Scribble 2.0

User Guide

Installation

This is the installation guide for the Scribble tools. Scribble is a notation for describing interaction based protocols between multiple parties.

1.1. Pre-Requisites

The pre-requisites for the Scribble design time tools are:

1. Java
   The design time tools are Java based, so you will need a suitable JVM (Java Virtual Machine) to run the tools. If you also intend to generate Java APIs, for use at runtime, then you will need a JDK (Java Development Kit).

   Download the version 1.6 (or higher) from the http://www.java.com. Once downloaded, follow the instructions to install the JVM or JDK on your system.

2. Eclipse
   The Scribble protocol descriptions can be edited using a standard text editor, and the Scribble tools can be invoked using command line tools (as described in the following chapter).

   However it is also possible to use the Scribble tools from within the Eclipse IDE environment, by installing the Scribble tools as plugins. If you wish to use this approach, then you will need a version of Eclipse (3.6 or higher) which can be downloaded from the Eclipse website: http://www.eclipse.org.

1.2. Installing the Command Line Version

To install the command line version of the Scribble tools:

1. Download the Scribble tools distribution from: http://www.jboss.org/scribble/downloads.

2. Unpack the tool distribution in a suitable location

3. Setup environment
   The commands can be executed from the bin folder of the Scribble tools distribution. Alternatively, the `bin` folder can be added to the execution path, to enable the commands to be performed from any folder.

   For example, on Linux running bash, simply edit the `.bash_profile` file within your home directory to add:

```
PATH=$PATH:${path-to-scribble}/bin
```

1.3. Installing the Eclipse Version

The Eclipse plugins for the Scribble tools can be loaded from an update site using the update manager.

Select the *Help->Install New Software...* menu item, from your Eclipse environment, and follow the instructions.

If installing a stable or development milestone release of the tools, then the plugins can be installed directly from the network. If installing a nightly build of the plugins, to access the latest (but unstable) version of the tools, then it will be necessary to download the update site as an archive, and point the Eclipse update manager at the downloaded zip file.

The location of the stable, development and nightly builds can be found on the Scribble download page: http://www.jboss.org/scribble/downloads.

Once the Eclipse environment has restarted, and if the distribution has been downloaded and expanded, then the next step is to import the samples into the Eclipse workspace.

This can be done by selecting the *Import->General->Existing Projects into Workspace* and locating the `samples` folder from the Scribble distribution.

This will display the set of projects within the `samples` folder, which can either be individually selected, or all imported at once. Once imported, the Scribble Protocol files can be opened by double clicking the file, to open the protocol context sensitive editor.

Command Line Tools

This section describes how to use the command line tools that are available in the `bin` folder of the Scribble protocol tools distribution.

Information on the Scribble protocol notation (or language) can be found in the *Scribble Protocol Guide*.

2.1. Parsing a Protocol description

The *parse* command takes a single parameter, which is the path to the file containing the protocol description to be parsed.

For example, if the user is in the top level folder of the Scribble tools distribution, without the `bin` folder being added to the system path, then the following command can be executed to parse one of the sample protocol descriptions:

```
bin/parse.sh samples/parse_and_validate/OrderProcess.spr
```

If the supplied file path is not valid, then the command will report an error.

This command will read the protocol description, as shown below, and convert it into an internal object model representation.

```
import MyOrder;

protocol OrderProcess {
 role Buyer, Seller;

 MyOrder from Buyer to Seller;
}
```

If any errors are detected in the syntax of the parsed protocol description, then these will be reported to the command window. For example, if you edit the supplied file, and change the keyword *from* to append an 'X', then the following error would be produced:

```
ERROR: [line 6] no viable alternative at input 'fromX'
```

2.2. Validating a Protocol description

The *validate* command takes a single parameter, which is the path to the file containing the protocol description to be validated.

For example, if the user is in the top level folder of the Scribble tools distribution, without the `bin` folder being added to the system path, then the following command can be executed to validate one of the sample protocol descriptions:

```
bin/validate.sh samples/parse_and_validate/OrderProcess.spr
```

When this command is performed initially, it will complete without any errors. However if you edit the `samples/parse_and_validate/OrderProcess.spr` file, and change the following line:

```
 MyOrder from Buyer to Seller;
```

For example, change the *Seller* role to *Seller2*, and then re-run the *validate* command. This will result in the following error messages:

```
ERROR: [line 6] Unknown role 'Seller2'
```

2.3. Checking Conformance a Protocol description

The *conforms* command takes two parameters, which are both paths to a file containing a protocol description. The first parameter is the protocol description to be checked for conformance against the second parameter's protocol description. So the second parameter is the *reference* protocol description.

For example, if the user is in the top level folder of the Scribble tools distribution, without the `bin` folder being added to the system path, then the following command can be executed to check one of the sample protocol descriptions as being conformant with another reference protocol description:

```
bin/conforms.sh           samples/conformance_descriptions/OrderProcess.spr           samples/
conformance_descriptions/ReferenceOrderProcess.spr
```

If you inspect the two process definitions, you will find one difference. The first protocol definition has the following interaction:

```
 MyOrder from Buyer to Seller;
```

The second, reference protocol description, has the following interaction:

```
 Order from Buyer to Seller;
```

This results in the following conformance error message:

```
ERROR: Type mismatch with referenced description, was expecting Order
```

2.4. Projecting a Protocol description

The *project* command takes two parameters. The first parameter is the protocol description to be projected and the second parameter is the *participant*.

For example, if the user is in the top level folder of the Scribble tools distribution, without the `bin` folder being added to the system path, then the following command can be executed to project one of the sample protocol descriptions:

```
bin/project.sh samples/parse_and_validate/OrderProcess.spr Seller
```

This results in the following located Protocol being displayed on the console:

```
import MyOrder;
protocol OrderProcess @ Seller {
 role Buyer;
 MyOrder from Buyer;
}
```

2.5. Simulating a Protocol description against an Event List

The *simulate* command takes two parameters. The first parameter is the located protocol description and the second parameter is the event list to be simulated against the protocol.

For example, if the user is in the top level folder of the Scribble tools distribution, without the `bin` folder being added to the system path, then the following command can be executed to simulate the protocol description:

```
bin/simulate.sh samples/monitor/Purchasing@Buyer.spr samples/monitor/Purchasing@Buyer.events
```

The event file is a *comma separated value (csv)* format, with the first column representing the event type, and the second representing the value relevant for the event type. The event types are listed below:

- sendMessage
  The value represents the message type.
- receiveMessage
  The value represents the message type.
- sendChoice
  The value represents the choice label.
- receiveChoice
  The value represents the choice label.
- sendDecision
  The value represents the decision boolean value (e.g. true or false). This can be used in conjunction with an *Optional* or *Repeat* protocol construct.
- receiveDecision
  The value represents the decision boolean value (e.g. true or false). This can be used in conjunction with an *Optional* or *Repeat* protocol construct.

The event file used in the sample command above is:

```
sendMessage,Order,Broker
receiveChoice,_Confirmation,Broker
receiveMessage,Confirmation,Broker
```

and the result of running the command is:

```
INFO: Validated SendMessage Order to Broker
INFO: Validated ReceiveChoice _Confirmation from Broker
INFO: Validated ReceiveMessage Confirmation from Broker
```