



Genera Integration Guide

Edition 1.1.5

ReTiSoft Inc.

48 Forty-First Street, Suite 1
M8W-3N6 Toronto, Ontario, Canada

Tel: (416)521-9720

Fax: (416)521-9277

info@retisoft.ca

www.retisoft.ca

© ReTiSoft Inc., July 2002

Contents

1	Architecture	Page 1
2	Instrument Plugin	Page 4
2.1	Virtual Instrument	Page 4
2.1.1	Virtual Instrument Behaviour	Page 5
2.1.2	Virtual Device Behaviour	Page 13
2.2	Instrument Directory	Page 17
2.2.1	Documentation	Page 17
2.2.2	Images	Page 18
2.2.3	Configuration Files	Page 18
2.2.4	XML File	Page 19
2.3	Driver Development	Page 21
2.3.1	ControllableDriver Interface	Page 21
2.3.2	ControllableInstrument Interface	Page 23
2.4	Configuring Genera	Page 25
3	Fixture Plugin	Page 26
3.1	Fixture Directory	Page 26
3.1.1	Images	Page 27
3.1.2	XML File	Page 27
3.2	Configuring Genera	Page 28
4	Container Plugin	Page 30
4.1	Container Directory	Page 30
4.1.1	Images	Page 31
4.1.2	XML File	Page 31
4.2	Configuring Genera	Page 32

5	Resource Plugin	Page 34
5.1	Resource Directory	Page 34
5.1.1	Images	Page 35
5.1.2	XML File	Page 36
5.2	Configuring Genera	Page 36
A	Instrument DTD	Page 38
B	Instrument XML	Page 41
C	Fixture DTD	Page 48
D	Fixture XML	Page 50
E	Container DTD	Page 52
F	Container XML	Page 54
G	Resource DTD	Page 58
H	Resource XML	Page 60

List of Figures

1.1	Genera Architecture	Page 1
2.1	Virtual Instrument	Page 5
2.2	Virtual Instrument Behaviour	Page 6
2.3	Connect Accepted Interaction	Page 7
2.4	Connect Rejected Interaction	Page 8
2.5	Disconnect Accepted Interaction	Page 9
2.6	Disconnect Rejected Interaction	Page 10
2.7	Initialize Accepted Interaction	Page 11
2.8	Initialize Rejected Interaction	Page 12
2.9	Emergency Stop Interaction	Page 13
2.10	Virtual Device Behaviour	Page 14
2.11	Command Accepted Interaction	Page 15
2.12	Command Rejected Interaction	Page 16
2.13	Instrument Directory	Page 17
2.14	Instrument List Window	Page 18
3.1	Fixture Directory	Page 26
3.2	Fixture List Window	Page 27
4.1	Container Directory	Page 30
4.2	Container List Window	Page 31
5.1	Resource Directory	Page 35
5.2	Resource List Window	Page 35

Chapter 1

Architecture

Genera, our instrument integration framework, is based on an open and extensible architecture that allows the user to control and monitor any instrument. Due to the generic nature of its architecture, the instruments may use different communication technologies and protocols.

With Genera, your lab automation system can quickly become a mix of instruments from multiple hardware vendors, which allows you to competitively select the best-of-breed equipment to meet your unique lab automation requirements. Figure 1.1 shows the high-level architecture of our integration framework.

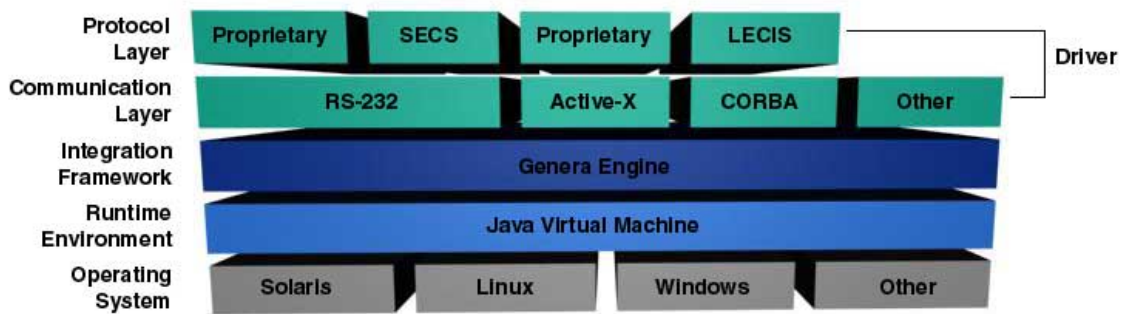


Figure 1.1: Genera Architecture

Operating System Genera application is entirely written in Java and it runs on *Sun SolarisTM*, *Linux*, *Microsoft WindowsTM* and other operating systems which contain Java Virtual Machine.

Runtime Environment In order to run our integration framework Java Runtime Environment (JRE) version 1.3.1 or later must be installed.

Integration Framework Genera engine along with its underlying libraries allows easy integration of diversified laboratory instruments. Instrument properties are described by well-defined XML files, which can also serve as instrument interface documentation.

Driver Genera driver is a pluggable software component which sends commands to the laboratory instrument and listens for responses and alarms coming from the instrument. The driver is comprised of two software layers:

Communication layer determines what type of communication media the instrument is using, e.g. RS-232 port, Active-X, CORBA, TCP/IP sockets, XML RPC, SOAP, etc.

Protocol layer determines the structure of messages and hand-shaking which occurs between the driver and the instrument. The driver may either implement a standardized protocol, such as SECS¹ or LECIS², or proprietary protocols provided by instrument vendors.

The guide explains how to integrate instrument drivers, fixtures, containers and resources into the Genera framework. Chapter 1 provides an overview of the Genera architecture. Here is a brief description of what is included in the remaining chapters and appendices:

Chapter 2 Explains the structure and behaviour of instruments. It also describes how to write and integrate instrument drivers into the Genera framework.

Chapter 3 Explains the concept of fixtures and shows how they can be integrated into the Genera framework.

Chapter 4 Explains the concept of containers and shows how they can be integrated into the Genera framework.

Chapter 5 Explains the concept of resources and shows how they can be integrated into the Genera framework.

¹SECS is a coordinated pair of standards for the semiconductor industry that defines an RS-232 communications interface between semiconductor equipment and a host. SECS stands for SEMI Equipment Communications Standard and is defined in SEMI E4-91 (SECS-I) and SEMI E5-95 (SECS-II) of the SEMI standards.

²LECIS stands for the Laboratory Equipment Control Interface Specifications. The standard defines a remote control interface between laboratory instruments and a host. LECIS was initially defined by the American Society for Testing and Materials (ASTM).

Appendix A Data Type Definition (DTD) of an XML file for an instrument.

Appendix B Example of an XML file for an instrument.

Appendix C Data Type Definition (DTD) of an XML file for a fixture.

Appendix D Example of an XML file for a fixture.

Appendix E Data Type Definition (DTD) of an XML file for a container.

Appendix F Example of an XML file for a container.

Appendix G Data Type Definition (DTD) of an XML file for a resource.

Appendix H Example of an XML file for a resource.

Chapter 2

Instrument Plugin

The Genera framework uses instrument plugins (drivers) to communicate with physical instruments. To develop a new instrument driver, you should know the following:

1. How the physical instrument is represented (modeled) in our framework (refer to Section 2.1)
2. Directory structure for the instrument driver (refer to Section 2.2)
3. Java APIs which are used to integrate the driver with the Genera framework (refer to Section 2.3)
4. How to configure Genera to recognize the instrument driver (refer to Section 2.4)

2.1 Virtual Instrument

The entire automation system (a system which is composed of several physical instruments) is modeled by the corresponding virtual instruments and virtual devices within the Genera framework. In other words, each virtual instrument within the framework maps to the physical instrument being controlled.

The behaviour of the entire automation system can be characterized by the behaviour of each virtual instrument and its virtual devices. During the system execution, the state of each physical instrument must be synchronized with the state of its corresponding virtual instrument.

Each virtual instrument contains one or more virtual devices. Virtual devices in Genera represent subcomponents, (e.g. robotic arm, decapper, moving tray, carousel, etc) within the physical instrument. Devices use the same communication link as their instrument and they can accept commands when other commands are still executing in other devices. In other words, commands can execute in parallel among several different virtual devices.

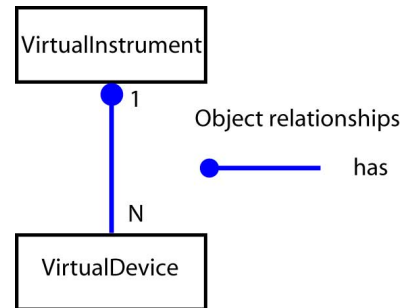


Figure 2.1: Virtual Instrument

Before we describe the behaviour of virtual instruments and devices, let us explain the terminology we will use in the remaining paragraphs.

Command is defined as a message which propagates from the scheduler to the physical instrument.

Response is defined as a message which is sent by the physical instrument to the controlling software.

2.1.1 Virtual Instrument Behaviour

Virtual instrument behaviour can be denoted by well-defined instrument states (e.g. Connected or Disconnected), and commands and responses which are accepted in these states.

States

Disconnected - physical instrument must be controlled manually by the operator.

Disconnecting - Genera driver is closing the connection with the physical instrument.

Connected - physical instrument is remotely controlled by the Genera driver.

Connecting - Genera driver tries to establish connection with the physical instrument.

Initializing - physical instrument is being initialized.

Ready - physical instrument is ready for operation.

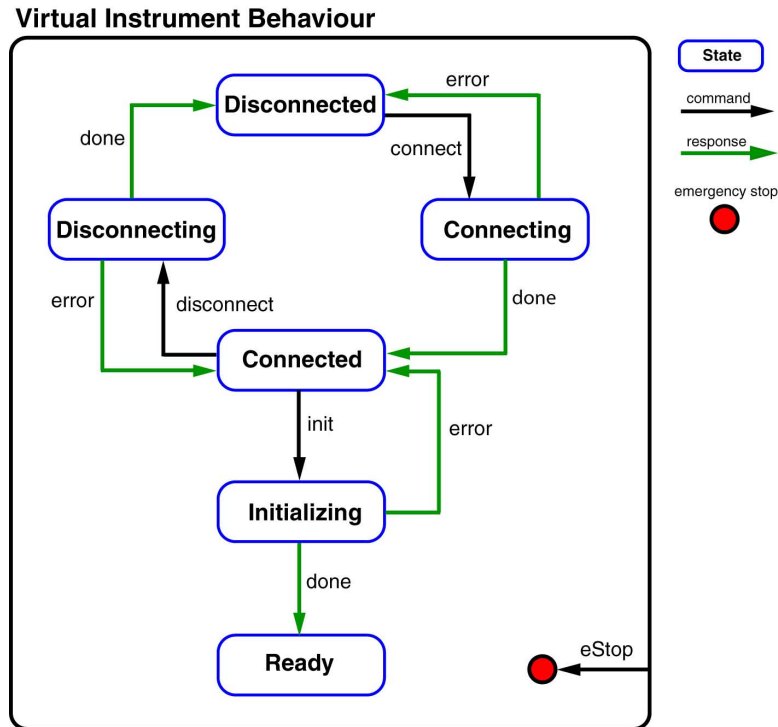


Figure 2.2: Virtual Instrument Behaviour

Commands and Responses

connect - General driver initiates remote connection with the physical instrument.

disconnect - General driver terminates remote connection with the physical instrument.

initialize - General driver initializes the physical instrument.

eStop - General driver sends an emergency stop to the physical instrument.

done - command completed its execution on the physical instrument.

error - non-critical error occurred on the physical instrument.

