are:

1. GEO. The Gravitational Waves tools.
2. Gravity. Example tools from the book "Gravity From The Ground Up" by Bernard Schutz.
3. GriPhyN. Tools from collaboration work with the GriPhyN project.

There are two standard ways to structure a Triana distribution from CVS, dependant on: whether you expect to be making regular changes to the source code under your control and/or you want to do frequent updates for the latest version from CVS; Or you just want to check out the latest version and build it once. CVS allows checking modules out inside other modules which will give the same structure as the packaged version of Triana but it will complain if an `update` or `commit` is attempted in the source tree where there is a foreign module lower down the tree.

4.1.5 Easy Install

These instructions will checkout and install Triana to the same structure as the packaged release version. It is fine for most users however if you expect to use CVS for more than updating small numbers of files it is suggested you use the other set of instructions.

This install will put all of the various toolboxes inside the Triana source tree and they will need to be removed to perform a `cvs update` on the core Triana code and then checked back out again.

From the command line:

prompt$ `cvs -d :pserver:`*<username>*`@trianacode.org:/home/cvsroot/triana login`
(Logging in to username@trianacode.org)
CVS password: *<userpassword>*

prompt$ `cvs -d :pserver:`*<username>*`@trianacode.org:/home/cvsroot/triana checkout -P triana`

prompt$ `cd triana`

prompt$ `cvs -d :pserver:`*<username>*`@trianacode.org:/home/cvsroot/trianatools login`
(Logging in to username@trianacode.org)
CVS password:] *<userpassword>*

prompt$ `cvs -d :pserver:`*<username>*`@trianacode.org:/home/cvsroot/trianatools checkout -P toolboxes`

Note: The rest of the Triana modules from CVS are optional and depend on project permissions to access.

prompt$ `cd toolboxes`

prompt$ `cvs -d :pserver:`*<username>*`@trianacode.org:/home/cvsroot/trianatools checkout -P toolboxes-dev`

prompt$ `cvs -d :pserver:`*<username>*`@trianacode.org:/home/cvsroot/geotools login`
(Logging in to username@trianacode.org)
CVS password:] *<userpassword>*

prompt$ `cvs -d :pserver:`*<username>*`@trianacode.org:/home/cvsroot/geotools checkout -P GEO`

prompt$ `cvs -d :pserver:`*<username>*`@trianacode.org:/home/cvsroot/gravity login`
(Logging in to username@trianacode.org)
CVS password:] *<userpassword>*

prompt$ `cvs -d :pserver:`*<username>*`@trianacode.org:/home/cvsroot/gravity checkout -P GravityFromTheGroundUp`

prompt$ `cvs -d :pserver:`*<username>*`@trianacode.org:/home/cvsroot/griphyntools login`

(Logging in to username@trianacode.org)
CVS password:] *<userpassword>*

prompt$ `cvs -d :pserver:`*<username>*`@trianacode.org:/home/cvsroot/griphyntools checkout -P GriPhyN`

After those CVS commands you should have a directory structure like this:

```
triana/
      /bin/
      /toolboxes/
                /Audio
                /... other standard toolboxes
                /toolboxes-dev
                /GEO
                /GravityFromTheGroundUp/
                /GriPhyN
```

The toolbox structure is the import thing, there will be extra directories under `triana/` not mentioned here.

Build and Run

To build set an environment variable for your system, `$TRIANA` for Unix like systems or `%TRIANA%` for Windows, to point the top level `triana` directory. Run the build script, `buildTriana` for Unix or `buildTriana.bat` for Windows:

prompt$ `$TRIANA/bin/buildTriana`

Run the start script, `triana` or `triana.bat`:

prompt$ `$TRIANA/bin/triana`

4.1.6 Developer Install

If you are intending to write or modify any code within the various CVS modules or you just want to regularly update the modules from CVS. Then we suggest that you keep all the CVS modules in their own separate directory stuctures. The *Ant*[1] build file is written to be able to build from default either the previous structure or a directory structure where all the cvs modules are at the same level

[1] `http://ant.apache.org/`

in the file system.

