INDS Execution Manager Reference Manual

Prepared by:
William H. Finger
Jesse A. Norris
Creare Incorporated

Creare Incorporated
PO Box 71
Hanover, NH 03755

TM-2910
Project No. 6249
April 2009

Introduction

This document is a user's manual for the Execution Manager, a software package for managing the deployment, configuration, startup, shutdown, and administration of INDS tasks. The intended audiences are the system's users and developers.

Operation

The Execution Manager is invoked from the command line as follows:

```
java –classpath inds_exec.jar startup.xml
```

where startup.xml is the file containing the commands to execute and manage. Once invoked, the Execution Manager performs this procedure:

1. initialization: the remote interface (based on Java RMI) is started.
2. parsing: the startup document is parsed to ensure all commands are valid.
3. execution: all commands are executed, in the order specified in the document. Asynchronous tasks will continue to run in the background.
4. waiting: the Execution Manager waits for every asynchronous task to complete.

During all these stages, the Execution Manager listens to the remote interface for incoming commands. Also, if Control-C is entered into the console, it cleanly shuts down each running command.

INDS Startup Commands

Below is a list of commands available to the INDS startup system. These commands should be placed in a text (.xml) file and invoked from the Execution Manager command line. Note that some commands also depend on settings in the "commands.xml" configuration file; see section "Register Your Command" for details.

Attributes for All Commands

a. initialDirectory

b. logFile

c. tag

A command may have any of these attributes. Attribute "initialDirectory" specifies an alternative directory to start the command's process. The default is the directory of the startup.xml file. The second attribute, "logFile", specifies a text file containing the standard output for the process. The last attribute is intended to allow the user to add descriptive text about a command.

Child Tags for All Commands

Any command can have zero or more of the following tags.

1. input
2. output

Each of these describes the inputs and outputs of a command, and can have the following attributes:

a. type
b. port
c. name
d. channel
e. cacheFrames
f. archiveFrames
g. overwriteArchive

The "type" attribute is one of "tcp", "udp", or "rbnb". This indicates the type of connection to use. In general each command only supports one connection type for inputs and one for outputs (see below).

The "port" attribute contains the port number used for the connection. The remaining attributes are specific to RBNB connections. The "name" is the name to give to the connection when it appears in the RBNB server. The "channel" is a path used to make data requests or to post the data to the server. The remaining attributes specify the size of the cache, archive, and whether to write over the archive, respectively.

Java Commands

All Java commands support the optional attribute "jvmMaxHeap", which specifies the amount of RAM available to the process heap.

1. dataTurbine

a.	name	server identifier string
b.	address	TCP host:port to bind to
c.	parent	server to attach to, if any
d.	loadArchivesAtStart	create sources for archives if "true"

2. xmlDemux

a.	silentMode	do not output logging information
b.	chanNameFromID	use the ID to generate output channel name
c.	xmlFile	required configuration filename

3. csvDemux

a.	dateFormat	format string for date output
b.	checkEmbeddedTimestamp	read the timestamp inside the file
c.	silentMode	(see xmlDemux)
d.	chanNameFromID	(see xmlDemux)
e.	xmlFile	(see xmlDemux)

4. udpCapture	
5. httpMonitor	
6. configFile	required configuration filetimeDrive
a. multiUserMode	Specifies sharing/security mode

Plugins

These are Java commands which form PlugIn connections to the DataTurbine.

1. PNG	
a. width	of the output image, in pixels
b. height	of the output image, in pixels
2. ToString	
3. ThumbNail	
a. scale	scale factor
b. quality	for JPEG images, between 0 and 1
c. maxImages	bound on images for large requests
4. TrackData	
a. noGUI	do not provide a configuration GUI
b. configFile	required configuration filename
c. compression	“decimation” or “simplification”
5. TrackKML	
a. consolidateResponse	track data is included in the response
b. noGUI	do not provide a configuration GUI
c. configFile	required configuration filename
6. DeadReckoning	
a. maxLatency_sec	maximum latency, in seconds

Other Commands

1. sleep	
a. duration_ms	time to pause, in milliseconds
2. del	
a. filename	name of file to remove, wildcards ok
3. tomcat	

Adding New Commands

Commands that are not listed above can be added to the Execution Manager by following this procedure:

1. Create a Java class that extends com.rbnb.inds.exec.Command.
2. Edit the file “commands.xml” to register your command with the Execution Manager.
3. Edit the configuration schema file “inds_startup.xsd”.
4. Edit your deployment command file to add your command.