Interactions

In this section, we describe the possible data flow (i.e. interactions) between the scheduler (i.e. controlling software which dispatches instrument commands), Genera framework and the physical instrument. Note that all of these interactions are related to the **virtual instrument** object defined by Genera.

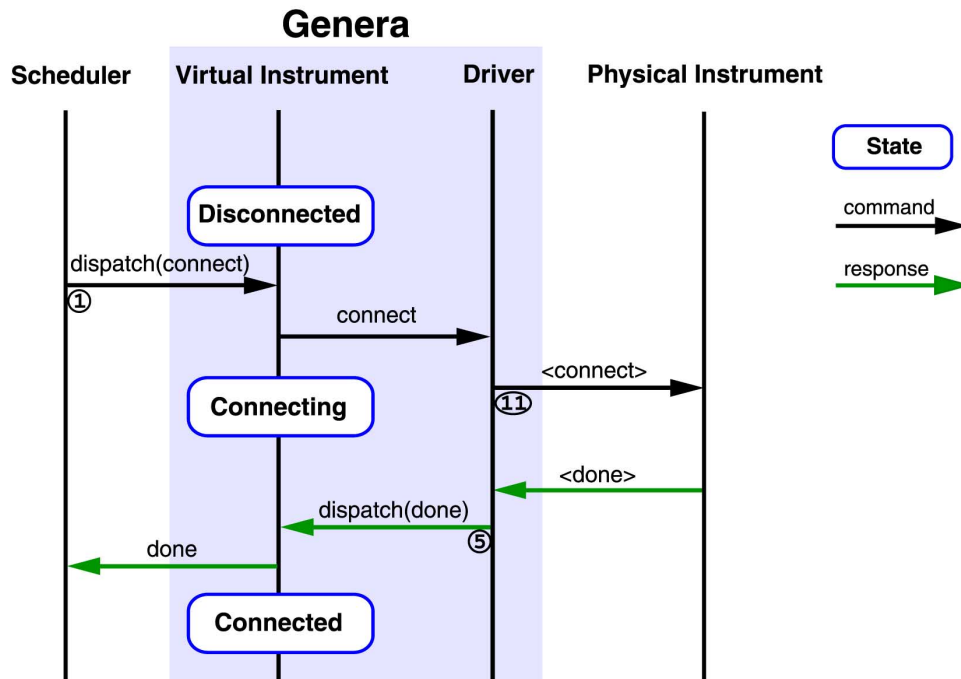


Figure 2.3: Connect Accepted Interaction

To initiate a connection with the physical instrument, the **connect** command must be dispatched to the corresponding virtual instrument. After the command is dispatched, the virtual instrument relays the command to the required instrument driver. The driver tries to establish the connection with the remote physical instrument. When the connection is established the driver must propagate the **done** response back to the virtual instrument within Genera. A few Java examples which illustrate the interactions are as follows:

```

1. InstrumentID id=new InstrumentID("PlateWasher-01");
   ConnectCommand connect=new ConnectCommand(id);
   InstrumentTimerTask task=new InstrumentTimerTask(connect);
   Dispatcher dispatcher=new Dispatcher();
   dispatcher.schedule(task, 0);
  
```

```

11. public void connect() throws DriverInitException {
    // Establish connection with the remote instrument
    ...
}

5. InstrumentID id=new InstrumentID("PlateWasher-01");
   DoneCommand done=new DoneCommand(id);
   InstrumentTimerTask task=new InstrumentTimerTask(done);
   Dispatcher dispatcher=new Dispatcher();
   dispatcher.schedule(task, 0);

```

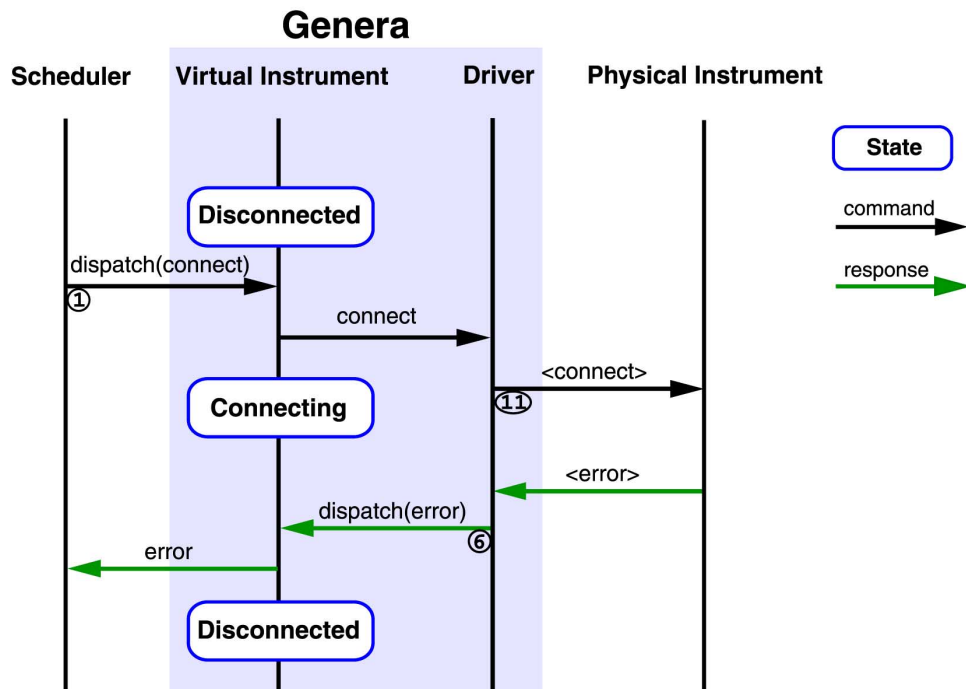


Figure 2.4: Connect Rejected Interaction

When the driver can not establish the connection with the remote instrument, it must dispatch the **error** response to the virtual instrument within Genera. If possible, the scheduler would attempt to recover from the error. A Java example which illustrates the dispatching of the `error` response is as follows:

```

6. InstrumentID id=new InstrumentID("PlateWasher-01");
   ErrorCommand error=new ErrorCommand(id);

```

```

InstrumentTimerTask task=new InstrumentTimerTask(error);
Dispatcher dispatcher=new Dispatcher();
dispatcher.schedule(task, 0);

```

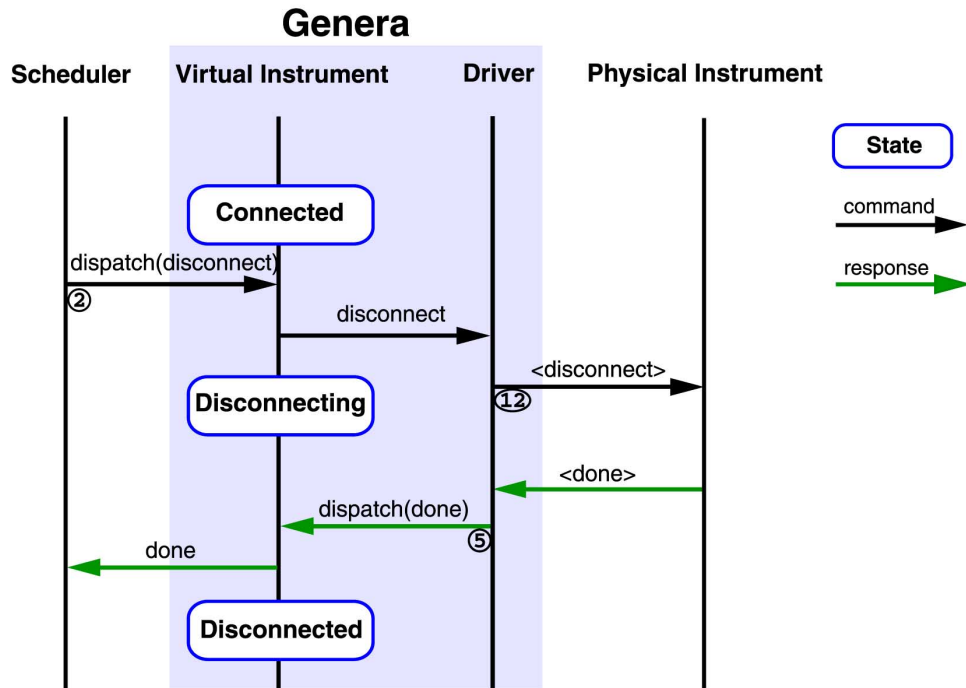


Figure 2.5: Disconnect Accepted Interaction

To disconnect the Genera driver from the physical instrument, the **disconnect** command must be dispatched to the corresponding virtual instrument. After the command is dispatched, the virtual instrument relays the command to the required instrument driver. The driver tries to disconnect itself from the remote instrument. When the connection is terminated the driver must propagate the **done** response back to the virtual instrument within Genera. A few Java examples which illustrate the interactions are as follows:

```

2. InstrumentID id=new InstrumentID("PlateWasher-01");
   DisconnectCommand disconnect=new DisconnectCommand(id);
   InstrumentTimerTask task=new InstrumentTimerTask(disconnect);
   Dispatcher dispatcher=new Dispatcher();
   dispatcher.schedule(task, 0);

```

```

12. public void disconnect() throws DriverCloseException {

```

```

    // Terminate connection with the remote instrument
    ...
}
5. InstrumentID id=new InstrumentID("PlateWasher-01");
   DoneCommand done=new DoneCommand(id);
   InstrumentTimerTask task=new InstrumentTimerTask(done);
   Dispatcher dispatcher=new Dispatcher();
   dispatcher.schedule(task, 0);

```

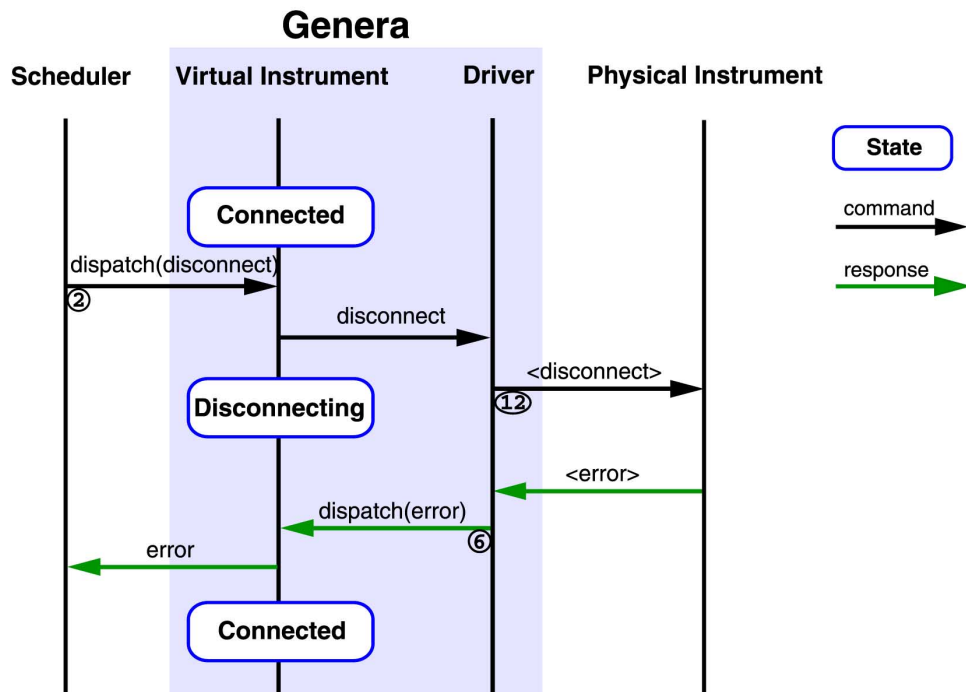


Figure 2.6: Disconnect Rejected Interaction

When the driver can not terminate the connection with the remote instrument, it must dispatch the **error** response to the virtual instrument within Genera. If possible, the scheduler would attempt to recover from the error. A Java example which illustrates the dispatching of the error response is as follows:

```

6. InstrumentID id=new InstrumentID("PlateWasher-01");
   ErrorCommand error=new ErrorCommand(id);
   InstrumentTimerTask task=new InstrumentTimerTask(error);
   Dispatcher dispatcher=new Dispatcher();
   dispatcher.schedule(task, 0);