For instance my directory structure looks like this:

```
project/triana/
project/toolboxes/
project/toolboxes-dev/
project/toolboxes_other
project/toolboxes_other/GEO
project/toolboxes_other/GravityFromTheGroundUp
project/toolboxes_other/GriPhyN
```

So from a sensible starting directory run the CVS commands from section 4.1.5 with the `cd` commands so that the Triana core and default toolboxes modules from cvs reside in your current directory.

Note: The GEO, Gravity and Griphyn toolboxes should really reside in a separate subdirectory. Here I've created a directory called "toolboxes_other", this is because Triana assumes that a toolbox contains packages, so GEO for instance is the top level package for the GEO tools. When Triana attempts to locate tools it uses the following path :-

[toolbox][tool package][toolname]*

e.g. `[/project/toolboxes/][SignalProc/][Input/][Wave]`
or `[/project/toolboxes_other/][GEO/][Algorithms/][SaturationMon]`

The build and run instructions are almost the same as for the easy install.

- Set the `TRIANA` environment variable to point to the `triana` directory.
- Either edit the build file (build.xml in the main Triana home directory) and set the appropriate toolbox location variables in the "User Editable" section, or the preferred method add a new file called "build.properties" to the Triana home directory with the properties and their values. The contents of my "build.properties" file can be seen in figure 4.1
- Run the `buildTriana` script.

```
javac.flag.debug=on
javac.flag.deprecation=on
triana.geo.tools=../toolboxes_other/GEO
triana.gravity.tools=../toolboxes_other/ \
    GravityFromTheGroundUp
triana.griphyn.tools=../toolboxes_other/GriPhyN}
```

Figure 4.1: example *build.properties* file

Note: Unlike the standard installation, Triana will not automatically pick up the toolboxes when it is started so you will have to add those within Triana from the menu *Tools, Edit Tool Box Paths*. In my case, I select the directories - `/project/toolboxes/`, `/project/toolboxes-dev/`, `/project/toolboxes_other/`.

4.1.7 Other Install

You can actually have your toolbox modules anywhere you like in your file system and still have the build process pick them up and compile them. In the file `build.xml` there is a commented section titled "USER EDITABLE PROPERTIES" within this section there are a number of commented out properties for the various toolbox modules. Uncomment any of these and set them to the appropriate location to override the default locations. Alternatively add a file called "build.properties" with the appropriate lines containing - `propertyname=value`.

4.2 Writing Your Own Tools

Writing tools in Triana can be started very easily. The only prerequisite is a little understanding of the *Java* programming language, working with files and directories in the operating system of your choice and the desire to experiment.

4.2.1 Toolbox Structure

Toolboxes in Triana are, as you might expect, just a series of directories and files within a set structure. However the relationship between the underlying file structure and the tree representation in the toolbox window is not necessarily

that simple. The logical (tree view) and the physical (file view) of the toolbox structure does not have to be the same although it often is.

Toolbox structure starts with a toolbox *root* directory. The toolbox root is the directory that Triana looks in recursively for toolboxes. Triana can, and in fact does, have more than one toolbox root they are a useful way of compartmentalising different groups of tools. If you are using the default Triana download then your *default* toolbox root will be `$TRIANA/toolboxes` (for unix) or `%TRIANA%\toolboxes\` (for windows). See the next section, 4.2.2, for more information about adding or changing toolbox roots.

Underneath the toolbox root are packages, represented by directories containing other directories, and toolboxes, represented by directories containing tools. The whole process is recursive so package hierarchies can be to any depth and a toolbox can contain sub-packages as well as tools. Triana searches the directory structure looking for tool files and then builds up the tree from leaf node (tool) back to toolbox root.

A tool box directory contains an XML component definition file for each tool, a `src` directory that contains any source code files for the tool, a `help` directory that contains any help files and an optional `lib` directory that typically holds *shared library* objects for specific platforms.