These steps are described in more detail below.

Create Your Command

The first step in adding a new command is to extend the Command class, found in package com.rbnb.inds.exec. This package is built into the Execution Manager library, inds_exec.jar.

The Command constructor takes one argument, an instance of org.xml.sax.Attributes. These are the XML attributes specified in the deployment configuration file. In the construction of your command subclass, you should pass these attributes onto the Command constructor, then pull out your own attributes.

There are several methods that you can override to tweak the behavior of your command. Only two are required. The doExecute() method does the actual work of the command. It should return true if synchronous, false if asynchronous.

The other required method is getPrettyName(). This method returns a string used by viewing tools to show the command, and should reflect some critical element of your command’s configuration. For example, a delete command might include the name of the file it will delete.

The other methods are optional from the compiler’s perspective, but may need to be implemented for a working system. The doKill() method stops the execution of a running command at system shutdown, and should be implemented for asynchronous commands. The doWaitFor() method blocks until a command is complete, and should be implemented for asynchronous commands. The getChildConfiguration() method returns a string that contains any configuration data which is not directly in the deployment configuration file. For example, the URLs for HttpMonitor are contained in a separate XML file; getChildConfiguration on that command returns the contents of that file.

The getStdOut and getStdErr methods return input streams that will read the contents of your command’s output to System.out and System.err, respectively.

There are subclasses of Command to handle general types of commands. These are located in the package com.rbnb.inds.exec.commands. ExternalCommand is specialized for the task of starting external processes. You set the path to start your process by using setExecutablePath(). You can add command line arguments to your process with addArgument() or addArguments(). You can add environment variables to the process with addEnvironment(). The other command methods are defined adequately, save for getPrettyName().

JavaCommand is a specialization of ExternalCommand for Java programs. It automatically finds the location of the Java executable, and parses attributes specific to the Java Virtual Machine.

You need only add arguments for your classpath and class file, and any arguments for the command itself.

DtCommand is a further specialization of JavaCommand for Java programs which depend on the DataTurbine library. It automatically includes the DataTurbine in the classpath, along with your command class, which you specify in the constructor.

For more details on these classes, refer to the Execution Manager javadocs.

Register Your Command

Next, you need to edit the file commands.xml, which should be located with your deployment files. An entry in the file has the format shown below:

```
      <command>
            <name>del</name>
            <class>com.rbnb.inds.exec.commands.Del</class>
            <classification>transient</classification>
      </command>
```

The name field is a tag used to identify your command in the configuration schema and file. The class field is the fully qualified name of your class. It can be in any package. The classification field describes the type of command. We currently have defined the types “transient”, “source”, “server, “plugin,” and “converter.”

External commands also have defined the property “executableDirectory”. This identifies the pathname relative to the current working directory used to find the executable file.

In addition to these properties, you can add other arbitrary values specific to your command. They will be made available by the getCommandProperties() method of Command.

You can also add properties to a section marked <global> at the top of the document. These properties are available to every command. The property “dataTurbineDirectory” is currently the only global property used. It specifies where to find the rbnb.jar file for processes which require it.

Add the Command to the Configuration Schema

The deployment configuration file is validated against a schema document, “inds_startup.xsd”. In order to deploy your command, its configuration specification must be added to this schema. Writing schema documents is outside the scope of this document; chose a command similar to yours to start from. An example for the del command is shown below.

```
<xsd:element name="del" substitutionGroup="command">
      <xsd:complexType>
            <xsd:complexContent>
                  <xsd:extension base="CommandType">
                        <xsd:attribute name="file" type="xsd:string"/>
                  </xsd:extension>
            </xsd:complexContent>
      </xsd:complexType>
</xsd:element>
```

Add the Command to the Deployment File

Lastly, add the XML string needed to invoke your command to your deployment file. The syntax will match what you have prescribed in the configuration schema. Continuing the delete example:

```
      <del initialDirectory="Server" file="rbnb.log"/>
```