```

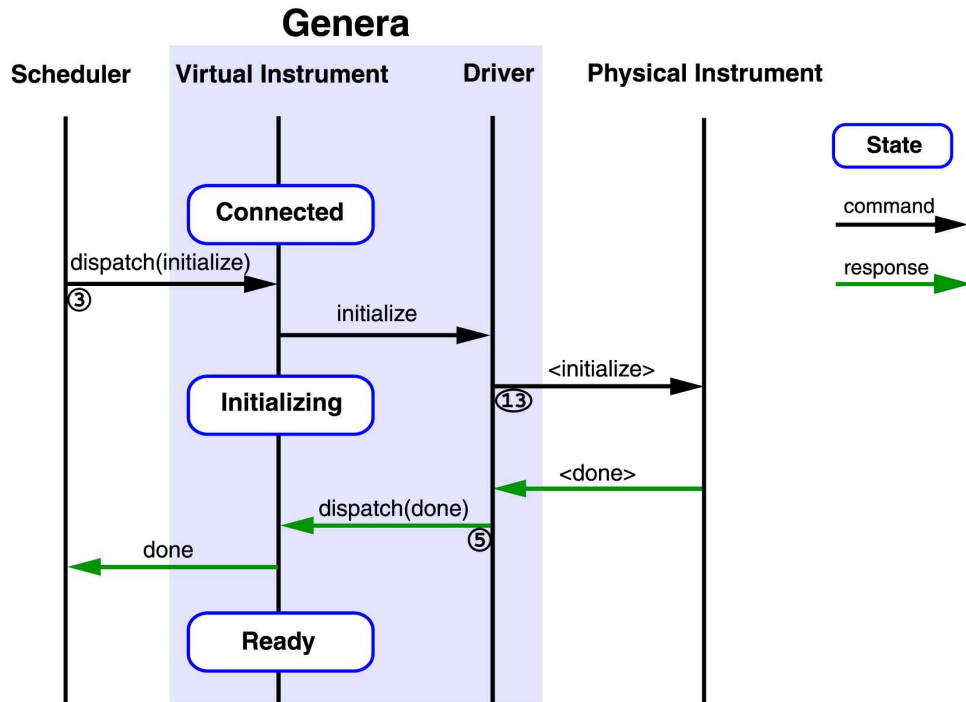


Figure 2.7: Initialize Accepted Interaction

Before proprietary commands can be relayed to the physical instrument, the instrument must first be initialized. To initialize the remote instrument, the **initialize** command must be dispatched to the corresponding virtual instrument. After the command is dispatched, the virtual instrument relays the command to the required instrument driver. The driver tries to initialize the remote instrument. When the command completes its execution on the remote instrument, the driver must propagate the **done** response back to the virtual instrument within Genera. A few Java examples which illustrate the interactions are as follows:

```

3. InstrumentID id=new InstrumentID("PlateWasher-01");
   InitializeCommand initialize=new InitializeCommand(id);
   InstrumentTimerTask task=new InstrumentTimerTask(initialize);
   Dispatcher dispatcher=new Dispatcher();
   dispatcher.schedule(task, 0);

13. public void initialize() throws CommandExecuteException {
    // Initialize the remote instrument
    ...
}
  
```

```

5. InstrumentID id=new InstrumentID("PlateWasher-01");
   DoneCommand done=new DoneCommand(id);
   InstrumentTimerTask task=new InstrumentTimerTask(done);
   Dispatcher dispatcher=new Dispatcher();
   dispatcher.schedule(task, 0);

```

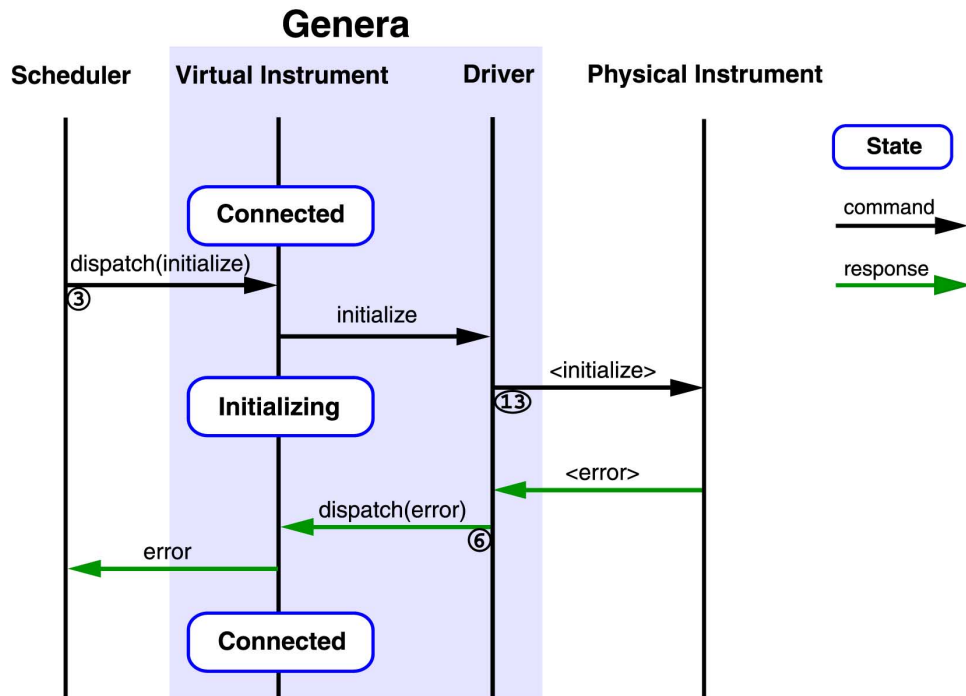


Figure 2.8: Initialize Rejected Interaction

When an error occurs on the physical instrument during its initializing stage, the driver must dispatch the **error** response to the virtual instrument within Genera. The scheduler would examine the error code and attempt to recover from the error. A Java example which illustrates the dispatching of the error response is as follows:

```

6. InstrumentID id=new InstrumentID("PlateWasher-01");
   ErrorCommand error=new ErrorCommand(id);
   InstrumentTimerTask task=new InstrumentTimerTask(error);
   Dispatcher dispatcher=new Dispatcher();
   dispatcher.schedule(task, 0);

```

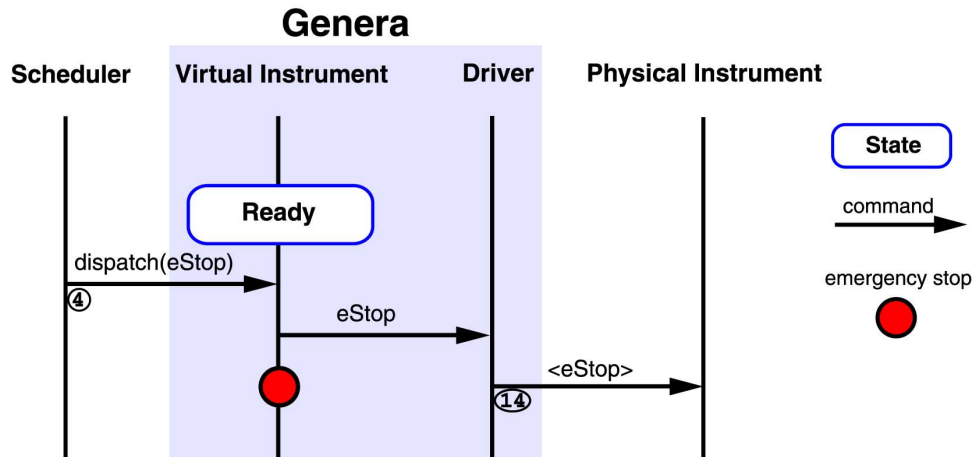


Figure 2.9: Emergency Stop Interaction

To stop the remote instrument in an emergency, the **eStop** command must be dispatched to the corresponding virtual instrument. After the command is dispatched, the virtual instrument relays the command to the required Genera driver. The driver will send the emergency stop signal to the remote instrument. A few Java examples which illustrate the interactions are as follows:

```

4. InstrumentID id=new InstrumentID("PlateWasher-01");
   EStopCommand eStop=new EStopCommand(id);
   InstrumentTimerTask task=new InstrumentTimerTask(eStop);
   Dispatcher dispatcher=new Dispatcher();
   dispatcher.schedule(task, 0);

14. public void eStop() throws CommandExecuteException {
    // Send the emergency stop signal to the remote instrument
    ...
}

```

2.1.2 Virtual Device Behaviour

Similar to the virtual instrument, virtual device behaviour can be denoted by well-defined device states (e.g. PoweredUp or Processing), and commands and responses which are accepted in these states.

States

PoweredUp - physical instrument is ready to accept commands.

Processing - physical instrument is processing a command.

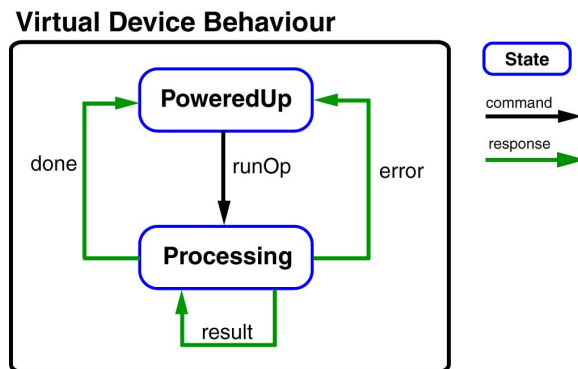


Figure 2.10: Virtual Device Behaviour

Commands and Responses

runOp - starts processing a command on the physical instrument.

result - returns data generated by the physical instrument.

done - command completed its execution on the physical instrument.

error - non-critical error occurred on the physical instrument.

Interactions

In a similar way to instrument interactions, we will describe the possible data flow (i.e. interactions) between the scheduler (i.e. controlling software which dispatches instrument commands), Genera framework and the physical instrument. Note that all of these interactions are related to the **virtual device** object defined by Genera.