All of the directory structure and files can be created by Triana and for most users that is how they should be created. In section 4.2.2 we will discuss how to create a new toolbox and in section 4.2.3 we will populate that toolbox with a new tool.

4.2.2 Creating a New Toolbox

There are a number of ways to create a new toolbox but first so that we don't disturb the existing tools and toolboxes within Triana we will create a *new toolbox root*. The new toolbox root will be used to hold all of our new toolboxes and tools, that way they can be kept separate from the existing stable tools if we need to update the Triana version at a later date.

To change or add toolbox roots to Triana select the *Tools* main menu item and then the *Edit Tool Box Paths* item. You should see the window in figure 4.2.

The paths will obviously look slightly different to this depending on you set up and operating system, the one shown is a unix system. In the default setting shown here there are already four toolbox roots set and each of these has a

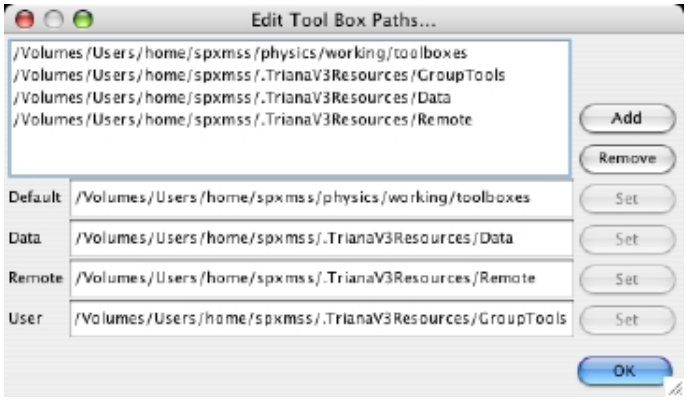


Figure 4.2: Edit Toolbox Paths

different purpose. The list of toolbox roots is presented in the top panel on the dialog with *Add* and *Remove* buttons which add or remove toolbox roots from that list.

Note: There is no way to edit an existing toolbox root, it should be removed and a new one added.

The four fields below the list, labelled *Default*, *Data*, *Remote* and *User* and the four default toolboxes.

Default is the standard toolbox root where Triana stores all of the normal included tools.

Data is a specific toolbox root for *data* tools such as files.

Remote is the toolbox root where Triana stores the component definition files for any *remote* tools such as P2PS services or web services.

User is the toolbox root that Triana uses to store *user* specific tools such as new group tools by default.

Note: Apart from the *default* toolbox root which will normally reside inside the Triana home directory, all of the other three toolbox roots are stored in a special hidden directory in the users home directory called `.TrianaV3Resources`. This directory is explained in more detail in the "trouble shooting" section on page 68.

Now to add a new toolbox root all we need to do is to select the *Add* button and use the file browser to navigate to a suitable empty directory or create a new directory. Select *OK* and the new path has been added to the list of tool box roots. We are now ready to create a new tool and toolbox structure within that

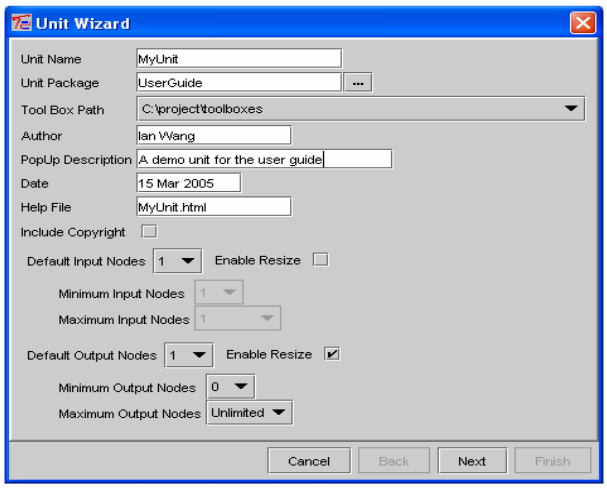


Figure 4.3: Unit Panel in the Unit Wizard.

root path. To do that we will use the *Tool Wizard* explained in the next section, section 4.2.3.

