Scripting

for the

ROES Server

1. Scripting Basics

What are scripts?

Scripts are pieces of code that can be added to the server which can interact with the server, its environment and the outside world. Through scripts you can extend the behavior of the server in any way that you want.

How is that possible?

The server is a Java application. Java has a mechanism where various scripting engines conforming to a specification called "JSR 223" can can be used by an application to run scripts. These scripts can interact within their own environment as well as the environment of the application, to the extent that the application provides access to it's environment.

Scripting Languages

There are a raft of scripting engines available that support different languages:

AWK, BeanShell, ejs, FreeMarker, Groovy, Jaskell, Javam, JavaScript, Jelly, JEP, Jexl, jst, JudoScript, JUEL, OGNL, Pnuts, Python, Ruby, Scheme, Sleep, Tcl, Velocity, XPath, XSLT, JavaFX Script, ABCL, AppleScript, Bex scrip, OCaml Scripting Project, PHP, Python, Smalltalk, CajuScript, MathEclipse

There are two engines that will always be available in the Server, Java Script (ECMAScript, Built in by default in the JRE) and BeanShell, a java engine. On Mac OS X AppleScript will also be available.

2. Integrating scripts with the server

Scripts can be used to extend and modify the behavior of the server. Scripts plug into the server architecture in one of three different ways: In a report, in a Script Agent or in a Custom Event Listener.

2.1. Scripting in Reports

Scripts can be incorporated into reports using the BEGIN_SCRIPT and END_SCRIPT macros.

The BEGIN_SCRIPT macro take a parameter that identifies the language the enclosed script will be written in.

[BEGIN_SCRIPT([“language." | “engine."]Name)]

if no prefix is provided then it is assumed to be "language". The Name is either the name of the language or the name of the engine that is to be used. For example

[BEGIN_SCRIPT(BeanShell)]

This indicates the enclosed script is written in the BeanShell language; as does:

[BEGIN_SCRIPT(language.BeanShell)]

The following says to use the Beanshell Engine

[BEGIN_SCRIPT(engine.BeanShell Engine)]

This says to use the Mozilla Rhino engine which runs ECMAScript (JavaScript):

[BEGIN_SCRIPT(engine.Mozilla Rhino)]

First simple script

Our scripts are going to be focused on BeanShell which is an Engine capable of executing java and supports other additional scripting features. You can find out more about BeanShell at: http://www.beanshell.org You can find out more about the BeanShell language at: http://www.beanshell.org/manual/bshmanual.html

Hello World Script

```
[//]———Begin Report————
[BEGIN_SCRIPT(BeanShell)]
      gOutputWriter.write( "Hello World\n");
[END_SCRIPT]
[//]———End Report————————
```

If you run this report it will produce a file that looks like:

Hello World

OK, what is that gOutputWriter object and where did it come from?

The script environment

When a script is run, before any of the script code is executed, there are three variables that are created and exist globally: gOutputWriter, gReportMacros and gSystemData. For the moment let's just focus on gOutputWriter.

Generating report content

gOutputWriter is a java.io.StringWriter which enables the script to write output back into the report being generated. At the point where we called the write method, that is where we actually wrote "Hello World\n" into report. So effectively the evaluation of a script block is the accumulation of all the text that the script writes into the gOutputWriter in that block.

Accessing report Macros

gReportMacros is a Hashtable that contains all the macros that exist at the context of where the script block has been placed in your report. This allows your script access to any macro that is available at the time of its execution. For example, let's get the macro LAB_ORDER_ID and use it in a report:

```
[//]———Begin Report————
Lab Order id: [\]
[BEGIN_SCRIPT(BeanShell)]
      String labOrderID = (String)gReportMacros.get("LAB_ORDER_ID");
      gOutputWriter.write( labOrderID );
[END_SCRIPT]
[//]———End Report————————
```

The output of the report if we were dealing with an order with an id of "000001" would be:

```
Lab Order id: 000001
```

You can also set or create new script macro by setting the macros' key to a value in the gReportMacros Hashtable. For example:

```
[//]———Begin Report————
[BEGIN_SCRIPT(BeanShell)]
      gReportMacros.put("MY_OWN_MACRO", "my own macro");
[END_SCRIPT]
Hello from [MY_OWN_MACRO]
[//]———End Report————————
```

The output of this report would be:

```
Hello from my own macro
```

Notice that the script does not use the gOutputWriter so the result of executing the script is the side effect of a new macro in the gReportMacros object which is accessed using the normal report macro evaluation mechanism.