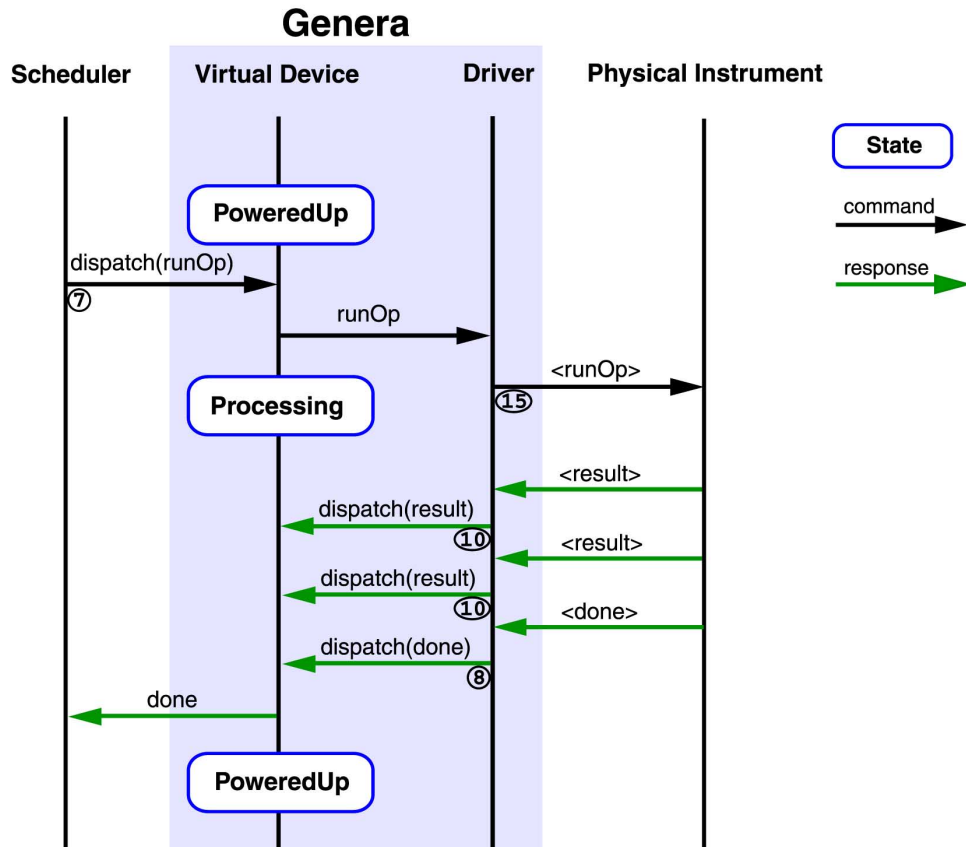


Figure 2.11: Command Accepted Interaction

To execute a proprietary command on the physical instrument, the **runOp** command must be dispatched to the corresponding virtual device. After the command is dispatched, the virtual device relays the command to the required instrument driver. The driver formats the command and sends it to the remote instrument. If the remote instrument generates a result, the driver must propagate the **result** response to the corresponding virtual device. When the command completes its execution on the remote instrument, the Genera driver must propagate the **done** response to the corresponding virtual device within Genera. A few Java examples which illustrate the interactions are as follows:

```

7. DeviceID id=new DeviceID(instrumentID, "0");
   RunOpCommand command=new RunOpCommand(id, "Wash", arguments);
   DeviceTimerTask task=new DeviceTimerTask(command);
   Dispatcher dispatcher=new Dispatcher();
   dispatcher.schedule(task, 0);
  
```

```

15. public void runOp(DeviceID deviceID, String commandName,
    CommandArgument[] commandArguments)
    throws CommandExecuteException {

    // Send the specified command to the remote instrument
    ...
}

10. DeviceID id=new DeviceID(instrumentID, "0");
ResultCommand response=new ResultCommand(id, arguments);
DeviceTimerTask task=new DeviceTimerTask(response);
Dispatcher dispatcher=new Dispatcher();
dispatcher.schedule(task, 0);

8. DeviceID id=new DeviceID(instrumentID, "0");
DoneCommand done=new DoneCommand(id);
DeviceTimerTask task=new DeviceTimerTask(done);
Dispatcher dispatcher=new Dispatcher();
dispatcher.schedule(task, 0);

```

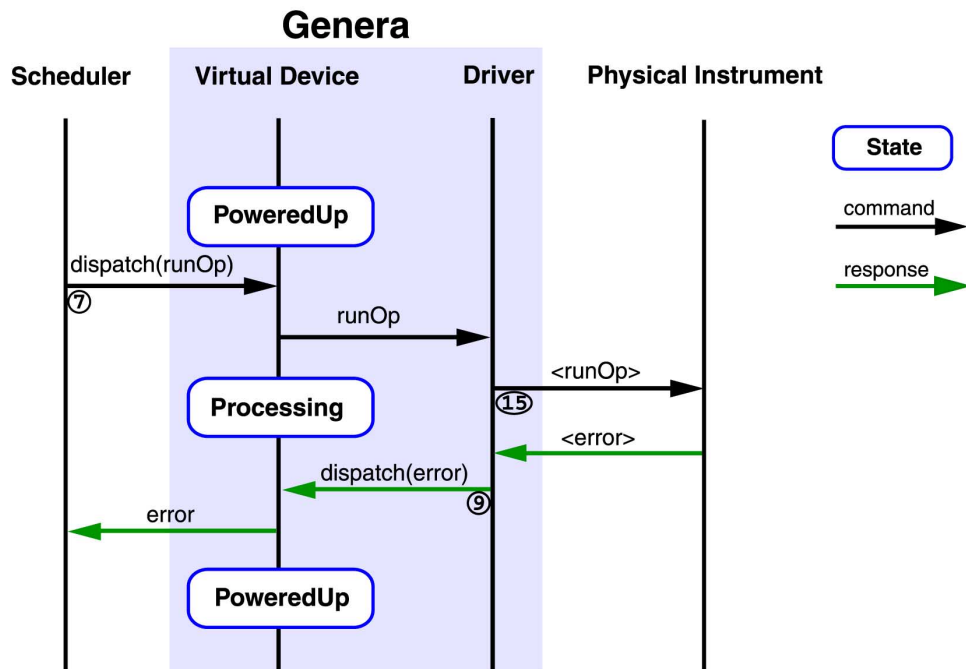


Figure 2.12: Command Rejected Interaction

When an error occurs on the physical instrument, during the execution of a proprietary command, the Genera driver must dispatch the **error** response to the corresponding virtual device. As before, the scheduler would examine the error code and would attempt to recover from the error. A few Java examples which illustrate the interactions are as follows:

```
9. DeviceID id=new DeviceID(instrumentID, "0");
   ErrorCommand error=new ErrorCommand(id);
   DeviceTimerTask task=new DeviceTimerTask(error);
   Dispatcher dispatcher=new Dispatcher();
   dispatcher.schedule(task, 0);
```

2.2 Instrument Directory

In this section, we describe the directory structure required for each instrument driver.

Each instrument driver must contain four directory folders (**doc**, **images**, **properties** and **xml**) which must be located in the **genera** directory of the corresponding instrument. If you want to install the driver in the C:\retisoft\platewasher-rs232 directory, then the C:\retisoft\platewasher-rs232\genera directory must contain four of the above mentioned directory folders.



Figure 2.13: Instrument Directory

In the following sections we describe the contents of each of these directory folders.

2.2.1 Documentation

The documentation folder for the Genera driver should contain the release notes and any other specific documentation for the driver.

2.2.2 Images

The images folder for the Genera driver must contain the picture of the instrument. The picture file must satisfy the following conditions:

file name - picture.jpg

file format - JPEG file format

picture dimensions - 150x150 pixels

The **Instrument List** window, depicted in Figure 2.14, displays all the instruments which can be loaded, as well as the image of each instrument. A single-click on the name of the instrument displays the picture.jpg file for the corresponding instrument.



Figure 2.14: Instrument List Window

2.2.3 Configuration Files

When an instrument is loaded (see Genera User's Guide), the user must specify a unique serial number for the virtual instrument. The serial number identifies the driver configuration file for this instrument. The location of the configuration file for each driver is determined as follows:

```
C:\retisoft\\genera\properties\
```

where:

instrument-name is the instrument name.

serial-number is the serial number retrieved from the XML file for the instrument driver.

For example, a RS-232 plate washer driver, which uses the PlateWasher-01 serial number will use the C:\retisoft\platewasher-rs232\genera\properties\PlateWasher-01 configuration file to communicate with the physical instrument.

2.2.4 XML File

The xml folder for the Genera driver must contain the XML file describing the virtual instrument. The XML file must satisfy the following conditions:

file name - `instrument.xml`

file format - see Appendix A.

When an instrument is loaded (see Genera User's Guide), all the information from this file is dynamically loaded into the program. An example of the XML property file for the plate washer is provided in Appendix B. The following XML tags are defined for the virtual instrument:

<**ObjectInfo**> contains the description and label of the instrument.

<**Position**> translates the 3D instrument model along the X-axis, Y-axis and Z-axis by the specified distance.

<**Rotation**> rotates the 3D instrument model around the X-axis, Y-axis and Z-axis by the specified angle.

<**DriverClass**> specifies the class name for the driver which is used to communicate with the physical instrument.

<**SimulationClass**> specifies the class name for the 3D simulation model of the physical instrument.

<**PhysicalProperties**> specifies the overall length, width, height, and weight of the instrument.

<**AdministrativeProperties**> is comprized of the following XML tags:

 <**Manufacturer**> instrument manufacturer name.

 <**ModelNumber**> specifies the model of the instrument.

 <**SerialNumber**> identifies the serial number of the instrument. Each instrument must have a unique serial number.

 <**UpdateAddress**> specifies an address of an internet site which contains the instrument update information.

 <**InformationAddress**> specifies an address of an internet site which contains the general instrument information.

<**VirtualDevice**> specifies the virtual device contained within the virtual instrument.

<**Holder**> describes each resource or container holder within the virtual instrument. It is comprised of the following XML tags:

<**ObjectInfo**> contains the description and label of the holder.

<**Capacity**> specifies the capacity of the holder.

Each virtual instrument contains a set of virtual devices. The following XML tags are defined for each virtual device.

<**ObjectInfo**> contains the description and label of the virtual device.

<**Command**> describes each command within the virtual device. It is comprised of the following XML tags:

<**ObjectInfo**> contains the description and label of the command.

<**CommandName**> specifies the command name.

<**CommandArgument**> describes each command argument. It is comprised of the following XML tags:

<**Description**> describes the command argument.

<**ArgumentName**> specifies the argument name.

<**ArgumentRange**> specifies the argument range, if it is required.

<**DefaultValue**> default value for the argument, if it exists.

<**ReturnArgument**> describes an optional return argument for the command. It is comprised of the following XML tags:

<**Description**> describes the return argument.

<**ArgumentName**> specifies the argument name.

<**Alarm**> describes each alarm within the virtual device. It is comprised of the following XML tags:

<**ObjectInfo**> contains the description and label of the alarm.

<**AlarmName**> specifies the alarm name.

<**AlarmArgument**> describes each alarm argument. It is comprised of the following XML tags:

<**Description**> describes the alarm argument.

<**ArgumentName**> specifies the argument name.

<**ArgumentRange**> specifies the argument range, if it is required.

<**DefaultValue**> default value for the argument, if it exists.

2.3 Driver Development

After creating the proper directory structure for the instrument driver, it's time to implement the communication with the physical instrument. The software module which communicates with the physical instrument is referred to as the **driver**. To integrate the driver with the Genera framework, you must do the following:

1. Include the driver class name in the `<DriverClass>` tag within the XML file.
2. Driver must have a default constructor (i.e. Java constructor with no arguments).
3. Driver must implement the `ControllableDriver` and `ControllableInstrument` interfaces.
4. Include the driver classes in the `C:\retisoft\lib\genera.jar` file.