4.2.3 Using the Unit Wizard

The Unit Wizard provides a simple GUI for generating skeleton Triana unit code. Once generated only the `process()` method requires implementation in order to produce a Triana unit. Note that this unit must then be compiled and the tool XML file generated before it can be used within Triana. We discuss compiling tools and generating tool XML is Section 4.2.4.

The Unit Wizard is located in *Tools→New Unit* menu. When selected this brings up the unit wizard window, which has a number of panels that need be completed before the skeleton code can be generated. In the next sections we look at the options on each of these panels.

Unit Panel

The first panel in the Unit Wizard (shown in Figure 4.3) has a number of fields allowing basic information about the unit to be specified. This information is as follows:

Unit Name - The name of the Java class that is generated and also the default name for the tool/tasks representing this unit. When a tool/task is renamed

this does not change the name of the underlying Java unit.

Unit Package - The package of the Java class that is generated and also the default package for the tool representing this unit. When a tool is moved to a different package this does not change the package of the underlying Java unit.

Tool Box Path - The base toolbox directory where the unit code will be generated (see Section 4.2.1).

Author - The name(s) of the unit author(s).

PopUp Description - A brief pop-up description of the units function. This description is displayed when the mouse of hovered over the unit.

Date - The date of creation.

Help File - The html file which is displayed when the user requests help on this unit. A skeleton help file is generated at the same time as the unit code.

Include Copyright - Whether the default Triana copyright is included in the generated skeletion.

Input Nodes - Allows the default number of input nodes to be specified. If resize is selected then the range of acceptable input nodes can be specified, otherwise the number is fixed at the default.

Output Nodes - Allows the default number of output nodes to be specified. If resize is selected then the range of acceptable output nodes can be specified, otherwise the number is fixed at the default.

Select *Next* to accept the values specified in the Unit Panel. Note that these values *can* be changed later, either in the wizard or in the generated code.

Data Type Panel

The second panel the Unit Wizard (shown in Figure 4.4) allows the data types which are input/output by the unit to be specified. This information allows type-checking between units to be done at connection time. When an input/output type is added, two pieces of information are required:

Node - The index of the node the type is input/output on (where 0 is the first node, 1 is the second and so on). The value All Nodes indicates that the data type applies to any node. The value Other Nodes indicates that the data type applies to nodes after those that have data types explicitly set.

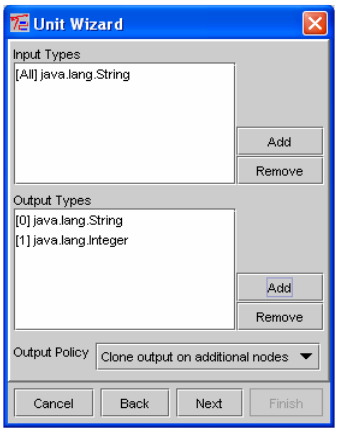


Figure 4.4: Data Type Panel in the Unit Wizard.

Data Type - The data type input/output. This can be any standard java type (e.g. java.lang.String), Triana type (e.g. triana.types.SampleSet), or any other type that is on the Triana classpath.

In addition to the data types, the data type panel also allows the data output policy to be set. This is the policy that is used when the general `output()` method is called by the unit (as opposed to the specific `outputAtNode()` method. The available output policy options are:

Copy output (pass by reference) - A reference to the data is output on each node, no cloning is performed. Any changes later tasks make to the data will affect all other tasks that received the reference. This policy should only be used if the effect is desired.

Clone output on additional nodes - A reference to the data is output on the first node, but clones of the data are output on all other nodes (if possible). This is the default policy as changes later tasks make to the data will not affect other tasks as they received a different version of the data.

Clone output on additional nodes - Clones of the data are output on all nodes (if possible). This policy is required if the task outputting the data holds on to the data for future update. With this policy these updates will not affect other tasks as every task received a clone of the data.

Select *Next* to accept the values specified in the data type panel. Note that these values *can* be changed later, either in the wizard or in the generated code.