Remote Interface

The Execution Manager uses Java's Remote Method Invocation technology to allow client applications, either on the local machine or over a network, to interact with the Execution Manager. The remote interface allows clients to inspect the configuration and current state of managed commands. The following commands are available:

• getCommandList	get the ID for each command.
• getCommandOut	get the output stream for a command.
• getCommandError	get the error stream for a command.
• getCommandClassification	get the classification for each command, currently one of "transient", "source", "server", "plugin", or "converter".
• getConfiguration	get the XML snippet which the Execution Manager used to invoke this Command.
• isComplete	returns true if the command has completed, false otherwise.
• getChildConfiguration	yields the contents of the separate file used to configure the command, if any.
• getRootConfiguration	this gives the contents of the entire configuration document.
• getName	this gives a human readable descriptive name to identify the command.

Note that the commands "getCommandList" and "getRootConfiguration" take no arguments; the rest require the process ID of the command, a String.

Execution Manager Servlet Documentation Notes

The Execution Manager servlet folder includes two components. A servlet that provides an HTTP interface for accessing the functionality of the com.rbnb.inds.exec.Remote Java interface, and a web page that allows navigation through an Execution Manager using the com.rbnb.inds.exec.Remote Java interface. This document summarizes these two components.

Compiling and Dependencies

Compiling. Compile the components using Ant, and the build.xml file provided in INDSExManServlet. The output will be indsViewer.war and indsExec.war. These two files may be deployed by copying into the webapps directory of the current tomcat installation.

Dependencies. The viewer requires that the python xRender for INDS is installed in the current tomcat directory, and makes use of the remote interface for the INDS execution manager, inds_rmistubs.jar. Refer to:

RBNB\V3.2B1\apache-tomcat-5.5.12\webapps\ROOT\WEB-INF\cgi\
INDSExecutionManager\

Execution Manager Servlet

INDSExManServlet\src\ExecutionManagerServlet.java

The Execution Manager Servlet is designed to be a lower-level utility to query information from an Execution Manager using http requests with query strings appended to the base URL. The Execution Manager Servlet provides an http interface for accessing the functionality of the com.rbnb.inds.exec.Remote Java interface.

Class Variables

com.rbnb.inds.exec.Remote	remoteIndsObject
Class	remoteClass

Constructor Summary

ExecutionManagerServlet()
Initializes the class variables

Method Summary

void	doGet(HttpServletRequest request, HttpServletResponse response) Directs request to doPost()
void	doPost(HttpServletRequest request, HttpServletResponse response) Parses url query string, uses reflection to invoke the desired action on the remoteIndsObject, writes response.[1]

[1] The url query string has two variables that are parsed: command, which is the name of the process of interest, and action, which is the desired method that should be invoked on the remoteIndsObject. Refer to the com.rbnb.inds.exec.Remote Java interface for a list of acceptable actions.

Execution Manager Viewer

The execution manager viewer is a webpage that may be used to view an Execution Manager, navigate through the commands and invoke actions to query information from the Execution Manager. The viewer includes jsp pages, a Java bean that handles the connection and queries to the Execution Manager, and makes use of a SVG image generated by a Python script.

JSP pages

INDSExManServlet\viewerJsp

index.jsp[2]	This is the main page for working with the viewer. It uses three html frames to separate the content. Frames were chosen over other methods of layout because browsers support frame resizing easily and the content being passed between the frames is handled by a Java Bean with session scope.
action.jsp	This page is located in the right column of index.jsp. It is used to invoke different actions on and return the response from the Execution Manager.
commandlist.jsp	This page is located in the center column of index.jsp. It queries the Execution Manager and displays the commands. The commands are hyperlinked and force the action.jsp page to update as different commands are selected.
default.css	This is the cascading style sheet definitions for the jsp pages.

Execution Manager Bean

INDSExManServlet\src\indsBean\ExecutionManagerBean.java

The Execution Manager Bean contains the majority of Java scripting necessary for the JSP pages, thus reducing the scripting that occurs in the JSP pages. In addition, the bean is used to retain state for a given HTTP session. This minimizes the query strings that need to be passed to and from the JSP pages.