ControllableDriver is a base interface for each driver.

ControllableInstrument contains the methods to connect, disconnect and send commands to the physical instrument.

2.3.1 ControllableDriver Interface

The methods in the `ControllableDriver` interface are the base methods for each driver. The `initDriver`, `getInstrumentID`, `getPropertyFile` and `getDebug` methods are already implemented by the `com.retisoft.java.genera.driver.BaseDriver` class. Thus, your new driver should extend the `BaseDriver` class and implement the `closeConnection` method. The following is the listing of the `ControllableDriver` methods:

```
package com.retisoft.java.genera.drivers;

// ReTiSoft packages
import com.retisoft.java.lecis.instruments.adt.InstrumentID;

/*****
** Interface for the Genera driver.<P>
**
** Copyright (c) ReTiSoft Inc.
** @author Paul Rodziewicz
** @version 1.0
*****/
public interface ControllableDriver {
```

```

/*****
 * Initialize the instrument driver.
 *
 * @param instrumentID  instrument identifier for the driver
 * @exception DriverInitException  could not initialize the driver
 *****/
public void initDriver(InstrumentID instrumentID) throws DriverInitException;

/*****
 * Retrieve the instrument identifier for the driver.
 *
 * @return  instrument identifier for the driver
 *****/
public InstrumentID getInstrumentID();

/*****
 * Retrieve the configuration file which contains specific communication
 * parameters for the driver.
 *
 * @return  configuration file which contains specific communication
 * parameters for the driver.
 *****/
public String getPropertyFile();

/*****
 * Retrieve the debug flag for the driver. The DEBUG flag
 * must be specified within the configuration file for the driver.
 *
 * @return  debug flag for the driver
 *****/
public boolean getDebug();

/*****
 * Close the connection with the physical instrument.
 * The method must be implemented by the driver.
 *
 * @exception DriverCloseException  could not close the connection
 * with the physical instrument
 *****/
public void closeConnection() throws DriverCloseException;
}

```

2.3.2 ControllableInstrument Interface

The methods in the `ControllableInstrument` interface establish connection with the physical instrument. Commands are then sent to the physical instrument using the communication technology (i.e. RS-232, Active-X, CORBA, etc) and protocol (i.e. LECIS, SECS, or proprietary protocol) of the physical instrument. The `getInstrumentState` and `getDeviceState` are already implemented by the `com.retisoft.java.genera.driver.BaseDriver` class. Thus, your new driver should extend the `BaseDriver` class and implement the `connect`, `disconnect`, `initialize`, `eStop` and `runOp` methods. These driver methods are automatically called by Genera when the corresponding commands are dispatched to the framework. The following is the listing of the `ControllableInstrument` methods:

```
package com.retisoft.java.genera.drivers;

// ReTiSoft packages
import com.retisoft.java.lecis.instruments.adt.InstrumentID;
import com.retisoft.java.lecis.instruments.commands.interfaces.MethodArgument;
import com.retisoft.java.lecis.instruments.statecharts.states.InstrumentState;
import com.retisoft.java.lecis.instruments.statecharts.states.VirtualDeviceState;
import com.retisoft.java.lecis.instruments.commands.exceptions.DeviceNotFoundException;

/*****
** Interface for the Genera instrument.<P>
**
** Copyright (c) ReTiSoft Inc.
** @author Paul Rodziewicz
** @version 1.0
*****/
public interface ControllableInstrument {

    /*****
    * Retrieve the current state of the virtual instrument.
    *
    * @return current state of the virtual instrument
    *****/
    public InstrumentState getInstrumentState();

    /*****
    * Retrieve the current state of the virtual device.
    *
    * @param deviceIndex      device index
    * @return current state of the virtual device
    * @exception DeviceNotFoundException device with the specified
    * index does not exist within the current virtual instrument
    *****/
}
```

```
*****/
public VirtualDeviceState getDeviceState(int deviceIndex)
    throws DeviceNotFoundException;

/*****
 * Establish the connection with the physical instrument.
 *
 * @exception DriverInitException    could not connect to the instrument
 *****/
public void connect() throws DriverInitException;

/*****
 * Terminate the connection with the physical instrument.
 *
 * @exception DriverCloseException  could not disconnect from the instrument
 *****/
public void disconnect() throws DriverCloseException;

/*****
 * Send the initialize command to the physical instrument.
 *
 * @exception CommandExecuteException  could not execute the command
 *****/
public void initialize() throws CommandExecuteException;

/*****
 * Send the emergency stop to the physical instrument.
 *
 * @exception CommandExecuteException  could not execute the command
 *****/
public void eStop() throws CommandExecuteException;

/*****
 * Send the specified command to the instrument.
 *
 * @param deviceIndex    device index for the command
 * @param commandName   command name
 * @exception CommandExecuteException  could not execute the command
 *****/
public void runOp(String deviceIndex, String commandName)
    throws CommandExecuteException;

/*****
 * Send the specified command to the instrument.
 *

```

```

* @param deviceIndex      device index for the command
* @param commandName     command name
* @param commandArguments command arguments
* @exception CommandExecuteException could not execute the command
*****/
public void runOp(String deviceIndex, String commandName,
                 MethodArgument[] commandArguments)
    throws CommandExecuteException;
}

```

2.4 Configuring Genera

To gain access to the new instrument from the Genera GUI, the user must also add a property line for the plugin to the `instruments` file. The following is the format of the property line for each plugin:

```
<instrument-model> = <plugin-location>
```

where:

instrument-model must be identical to the model number of the instrument retrieved from the administrative properties within the XML file.

plugin-location is the installation directory for the instrument plugin.

An example of the `instruments` file with four plugins is given below:

```

# Directory locations of instruments
Labman-BathCamera = C:/retisoft/labman-bathcamera/
Labman-Dissolution = C:/retisoft/labman-dissolution/
Lmux-Spectrometer = C:/retisoft/lmux-spectrometer/
PlateWasher-RS232 = C:/retisoft/platewasher-rs232/

```

If you installed Genera in the `C:\retisoft\genera-1.1.5` directory, the `instruments` property file would be located in `C:\retisoft\genera-1.1.5\properties\instruments`.

When an instrument is loaded by Genera, the model of the instrument is located within the `instruments` property file and the corresponding installation directory for the instrument plugin is retrieved.

Chapter 3

Fixture Plugin

The Genera framework uses fixture plugins to represent non-automated objects which may hold containers and resources. Some fixture examples are hotels, laboratory tables, shelving units, etc. To integrate a new fixture into our framework, you should know the following:

1. Directory structure for the fixture (refer to Section 3.1)
2. How to configure Genera to recognize the fixture (refer to Section 3.2)

3.1 Fixture Directory

In this section, we describe the directory structure required for each fixture.

Each fixture must contain two directory folders (**images** and **xml**) which must be located in the **genera** directory of the corresponding fixture. If you wish to install the fixture in the `C:\retisoft\crs-table` directory, then the `C:\retisoft\crs-table\genera` directory must contain two of the above mentioned directory folders.



Figure 3.1: Fixture Directory

In the following sections we describe the contents of each of these directory folders.

3.1.1 Images

The images folder for the Genera fixture must contain the picture of the fixture. The picture file must satisfy the following conditions:

file name - picture.jpg

file format - JPEG file format

picture dimensions - 150x150 pixels

The **Fixture List** window, depicted in Figure 3.2, displays all the fixtures which can be loaded, as well as the image of each fixture. A single-click on the name of the fixture displays the picture.jpg file for the corresponding fixture.



Figure 3.2: Fixture List Window

3.1.2 XML File

The xml folder for the Genera fixture must contain the XML file describing the fixture. The XML file must satisfy the following conditions:

file name - fixture.xml

file format - see Appendix C.

When a fixture is loaded (see Genera User's Guide), all the information from this file is dynamically loaded into the program. An example of the XML property file for the CRS table is provided in Appendix D. The following XML tags are defined for the fixture:

<**ObjectInfo**> contains the description and label of the fixture.

<**Position**> translates the 3D fixture model along the X-axis, Y-axis and Z-axis by the specified distance.

<**Rotation**> rotates the 3D fixture model around the X-axis, Y-axis and Z-axis by the specified angle.

<**SimulationClass**> specifies the class name for the 3D simulation model of the physical fixture.

<**PhysicalProperties**> specifies the length, width, height, and weight of the fixture.

<**AdministrativeProperties**> is comprized of the following XML tags:

<**Manufacturer**> fixture manufacturer name.

<**ModelNumber**> specifies the model of the fixture.

<**SerialNumber**> identifies the serial number of the fixture. Each fixture must have a unique serial number.

<**UpdateAddress**> specifies an address of an internet site which contains the fixture update information.

<**InformationAddress**> specifies an address of an internet site which contains the general fixture information.

<**Holder**> describes each resource or container holder within the fixture. It is comprized of the following XML tags:

<**ObjectInfo**> contains the description and label of the holder.

<**Capacity**> specifies the capacity of the holder.

3.2 Configuring Genera

To gain access to the new fixture from the Genera GUI, the user must also add a property line for the plugin to the `fixtures` file. The following is the format of the property line for each plugin:

```
<fixture-model> = <plugin-location>
```

where:

fixture-model must be identical to the model number of the fixture retrieved from the administrative properties within the XML file.

plugin-location is the installation directory for the fixture plugin.

An example of the `fixtures` file with two plugins is given below:

```
# Directory locations of fixtures
CrsTable = C:/retisoft/crs-table/
SagianHotel = C:/retisoft/sagian-hotel/
```

If Genera was installed in the `C:\retisoft\genera-1.1.5` directory, the `fixtures` property file would be located in `C:\retisoft\genera-1.1.5\properties\fixtures`.

When a fixture is loaded by Genera, the model of the fixture is located within the `fixtures` property file and the corresponding installation directory for the fixture plugin is retrieved.

Chapter 4

Container Plugin

Containers represent trays and subtrays (e.g. sample vial tray, fraction vial subtray, microplate subtray, etc) which are used in a laboratory process. The containers are always placed in container holders with limited capacity. To integrate a new container into our framework, you should know the following:

1. Directory structure for the container (refer to Section 4.1)
2. How to configure Genera to recognize the container (refer to Section 4.2)

4.1 Container Directory

In this section, we describe the directory structure required for each container.

Each container must contain two directory folders (**images** and **xml**) which must be located in the **genera** directory of the corresponding container. If you wish to install the container in the `C:\retisoft\vial-subtray` directory, then the `C:\retisoft\vial-subtray\genera` directory must contain two of the above mentioned directory folders.



Figure 4.1: Container Directory

In the following sections we describe the contents of each of these directory folders.

4.1.1 Images

The images folder for the Genera container must contain the picture of the container. The picture file must satisfy the following conditions:

file name - `picture.jpg`

file format - JPEG file format

picture dimensions - 150x150 pixels

The **Container List** window, depicted in Figure 4.2, displays all the containers which can be loaded, as well as the image of each container. A single-click on the name of the container displays the `picture.jpg` file for the corresponding container.



Figure 4.2: Container List Window

4.1.2 XML File

The xml folder for the Genera container must contain the XML file describing the container. The XML file must satisfy the following conditions:

file name - `container.xml`

file format - see Appendix E.