Figure 4.5: Parameter Panel in the Unit Wizard.

Parameter Panel

The third panel the Unit Wizard (shown in Figure 4.5) allows the parameters used by the unit to be specified. Parameters are used to represent, steer and store the internal state of the unit. As the state of parameters is automatically shared between the unit and its graphical interface by Triana, they are the method of choice for communications between the unit and its graphical interface (also referred to as its parameter panel). When adding a parameter, the following information is required:

Parameter Name - The name of the parameter, also used as the variable name for the parameter in the Java code. This should not include spaces or other punctuation characters (except underscore).

Default Value - The default value assigned to the parameter.

Data Type - The data type of the parameter in the Java code. Note that all parameters specified in the Unit Wizard are stored internally as strings and automatically converted into the required data type within the Java code.

Parameter Type - The type of the parameter: `User Accessible` indicates it can be viewed/steered by the user, for example via the parameter panel or parameter input/output nodes; `Internal` indicates the parameter is not visible to the user; and `Transient` indicates the parameter is not visible to the user and is not stored when the tool is saved.

In addition to the parameters, the parameter panel also allows the parameter update policy to be specified. The parameter update policy states when parameters updated by the user are informed to the Java unit. The available options are:

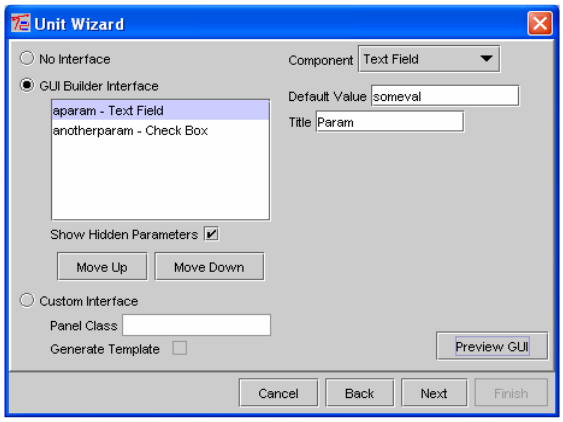


Figure 4.6: GUI Panel in the Unit Wizard.

Update at start of process - The updates are informed to the Java unit before the process method is called (but no while it is executing). This is the default policy as it prevents unexpected parameter changes during processing.

Update immediately - The updates are informed to the Java unit immediately the parameter is changed, even if in the middle of processing. This option is required by steerable units.

Do not update - The updates are never informed to the Java unit. Any update mechanism has to be implemented explicitly using TaskListeners.

Select *Next* to accept the values specified in the parameter panel. Note that these values *can* be changed later, either in the wizard or in the generated code.

GUI Panel

The fourth panel the Unit Wizard (shown in Figure 4.6) allows the graphical user interface for a unit to be specified. The three main categories available are:

No Interface - No user interface.

GUI Builder Interface - Uses the Triana GUI Builder to define an interface for updating the unit's parameters. We discuss this further below.

Custom Interface - Specifies a Java class to act as the parameter panel for the unit. This class must extend `ParmaterPanel` in `triana.gui.panels`. If required template Java code for the panel class can also be generated when the unit code is generated.

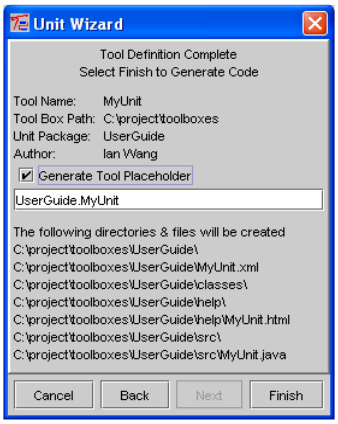


Figure 4.7: Final Panel in the Unit Wizard.

If GUI Builder interface is selected an additional panel appears allowing GUI components to be associated with each of the unit parameters (as specified in Section 4.2.3). A number of standard components are available, including TextField, CheckBox, Choice, ScrollBar and File Chooser (a TextField with Browse button). The Preview GUI button at the bottom of this panel allows the defined interface to be previewed.