Class Variables

com.rbnb.inds.exec.Remote	remoteIndsObject
Class	remoteClass
Method[]	actions
String	queryCommand
String	queryAction
String	queryDisplay

Constructor Summary

ExecutionManagerBean()

Initializes the class variables by calling the method ExecutionManagerConnect().

[2] The left frame of the index.jsp page invokes a Python script, xRender.py, that generates and returns a SVG image. Refer to documentation for xRender.py for additional details.

Method Summary

void	ExecutionManagerConnect() Initializes the class variables.
String	getActionResponse() Get HTML formatted response for the current queryCommand and queryAction
String	getActionList() Get HTML formatted list of available actions that may be invoked
String	getCommandList() Get HTML formatted list of commands
void	setQueryAction(String queryAction) Set the class variable queryAction which is the current action that should be invoked on the remoteIndsObject
String	getQueryAction() Return the class variable queryAction
void	setQueryCommand(String queryCommand) Set the class variable queryCommand which is the current action that should be invoked on the remoteIndsObject
String	getQueryCommand() Return the class variable queryCommand
String	getCommandName() Return the pretty name as opposed to command id
void	setQueryDisplay(String queryDisplay) Set the class variable queryDisplay which is used to switch between showing all commands and only the current commands
String	getQueryDisplay() Return the class variable queryDisplay

Execution Manager Exception

INDSExManServlet\src\indsBean\ExecutionManagerException.java

The Execution Manager Exception class allows for a message to be appended when an exception is caught in the Execution Manager Bean. This was useful for displaying when a connection could not be established.

Future Work

Although we have produced a powerful and useful tool, there are some areas where improvements could be made to the Execution Manager.

Commands

Many more commands could be added to list of commands which the Execution Manager supports. Those currently supported were driven by a desire to provide the equivalent functionality as previous INDS scripting; future problem domains may provide inspiration for added tasks. Also, some commands may require some tweaking. The DataTurbine task may require better shutdown handling, for example. For the Tomcat web server task we created a command instance just to oversee shutdown; a similar model could be used for the DataTurbine.

Remote Interface

Some features are lacking from the remote interface, in particular the ability to terminate tasks or to add new tasks (see Deployment below).

Deployment

In order for the configuration and administration of large INDS networks to be feasible, a means is required to distribute configuration information over the network. The Execution Manager could be expanded such that it can accept and verify configuration files via its remote interface. This would allow node machines to be configured dynamically from a central location in a platform independent manner.

Startup

Although the Execution Manager allows for remote administration, currently it needs to be started manually from a local machine. We could integrate the Execution Manager with Tomcat and thus enable remote start via HTTP request.

Servlet and Viewer

Here we have incorporated some of Lawrence Freudinger’s ideas in response to 25 February 2009 telecon.

- Improve exception handling to provide more useful information about why exceptions might have occurred.
- Allow for multiple commands to be selected, e.g. using check boxes. This would generate a response that includes a summary of invoking an action on different commands.
- Allow for multiple actions to be selected, e.g. using check boxes. This would allow a quick way to view the responses for different actions at once.
- Add ability to output XML formatted data. An XML output style plus the checkboxes establishes the ability to generate configurable configuration documents as a web service. It seems to me that this is only one or two steps away from

generating a formatted document suitable for printing (maybe another processing step that applies a style sheet and coughs out a PDF?).

- Replace a frame based layout with a DIV based approach. This could include headers and footers, possibly expandable into navigation aids, help file URLs, etc. These areas could be customized to look like a NASA site or a customer-specific or a project-specific site.
- Use JavaScript to improve updating of the content.

Lastly, we need to expand the scope of the viewer. Currently, it provides a detailed description of the layout and status of a single Execution Manager node. However, it provides no means for traversing to child, parent, or sibling nodes, or even a means to know about their existence. In addition, the Execution Manager exists as part of a larger system. To the extent that this larger system can be described and represented as a graph, the viewer could provide a means for displaying and traversing the nodes of the graph for the purposes of administration and understanding. At the current time, these other system components have not been defined; until then, only ad hoc methods of connecting them can be available.