When a container is loaded (see Genera User's Guide), all the information from this file is dynamically loaded into the program. An example of the XML property file for the vial subtray is provided in Appendix F. The following XML tags are defined for the container:

<**ObjectInfo**> contains the description and label of the container.

<**SimulationClass**> specifies the class name for the 3D simulation model of the physical container.

<**PhysicalProperties**> specifies the length, width, height, and weight of the container.

<**AdministrativeProperties**> is comprized of the following XML tags:

<**Manufacturer**> container manufacturer name.

<**ModelNumber**> specifies the model of the container.

<**SerialNumber**> identifies the serial number of the container.

<**UpdateAddress**> specifies an address of an internet site which contains the container update information.

<**InformationAddress**> specifies an address of an internet site which contains the general container information.

<**Holder**> describes each resource or container holder within the container. It is comprized of the following XML tags:

<**ObjectInfo**> contains the description and label of the holder.

<**Capacity**> specifies the capacity of the holder.

4.2 Configuring Genera

To gain access to the new container from the Genera GUI, the user must also add a property line for the plugin to the `containers` file. The following is the format of the property line for each plugin:

```
<container-model> = <plugin-location>
```

where:

container-model must be identical to the model number of the container retrieved from the administrative properties within the XML file.

plugin-location is the installation directory for the container plugin.

An example of the `containers` file with four plugins is given below:

```
# Directory locations of containers
Stacker-Tray = C:/retisoft/stacker-tray/
Filter-Subtray = C:/retisoft/filter-subtray/
Fraction-Subtray = C:/retisoft/fraction-subtray/
Vial-Subtray = C:/retisoft/vial-subtray/
```

If Genera was installed in the `C:\retisoft\genera-1.1.5` directory, the `containers` property file would be located in `C:\retisoft\genera-1.1.5\properties\containers`.

When a container is loaded by Genera, the model of the container is located within the `containers` property file and the corresponding installation directory for the container plugin is retrieved.

Chapter 5

Resource Plugin

Resources represent items (e.g. sample vials, microplates and microplate tips, baskets, etc) which are used in a laboratory process. The resources are placed in resource holders with limited or unlimited capacity. An example of a resource holder with limited capacity is a robotic arm gripper which can carry one sample vial, or a hotel shelf which can store two microplates. An example of a resource holder with unlimited capacity is a garbage bin which allows the automated system to discard an unlimited number of microplate tips. To integrate a new resource into our framework, you should know the following:

1. Directory structure for the resource (refer to Section 5.1)
2. How to configure Genera to recognize the resource (refer to Section 5.2)

5.1 Resource Directory

In this section, we describe the directory structure required for each resource.

Each resource must contain two directory folders (**images** and **xml**) which must be located in the **genera** directory of the corresponding resource. If you wish to install the resource in the C:\retisoft\sample-vial directory, then the C:\retisoft\sample-vial\genera directory must contain two of the above mentioned directory folders.



Figure 5.1: Resource Directory

In the following sections we describe the contents of each of these directory folders.

5.1.1 Images

The images folder for the Genera resource must contain the picture of the resource. The picture file must satisfy the following conditions:

file name - picture.jpg

file format - JPEG file format

picture dimensions - 150x150 pixels

The **Resource List** window, depicted in Figure 5.2, displays all the resources which can be loaded, as well as the image of each resource. A single-click on the name of the resource displays the picture.jpg file for the corresponding resource.

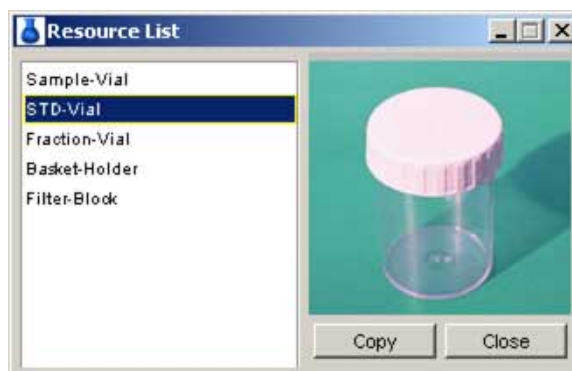


Figure 5.2: Resource List Window

5.1.2 XML File

The xml folder for the Genera resource must contain the XML file describing the resource. The XML file must satisfy the following conditions:

file name - resource.xml

file format - see Appendix G.

When a resource is loaded (see Genera User's Guide), all the information from this file is dynamically loaded into the program. An example of the XML property file for the vial subtray is provided in Appendix H. The following XML tags are defined for the resource:

<**ObjectInfo**> contains the description and label of the resource.

<**SimulationClass**> specifies the class name for the 3D simulation model of the physical resource.

<**PhysicalProperties**> specifies the length, width, height, and weight of the resource.

<**AdministrativeProperties**> is comprised of the following XML tags:

<**Manufacturer**> resource manufacturer name.

<**ModelNumber**> specifies the model of the resource.

<**SerialNumber**> identifies the serial number of the resource.

<**UpdateAddress**> specifies an address of an internet site which contains the resource update information.

<**InformationAddress**> specifies an address of an internet site which contains the general resource information.

5.2 Configuring Genera

To gain access to the new resource from the Genera GUI, the user must also add a property line for the plugin to the resources file. The following is the format of the property line for each plugin:

```
<resource-model> = <plugin-location>
```

where:

resource-model must be identical to the model number of the resource retrieved from the administrative properties within the XML file.

plugin-location is the installation directory for the resource plugin.

An example of the `resources` file with five plugins is given below:

```
# Directory locations of resources
Basket-Holder = C:/retisoft/basket-holder/
Filter-Block = C:/retisoft/filter-block/
Fraction-Vial = C:/retisoft/fraction-vial/
Sample-Vial = C:/retisoft/sample-vial/
STD-Vial = C:/retisoft/std-vial/
```

If Genera was installed in the `C:\retisoft\genera-1.1.5` directory, the `resources` property file would be located in `C:\retisoft\genera-1.1.5\properties\resources`.

When a resource is loaded by Genera, the model of the resource is located within the `resources` property file and the corresponding installation directory for the resource plugin is retrieved.

Appendix A

Instrument DTD

```
<!--=====-->
<!-- Common information for the instrument, device and -->
<!-- the resource holder -->
<!--=====-->
<!ELEMENT ObjectInfo (Description,
                      Label)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT Label (#PCDATA)>

<!--=====-->
<!--          Virtual instrument information -->
<!--=====-->
<!ELEMENT Instrument (ObjectInfo,
                     Position,
                     Rotation,
                     DriverClass,
                     SimulationClass,
                     PhysicalProperties,
                     AdministrativeProperties,
                     VirtualDevice+,
                     Holder*)>

<!ELEMENT DriverClass (#PCDATA)>
<!ELEMENT SimulationClass (#PCDATA)>
<!ELEMENT PhysicalProperties (Length, Width, Height, Weight)>
<!ELEMENT Length (#PCDATA)>
<!ATTLIST Length data-type (Integer | Float) #REQUIRED>
<!ELEMENT Width (#PCDATA)>
<!ATTLIST Width data-type (Integer | Float) #REQUIRED>
<!ELEMENT Height (#PCDATA)>
```

```

<!ATTLIST Height data-type (Integer | Float) #REQUIRED>
<!ELEMENT Weight (#PCDATA)>
<!ATTLIST Weight data-type (Integer) #REQUIRED>

<!ELEMENT AdministrativeProperties (Manufacturer, ModelNumber,
                                   SerialNumber, UpdateAddress,
                                   InformationAddress)>

<!ELEMENT Manufacturer (#PCDATA)>
<!ELEMENT ModelNumber (#PCDATA)>
<!ELEMENT SerialNumber (#PCDATA)>
<!ELEMENT UpdateAddress (#PCDATA)>
<!ELEMENT InformationAddress (#PCDATA)>

<!------->
<!--           Virtual device information           -->
<!------->
<!ELEMENT VirtualDevice (ObjectInfo,
                        Command+,
                        Alarm*)>

<!------->
<!--           Command information           -->
<!------->
<!ELEMENT Command (ObjectInfo,
                  CommandName,
                  CommandArgument*,
                  ReturnArgument*)>

<!ELEMENT CommandName (#PCDATA)>

<!ELEMENT CommandArgument (Description, ArgumentName,
                          ArgumentRange?, DefaultValue?)>
<!ATTLIST CommandArgument argument-type (Integer|Float|String|Boolean) #REQUIRED>
<!ELEMENT ArgumentName (#PCDATA)>
<!ELEMENT ArgumentRange (LowLimit?, HighLimit?)>
<!ELEMENT LowLimit (#PCDATA)>
<!ELEMENT HighLimit (#PCDATA)>
<!ELEMENT DefaultValue (#PCDATA)>

<!ELEMENT ReturnArgument (Description, ArgumentName)>
<!ATTLIST ReturnArgument argument-type (Integer|Float|String|Boolean) #REQUIRED>

<!------->
<!--           Alarm information           -->
<!------->

```

```
<!ELEMENT Alarm (ObjectInfo,
                 AlarmName,
                 AlarmArgument*)>
<!ELEMENT AlarmName (#PCDATA)>
<!ELEMENT AlarmArgument (Description, ArgumentName,
                         ArgumentRange?, DefaultValue?)>
<!ATTLIST AlarmArgument argument-type (Integer|Float|String|Boolean) #REQUIRED>

<!--=====
<!--           Holder information           -->
<!--=====
<!ELEMENT Holder (ObjectInfo,
                 Capacity)>
<!ATTLIST Holder holder-type (Container | Resource) #REQUIRED>

<!ELEMENT Capacity (#PCDATA)>
<!ATTLIST Capacity data-type (Integer) #REQUIRED>

<!--=====
<!--           Point position           -->
<!--=====
<!ELEMENT Position (X_Translation, Y_Translation, Z_Translation)>

<!ELEMENT X_Translation (#PCDATA)>
<!ATTLIST X_Translation data-type (Integer) #REQUIRED>
<!ELEMENT Y_Translation (#PCDATA)>
<!ATTLIST Y_Translation data-type (Integer) #REQUIRED>
<!ELEMENT Z_Translation (#PCDATA)>
<!ATTLIST Z_Translation data-type (Integer) #REQUIRED>

<!--=====
<!--           Axis rotations           -->
<!--=====
<!ELEMENT Rotation (X_Rotation, Y_Rotation, Z_Rotation)>

<!ELEMENT X_Rotation (#PCDATA)>
<!ATTLIST X_Rotation data-type (Degrees) #REQUIRED>
<!ELEMENT Y_Rotation (#PCDATA)>
<!ATTLIST Y_Rotation data-type (Degrees) #REQUIRED>
<!ELEMENT Z_Rotation (#PCDATA)>
<!ATTLIST Z_Rotation data-type (Degrees) #REQUIRED>
```