Select *Next* to accept the values specified in the GUI panel. Note that these values *can* be changed later, either in the wizard or in the generated code.

Final Panel

The final panel the Unit Wizard (shown in Figure 4.7) shows information on the unit and which files and directories will be created when the unit code is generated. The Generate Tool Placeholder option indicates whether a dummy tool in the tool tree representing the unit will be created. Once the `process()` method has been implemented, right-clicking on this dummy tool and selecting *Compile* will compile the actual tool.

Select *Finish* to generate the tool specified in the Unit Wizard. The Java source files for the unit, html help and parameter panel (if specified) will be created in the specified toolbox.

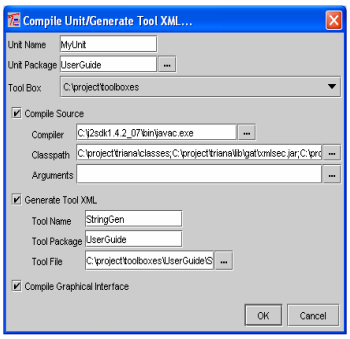


Figure 4.8: Compile Unit/Generate Tool XML Dialog.

4.2.4 Compiling Units/Generating Tool XML

Once a Java unit has been created (see Section 4.2.3) it needs to be compiled before it can be used within Triana. Also, in order for the unit to appear in the Triana tool tree, an XML tool representing the unit must also be generated. Both compilation and tool XML generation can be done using the *Tools→Compile Unit/Generate Tool XML* menu option. Alternatively, to recompile an existing tool, right-click on that tool in the tool tree and select *Compile*.

In Figure 4.8 we show the dialog for compiling Triana tools. The available options in this dialog include:

Unit Name - The name of the unit, as specified in the Unit Wizard.

Unit Package - The package of the unit, as specified in the Unit Wizard. Using the browse button next the unit package field allows the Java source for a unit to be selected; the unit name and unit package will automatically be set.

Toolbox - The base toolbox for the unit (see Section 4.2.1)

Compile Source - Selects whether the source code for the unit is compiled. The Javac compiler, classpath and arguments can be set.

Generate Tool XML - Selects whether a XML tool definition of the unit is generated. The name, package and file for the generated tool can be set.

Compile Graphical Interface - Selects whether the parameter panel source for the unit is compiled. This only applies to custom parameter panels (see Section 4.2.3.

Select *OK* to compile the unit. This process can take a while as the javac compiler

must be invoked (unless only generating tool XML). A debug window appears showing the output from the compiler. Note that if the graphical interface is also being compiled the two debug windows will appear.

4.3 Advanced Tool Techniques

In this section we look at some more advanced techniques that a tool developer may wish to use. Where source code is listed only the important code or code modified from template code is listed. For full listings look at the tool source code listings in the `UserGuide` toolbox in Triana.

4.3.1 Showing and Hiding a Unit's Parameter Panel

Problem

You want to simulate the user double clicking on a task to display the unit parameter panel.

Solution

Call the method `public void showParameterPanel()` to display the unit's parameter panel and `public void showParameterPanel()` to hide the panel again. These methods are inherited from the `Unit` superclass.

Discussion

Inside the `process()` method for your unit call the `showParameterPanel()` method.

```
/*
 * Called whenever there is data for the unit to process
 */
public void process() throws Exception {

    // display the units parameter panel interface
    showParameterPanel();
}
```

4.3.2 Pausing Unit Execution

Problem

You want to pause the execution of your unit programmatically until some event happens.

Solution

Pause the thread that is running your unit and interrupt on your desired event to resume processing.

Discussion

Say for instance that you wish to pause the execution of your unit, between the point in the main `process()` method where the unit gets input from its input node, and the point where it outputs the result. The execution should halt until a particular parameter has been updated.

The preferred mechanism for doing this is to cause the current thread to sleep, interrupting the thread on the parameter being updated.