Appendix B

Instrument XML

```
<!--=====-->
<!--          Instrument information          -->
<!--=====-->
<Instrument>
  <ObjectInfo>
    <Description>Plate washer which uses RS-232 communication.</Description>
    <Label>Plate Washer - RS232</Label>
  </ObjectInfo>
  <Position>
    <X_Translation data-type="Integer">0</X_Translation>
    <Y_Translation data-type="Integer">0</Y_Translation>
    <Z_Translation data-type="Integer">0</Z_Translation>
  </Position>
  <Rotation>
    <X_Rotation data-type="Degrees">0</X_Rotation>
    <Y_Rotation data-type="Degrees">0</Y_Rotation>
    <Z_Rotation data-type="Degrees">0</Z_Rotation>
  </Rotation>
  <DriverClass>com.retisoft.java.platewasher_rs232.genera.Driver</DriverClass>
  <SimulationClass>com.retisoft.java.platewasher_rs232.model.Model</SimulationClass>
  <PhysicalProperties>
    <Length data-type="Integer">350</Length>
    <Width data-type="Integer">350</Width>
    <Height data-type="Integer">150</Height>
    <Weight data-type="Integer">0</Weight>
  </PhysicalProperties>
  <AdministrativeProperties>
    <Manufacturer>ReTiSoft Inc.</Manufacturer>
    <ModelNumber>PlateWasher</ModelNumber>
    <SerialNumber>PlateWasher-01</SerialNumber>
  </AdministrativeProperties>
</Instrument>
```

```
<UpdateAddress>www.retisoft.ca</UpdateAddress>
<InformationAddress>www.retisoft.ca</InformationAddress>
</AdministrativeProperties>

<!--=====-->
<!--           Virtual device information           -->
<!--=====-->
<VirtualDevice>
  <ObjectInfo>
    <Description>Soaks, aspirates and washes microplates.</Description>
    <Label>Plate Washer</Label>
  </ObjectInfo>

  <!--=====-->
  <!--           Aspirate           -->
  <!--=====-->
  <Command>
    <ObjectInfo>
      <Description>Aspirate the liquid from a microplates.</Description>
      <Label>Aspirate</Label>
    </ObjectInfo>
    <CommandName>Aspirate</CommandName>

    <!--=====-->
    <!--           Command arguments           -->
    <!--=====-->
    <CommandArgument argument-type="Integer">
      <Description>Specifies the duration of the aspiration.</Description>
      <ArgumentName>time</ArgumentName>
      <ArgumentRange>
        <LowLimit>1</LowLimit>
        <HighLimit>20</HighLimit>
      </ArgumentRange>
      <DefaultValue>4</DefaultValue>
    </CommandArgument>

    <CommandArgument argument-type="Integer">
      <Description>Specifies the speed of the aspiration.</Description>
      <ArgumentName>aspirateSpeed</ArgumentName>
      <ArgumentRange>
        <LowLimit>1</LowLimit>
        <HighLimit>20</HighLimit>
      </ArgumentRange>
      <DefaultValue>10</DefaultValue>
    </CommandArgument>
```

```
</Command>

<!--=====-->
<!--          Dispense          -->
<!--=====-->
<Command>
  <ObjectInfo>
    <Description>Dispense the liquid to a microplate from the
      specified channel.
    </Description>
    <Label>Dispense</Label>
  </ObjectInfo>
  <CommandName>Dispense</CommandName>

  <!--=====-->
  <!--          Command arguments          -->
  <!--=====-->
  <CommandArgument argument-type="Integer">
    <Description>Specifies a channel for dispensing.</Description>
    <ArgumentName>channel</ArgumentName>
    <ArgumentRange>
      <LowLimit>1</LowLimit>
      <HighLimit>4</HighLimit>
    </ArgumentRange>
    <DefaultValue>1</DefaultValue>
  </CommandArgument>

  <CommandArgument argument-type="Float">
    <Description>Dispensing volume.</Description>
    <ArgumentName>volume</ArgumentName>
    <ArgumentRange>
      <LowLimit>50.0</LowLimit>
      <HighLimit>300.0</HighLimit>
    </ArgumentRange>
    <DefaultValue>300.0</DefaultValue>
  </CommandArgument>

  <CommandArgument argument-type="Integer">
    <Description>Specifies the speed of the dispensing.</Description>
    <ArgumentName>dispenseSpeed</ArgumentName>
    <ArgumentRange>
      <LowLimit>1</LowLimit>
      <HighLimit>20</HighLimit>
    </ArgumentRange>
    <DefaultValue>10</DefaultValue>
```

```

    </CommandArgument>
  </Command>

  <!--=====-->
  <!--          Wash          -->
  <!--=====-->
  <Command>
    <ObjectInfo>
      <Description>Wash a microplate by aspirating the existing
        liquid and then dispensing another liquid from the
        specified channel.
      </Description>
      <Label>Wash</Label>
    </ObjectInfo>
    <CommandName>Wash</CommandName>

    <!--=====-->
    <!--          Command arguments          -->
    <!--=====-->
    <CommandArgument argument-type="Integer">
      <Description>Specifies the duration of the aspiration.</Description>
      <ArgumentName>time</ArgumentName>
      <ArgumentRange>
        <LowLimit>1</LowLimit>
        <HighLimit>20</HighLimit>
      </ArgumentRange>
      <DefaultValue>4</DefaultValue>
    </CommandArgument>

    <CommandArgument argument-type="Integer">
      <Description>Specifies the speed of the aspiration.</Description>
      <ArgumentName>aspirateSpeed</ArgumentName>
      <ArgumentRange>
        <LowLimit>1</LowLimit>
        <HighLimit>20</HighLimit>
      </ArgumentRange>
      <DefaultValue>10</DefaultValue>
    </CommandArgument>

    <CommandArgument argument-type="Integer">
      <Description>Specifies a channel for dispensing.</Description>
      <ArgumentName>channel</ArgumentName>
      <ArgumentRange>
        <LowLimit>1</LowLimit>
        <HighLimit>4</HighLimit>

```

```
    </ArgumentRange>
    <DefaultValue>1</DefaultValue>
  </CommandArgument>

  <CommandArgument argument-type="Float">
    <Description>Dispensing volume.</Description>
    <ArgumentName>volume</ArgumentName>
    <ArgumentRange>
      <LowLimit>50.0</LowLimit>
      <HighLimit>300.0</HighLimit>
    </ArgumentRange>
    <DefaultValue>300.0</DefaultValue>
  </CommandArgument>

  <CommandArgument argument-type="Integer">
    <Description>Specifies the speed of the dispensing.</Description>
    <ArgumentName>dispenseSpeed</ArgumentName>
    <ArgumentRange>
      <LowLimit>1</LowLimit>
      <HighLimit>20</HighLimit>
    </ArgumentRange>
    <DefaultValue>10</DefaultValue>
  </CommandArgument>

</Command>

<!------->
<!--           Soak           -->
<!------->
<Command>
  <ObjectInfo>
    <Description>Soaks a microplate.</Description>
    <Label>Soak</Label>
  </ObjectInfo>
  <CommandName>Soak</CommandName>

  <!------->
  <!--           Command arguments           -->
  <!------->
  <CommandArgument argument-type="Integer">
    <Description>Specifies the duration of soaking.</Description>
    <ArgumentName>time</ArgumentName>
    <ArgumentRange>
      <LowLimit>1</LowLimit>
      <HighLimit>60</HighLimit>
  </CommandArgument>

```

```
        </ArgumentRange>
        <DefaultValue>1</DefaultValue>
    </CommandArgument>
</Command>

<!--=====-->
<!--          ReadBarcode          -->
<!--=====-->
<Command>
    <ObjectInfo>
        <Description>Read the bar code from a microplate.</Description>
        <Label>Read Barcode</Label>
    </ObjectInfo>
    <CommandName>ReadBarcode</CommandName>
</Command>

<!--=====-->
<!--          Alarms          -->
<!--=====-->
<Alarm>
    <ObjectInfo>
        <Description>No plate is sensed on the washer plate handler.</Description>
        <Label>No Plate Sensed</Label>
    </ObjectInfo>
    <AlarmName>NoPlateSensed</AlarmName>
</Alarm>

<Alarm>
    <ObjectInfo>
        <Description>Liquid overflow occurred on the plate during
            dispensing or washing cycle.
        </Description>
        <Label>Plate Overflow</Label>
    </ObjectInfo>
    <AlarmName>PlateOverflow</AlarmName>
</Alarm>

<Alarm>
    <ObjectInfo>
        <Description>Pump is defective during dispensing or washing cycle.</Description>
        <Label>Dispense Pump Error</Label>
    </ObjectInfo>
    <AlarmName>DispensePumpError</AlarmName>
</Alarm>
```

```
</VirtualDevice>  
</Instrument>
```

Appendix C

Fixture DTD

```
<!--=====-->
<!--          Common information for fixtures          -->
<!--=====-->
<!ELEMENT ObjectInfo (Description,
                      Label)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT Label (#PCDATA)>

<!--=====-->
<!--          Fixture information          -->
<!--=====-->
<!ELEMENT Fixture (ObjectInfo,
                  Position,
                  Rotation,
                  SimulationClass,
                  PhysicalProperties,
                  AdministrativeProperties,
                  Holder*)>

<!ELEMENT SimulationClass (#PCDATA)>
<!ELEMENT PhysicalProperties (Length, Width, Height, Weight)>
<!ELEMENT Length (#PCDATA)>
<!ATTLIST Length data-type (Integer | Float) #REQUIRED>
<!ELEMENT Width (#PCDATA)>
<!ATTLIST Width data-type (Integer | Float) #REQUIRED>
<!ELEMENT Height (#PCDATA)>
<!ATTLIST Height data-type (Integer | Float) #REQUIRED>
<!ELEMENT Weight (#PCDATA)>
<!ATTLIST Weight data-type (Integer) #REQUIRED>
```

```
<!ELEMENT AdministrativeProperties (Manufacturer, ModelNumber,
                                   SerialNumber, UpdateAddress,
                                   InformationAddress)>

<!ELEMENT Manufacturer (#PCDATA)>
<!ELEMENT ModelNumber (#PCDATA)>
<!ELEMENT SerialNumber (#PCDATA)>
<!ELEMENT UpdateAddress (#PCDATA)>
<!ELEMENT InformationAddress (#PCDATA)>

<!--=====
<!--           Holder information           -->
<!--=====
<!ELEMENT Holder (ObjectInfo,
                  Capacity)>
<ATTLIST Holder holder-type (Container | Resource) #REQUIRED>

<!ELEMENT Capacity (#PCDATA)>
<ATTLIST Capacity data-type (Integer) #REQUIRED>

<!--=====
<!--           Point position           -->
<!--=====
<!ELEMENT Position (X_Translation, Y_Translation, Z_Translation)>

<!ELEMENT X_Translation (#PCDATA)>
<ATTLIST X_Translation data-type (Integer) #REQUIRED>
<!ELEMENT Y_Translation (#PCDATA)>
<ATTLIST Y_Translation data-type (Integer) #REQUIRED>
<!ELEMENT Z_Translation (#PCDATA)>
<ATTLIST Z_Translation data-type (Integer) #REQUIRED>

<!--=====
<!--           Axis rotations           -->
<!--=====
<!ELEMENT Rotation (X_Rotation, Y_Rotation, Z_Rotation)>

<!ELEMENT X_Rotation (#PCDATA)>
<ATTLIST X_Rotation data-type (Degrees) #REQUIRED>
<!ELEMENT Y_Rotation (#PCDATA)>
<ATTLIST Y_Rotation data-type (Degrees) #REQUIRED>
<!ELEMENT Z_Rotation (#PCDATA)>
<ATTLIST Z_Rotation data-type (Degrees) #REQUIRED>
```

Appendix D

Fixture XML

```
<!--=====-->
<!--           Fixture information           -->
<!--=====-->
<Fixture>
  <ObjectInfo>
    <Description>The table accommodates CRS track robots and
      is available in 5 lengths from 1 to 5 meters.
    </Description>
    <Label>CRS Table</Label>
  </ObjectInfo>
  <Position>
    <X_Translation data-type="Integer">0</X_Translation>
    <Y_Translation data-type="Integer">0</Y_Translation>
    <Z_Translation data-type="Integer">0</Z_Translation>
  </Position>
  <Rotation>
    <X_Rotation data-type="Degrees">0</X_Rotation>
    <Y_Rotation data-type="Degrees">0</Y_Rotation>
    <Z_Rotation data-type="Degrees">0</Z_Rotation>
  </Rotation>
  <SimulationClass>com.retisoft.java.crs_table.Model</SimulationClass>
  <PhysicalProperties>
    <Length data-type="Integer">244</Length>
    <Width data-type="Integer">200</Width>
    <Height data-type="Integer">86</Height>
    <Weight data-type="Integer">10</Weight>
  </PhysicalProperties>
  <AdministrativeProperties>
    <Manufacturer>CRS Robotics Corporation</Manufacturer>
    <ModelNumber>L5</ModelNumber>
  </AdministrativeProperties>
</Fixture>
```

```
<SerialNumber>CRS-L5-01</SerialNumber>  
<UpdateAddress>www.crsrobotics.com</UpdateAddress>  
<InformationAddress>www.retisoft.ca</InformationAddress>  
</AdministrativeProperties>  
</Fixture>
```

Appendix E

Container DTD

```
<!--=====-->
<!--          Common information for containers          -->
<!--=====-->
<!ELEMENT ObjectInfo (Description,
                      Label)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT Label (#PCDATA)>

<!--=====-->
<!--          Container information          -->
<!--=====-->
<!ELEMENT Container (ObjectInfo,
                    SimulationClass,
                    PhysicalProperties,
                    AdministrativeProperties,
                    Holder+)>

<!ELEMENT SimulationClass (#PCDATA)>
<!ELEMENT PhysicalProperties (Length, Width, Height, Weight)>
<!ELEMENT Length (#PCDATA)>
<!ATTLIST Length data-type (Integer | Float) #REQUIRED>
<!ELEMENT Width (#PCDATA)>
<!ATTLIST Width data-type (Integer | Float) #REQUIRED>
<!ELEMENT Height (#PCDATA)>
<!ATTLIST Height data-type (Integer | Float) #REQUIRED>
<!ELEMENT Weight (#PCDATA)>
<!ATTLIST Weight data-type (Integer) #REQUIRED>

<!ELEMENT AdministrativeProperties (Manufacturer, ModelNumber,
                                   SerialNumber, UpdateAddress,
```

```

                                InformationAddress)>
<!ELEMENT Manufacturer (#PCDATA)>
<!ELEMENT ModelNumber (#PCDATA)>
<!ELEMENT SerialNumber (#PCDATA)>
<!ELEMENT UpdateAddress (#PCDATA)>
<!ELEMENT InformationAddress (#PCDATA)>

<!--=====
<!--                Holder information                -->
<!--=====
<!ELEMENT Holder (ObjectInfo,
                  Capacity)>
<ATTLIST Holder holder-type (Container | Resource) #REQUIRED>
<!ELEMENT Capacity (#PCDATA)>
<ATTLIST Capacity data-type (Integer) #REQUIRED>
```

Appendix F

Container XML

```
<!--=====>
<!--           Container information           -->
<!--=====>
<Container>
  <ObjectInfo>
    <Description>The subtray provides resource holders for STDs,
      basket holders and sample vials.
    </Description>
    <Label>Vial Subtray</Label>
  </ObjectInfo>
  <SimulationClass>com.retisoft.java.vial_subtray.Model</SimulationClass>
  <PhysicalProperties>
    <Length data-type="Integer">300</Length>
    <Width data-type="Integer">132</Width>
    <Height data-type="Integer">30</Height>
    <Weight data-type="Integer">0</Weight>
  </PhysicalProperties>
  <AdministrativeProperties>
    <Manufacturer>Labman Automation Limited</Manufacturer>
    <ModelNumber>Vial Subtray</ModelNumber>
    <SerialNumber>VialSubtray-17</SerialNumber>
    <UpdateAddress>www.labman.co.uk</UpdateAddress>
    <InformationAddress>www.labman.co.uk</InformationAddress>
  </AdministrativeProperties>

  <!--=====>
  <!--           Resource holder information           -->
  <!--=====>
  <Holder holder-type="Resource">
    <ObjectInfo>
```

```
    <Description>Resource holder for an STD.</Description>
    <Label>STD Holder 1</Label>
  </ObjectInfo>
  <Capacity data-type="Integer">1</Capacity>
</Holder>

<Holder holder-type="Resource">
  <ObjectInfo>
    <Description>Resource holder for an STD.</Description>
    <Label>STD Holder 2</Label>
  </ObjectInfo>
  <Capacity data-type="Integer">1</Capacity>
</Holder>

<Holder holder-type="Resource">
  <ObjectInfo>
    <Description>Resource holder for an STD.</Description>
    <Label>STD Holder 3</Label>
  </ObjectInfo>
  <Capacity data-type="Integer">1</Capacity>
</Holder>

<Holder holder-type="Resource">
  <ObjectInfo>
    <Description>Resource holder for a basket holder.</Description>
    <Label>Basket Holder 1</Label>
  </ObjectInfo>
  <Capacity data-type="Integer">1</Capacity>
</Holder>

<Holder holder-type="Resource">
  <ObjectInfo>
    <Description>Resource holder for a basket holder.</Description>
    <Label>Basket Holder 2</Label>
  </ObjectInfo>
  <Capacity data-type="Integer">1</Capacity>
</Holder>

<Holder holder-type="Resource">
  <ObjectInfo>
    <Description>Resource holder for a basket holder.</Description>
    <Label>Basket Holder 3</Label>
  </ObjectInfo>
  <Capacity data-type="Integer">1</Capacity>
</Holder>
```

```
<Holder holder-type="Resource">
  <ObjectInfo>
    <Description>Resource holder for a basket holder.</Description>
    <Label>Basket Holder 4</Label>
  </ObjectInfo>
  <Capacity data-type="Integer">1</Capacity>
</Holder>

<Holder holder-type="Resource">
  <ObjectInfo>
    <Description>Resource holder for a basket holder.</Description>
    <Label>Basket Holder 5</Label>
  </ObjectInfo>
  <Capacity data-type="Integer">1</Capacity>
</Holder>

<Holder holder-type="Resource">
  <ObjectInfo>
    <Description>Resource holder for a basket holder.</Description>
    <Label>Basket Holder 6</Label>
  </ObjectInfo>
  <Capacity data-type="Integer">1</Capacity>
</Holder>

<Holder holder-type="Resource">
  <ObjectInfo>
    <Description>Resource holder for a basket holder.</Description>
    <Label>Basket Holder 7</Label>
  </ObjectInfo>
  <Capacity data-type="Integer">1</Capacity>
</Holder>

<Holder holder-type="Resource">
  <ObjectInfo>
    <Description>Resource holder for a sample vial.</Description>
    <Label>Sample Vial Holder 1</Label>
  </ObjectInfo>
  <Capacity data-type="Integer">1</Capacity>
</Holder>

<Holder holder-type="Resource">
  <ObjectInfo>
    <Description>Resource holder for a sample vial.</Description>
    <Label>Sample Vial Holder 2</Label>
```

```
</ObjectInfo>
<Capacity data-type="Integer">1</Capacity>
</Holder>

<Holder holder-type="Resource">
  <ObjectInfo>
    <Description>Resource holder for a sample vial.</Description>
    <Label>Sample Vial Holder 3</Label>
  </ObjectInfo>
  <Capacity data-type="Integer">1</Capacity>
</Holder>

<Holder holder-type="Resource">
  <ObjectInfo>
    <Description>Resource holder for a sample vial.</Description>
    <Label>Sample Vial Holder 4</Label>
  </ObjectInfo>
  <Capacity data-type="Integer">1</Capacity>
</Holder>

<Holder holder-type="Resource">
  <ObjectInfo>
    <Description>Resource holder for a sample vial.</Description>
    <Label>Sample Vial Holder 5</Label>
  </ObjectInfo>
  <Capacity data-type="Integer">1</Capacity>
</Holder>

<Holder holder-type="Resource">
  <ObjectInfo>
    <Description>Resource holder for a sample vial.</Description>
    <Label>Sample Vial Holder 6</Label>
  </ObjectInfo>
  <Capacity data-type="Integer">1</Capacity>
</Holder>

<Holder holder-type="Resource">
  <ObjectInfo>
    <Description>Resource holder for a sample vial.</Description>
    <Label>Sample Vial Holder 7</Label>
  </ObjectInfo>
  <Capacity data-type="Integer">1</Capacity>
</Holder>

</Container>
```

Appendix G

Resource DTD

```
<!--=====-->
<!--          Common information for resources          -->
<!--=====-->
<!ELEMENT ObjectInfo (Description,
                      Label)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT Label (#PCDATA)>

<!--=====-->
<!--          Resource information          -->
<!--=====-->
<!ELEMENT Resource (ObjectInfo,
                   SimulationClass,
                   PhysicalProperties,
                   AdministrativeProperties)>

<!ELEMENT SimulationClass (#PCDATA)>
<!ELEMENT PhysicalProperties (Length, Width, Height, Weight)>
<!ELEMENT Length (#PCDATA)>
<!ATTLIST Length data-type (Integer | Float) #REQUIRED>
<!ELEMENT Width (#PCDATA)>
<!ATTLIST Width data-type (Integer | Float) #REQUIRED>
<!ELEMENT Height (#PCDATA)>
<!ATTLIST Height data-type (Integer | Float) #REQUIRED>
<!ELEMENT Weight (#PCDATA)>
<!ATTLIST Weight data-type (Integer) #REQUIRED>

<!ELEMENT AdministrativeProperties (Manufacturer, ModelNumber,
                                   SerialNumber, UpdateAddress,
                                   InformationAddress)>
```

```
<!ELEMENT Manufacturer (#PCDATA)>  
<!ELEMENT ModelNumber (#PCDATA)>  
<!ELEMENT SerialNumber (#PCDATA)>  
<!ELEMENT UpdateAddress (#PCDATA)>  
<!ELEMENT InformationAddress (#PCDATA)>
```

Appendix H

Resource XML

```
<!--=====-->
<!--          Resource information          -->
<!--=====-->
<Resource>
  <ObjectInfo>
    <Description>Vial for samples.</Description>
    <Label>Sample Vial</Label>
  </ObjectInfo>
  <SimulationClass>com.retisoft.java.sample_vial.Model</SimulationClass>
  <PhysicalProperties>
    <Length data-type="Float">30.0</Length>
    <Width data-type="Float">30.0</Width>
    <Height data-type="Float">75.5</Height>
    <Weight data-type="Integer">0</Weight>
  </PhysicalProperties>
  <AdministrativeProperties>
    <Manufacturer>Labman Automation Limited</Manufacturer>
    <ModelNumber>Sample Vial</ModelNumber>
    <SerialNumber>Sample-Vial</SerialNumber>
    <UpdateAddress>www.whatman.plc.uk</UpdateAddress>
    <InformationAddress>www.retisoft.ca</InformationAddress>
  </AdministrativeProperties>
</Resource>
```