Note: An important thing to remember is that the *parameter update policy* must be set in the unit set up to be update immediately as opposed to the default behaviour of updating on the execution of the process method. Without this change to the default behaviour the execution will hang.

```
package UserGuide;

import triana.unit.Unit;

/**
 * This unit pops up its user interface and pauses execution until something
 * has been selected
 *
 * @author Matthew Shields
 * @created 23 Feb 2005
 */
public class PopUpAndPause extends Unit {

    // parameter data type definitions
    private String selectedString;

    // Flag set when a user selects string
    boolean stringSelected;

    // Flag used by a user initiated stop
    boolean stopped;

    /*
     * Called whenever there is data for the unit to process
     */
```

```
    public void process() throws Exception {
        // set the flags to false
        stringSelected = false;
        stopped = false;

        // display the units parameter panel interface
        showParameterPanel();

        while (!(stopped || stringSelected)) {
synchronized{this} {
                try {
                    this.wait();
                } catch (InterruptedException e) {
                }
            }
        }

        output(selectedString);
    }

    /**
     * Called when the unit is created. Initialises the unit's properties
     * and parameters.
     */
    public void init() {
        super.init();

        // Initialise node properties
        setDefaultInputNodes(0);
        setMinimumInputNodes(0);
        setMaximumInputNodes(0);

        setDefaultOutputNodes(1);
        setMinimumOutputNodes(1);
        setMaximumOutputNodes(Integer.MAX_VALUE);

        // Initialise parameter update policy and output policy
        // IMPORTANT - this must be changed for the unit to restart it's thread
        setParameterUpdatePolicy(IMMEDIATE_UPDATE);

        setOutputPolicy(CLONE_MULTIPLE_OUTPUT);

        // Initialise pop-up description and help file location
        setPopUpDescription("This unit pops up its user interface and pauses");
        setHelpFileLocation("PopUpAndPause.html");

        // Define initial value and type of parameters
        defineParameter("selectedString", "", USER_ACCESSIBLE);
```

```
        // Initialise GUI builder interface
        String guilines = "";
        guilines += "Select a string $title selectedString Choice";
        guilines += "[Mary] [Had] [A] [Little] [Lamb]\n";
        setGUIBuilderV2Info(guilines);
    }

    /**
     * This is called when the network is forcably stopped by the user.
     * This should be over-ridden with the desired tasks.
     * <p/>
     * We need to override this method to quit our paused thread
     */
    public void stopping() {
        super.stopping();
        stopped = true;
        synchronized(this) {
            unitThread.notifyAll();
        }
    }

    /**
     * Called a parameters is updated (e.g. by the GUI)
     */
    public void parameterUpdate(String paramname, Object value) {
        // Code to update local variables
        if (paramname.equals("selectedString")) {
            selectedString = (String) value;
            System.out.println("selectedString = " + selectedString);

            // We need to check that the value is not an empty string as
            // Triana updates parameters at
            // intialisation time and we don't want that to trigger here
            if (!(selectedString.equals(""))) {
                stringSelected = true;

                synchronized(this) {
                    unitThread.notifyAll();
                }
            }
        }
    }
}
```

Symptom	Solution
No tools in the toolbox tree	remove the Triana resource files and restart. See page 68

Table 4.1: Trouble Shooting Symptoms

4.4 Trouble Shooting

4.4.1 Triana Resource Directory

Triana stores its settings in a special directory which can be found in the *user home* directory, the location of *user home* is specific to your operating system and system user name but typically on linux or unix based systems it will be designated by the ~ symbol, so typing `cd ~` at the command line should take you to your home directory. On windows the user home directory is typically found under `C:\Documents & Settings\<username>\`.

Under the user home directory there is a special hidden directory created by Triana called `.TrianaV3Resources`. This contains all of the settings that Triana stores in various files. If you have upgraded Triana or it is not behaving as expected, for instance no tools or toolboxes are showing up in the toolbox tree, delete the whole directory. Next time Triana is started it will recreate the directory with the default settings.

Index