Macro added to the gReportMacros will exist for the duration of time that the enclosing report is being evaluated.

Accessing gSystemData

gSystemData is the primary means by which information about the environment is provided to your script. It is a Hashtable that is pre-populated with a number of fields that contain useful data. Broadly, the data in gSystemData falls into four categories: Workstation, Directories, Report Context and Other. Here is a list of the fields and the data each field contains:

Data about the workstation

Field	Type	Description
WORKSTATION_NAME	String	The name of this workstation.
DATABASE_URL	String	The database URL for this workstations database connection.
DATABASE_UN	String	The database username for this workstations database connection.
DATABASE_PW	String	The database password for this workstations database connection.
ENSEMBLE_ADDRESS	String	The address that this workstation is using to communicate with other ensemble participants.
ENSEMBLE_PORT	String	The port that this workstation is listening on for connections from other ensemble participants.

Data about directories

Field	Type	Description
DIR_ROOT_PATH	String	The value of the ROOT_PATH setting for this workstation.
DIR_INCOMING	String	The "Incoming orders" directory path.
DIR_DONE	String	The "Finished orders" directory path
DIR_WORK	String	The "Working files" directory path.
DIR_LAYOUT_BACKGROUNDS	String	The "Layout Backgrounds" directory path.
DIR_ATTACHMENTS	String	The "Attachments" directory path
DIR_GENERATED_IMAGES	String	The "Client Generated Images" directory path.
DIR_IMAGES	String	The "Order Images" directory path.
DIR_REPORTS	String	The "Client Generated Images" directory path.

Data about the report context

Field	Type	Description
ReportMacros	Hashtable	A Hashtable containing all the current macro values given the context of the script block. This is the same object referenced by gReportMacros.
OutputWriter	java.io.StringWriter	A StringWriter that can be used to generate the data that this report block will evaluate to. This is the same object referenced by gOutputWriter.

Other handy data

Field	Type	Description
PersistentSystemData	Hashtable	A Hashtable reference to the global persistent data of this workstation.
SystemDataKeys	ArrayList<String>	An array list containing the field names in the gSystemData object.

Accessing the data in the gSystemData Hashtable is just a matter of calling get with the appropriate key. For example, let's write a script that gets the name of the workstation that we are running on and tells us how many orders are waiting to be processed in the incoming directory.

```
[//]———Begin Report—————
[BEGIN_SCRIPT(BeanShell)]
      String workstationName =
                        (String)gSystemData.get("WORKSTATION_NAME");
      File incomingDir = new File( gSystemData.get("DIR_INCOMING"));
      gOutputWriter.write( "The workstation " + workstationName);
      gOutputWriter.write( " sees " + incomingDir.list().length);
      gOutputWriter.write( " files still waiting to be processed\n");
[END_SCRIPT]
[//]———End Report—————
```

The output of this script would be something like:

The workstation MyStation sees 5 files still waiting to be processed

Creating data that persist beyond the scope of this script

It is possible to create objects in a script that live beyond the scope of the script and its enclosing report. You can add or set data in PersistentSystemData and that data will survive beyond the scope of the script that created/set it. It will persist until the workstation shuts down.

For example:

```
[//]———Begin Report—————
[BEGIN_SCRIPT(BeanShell)]
      String persistentString = "I want to live!";
      Hashtable psd = gSystemData.get("PersistentSystemData");

      // Add persistentString to the psd Hashtable
      psd.put( "persistentString", persistentString);
[END_SCRIPT]
[//]———End Report—————
```

In a later report the data that we placed in the PersistentSystemData can be retrieved:

```
[//]———Begin Report—————
[BEGIN_SCRIPT(BeanShell)]
      Hashtable psd = gSystemData.get("PersistentSystemData");
      String persistentString = psd.get( "persistentString");
      gOutputWriter.write( persistentString );

      // Now, let's remove persistentString from the psd Hashtable
      psd.remove( "persistentString");
[END_SCRIPT]
[//]———End Report—————
```

If these two scripts were run one after the other then the output of the second script would be:

I want to live!

So one other interesting nugget: At the beginning of a script invocation, all the items in PersistentSystemData will be replicated into gSystemData. This means that once you've added an item to PersistentSystemData, it can be retrieved in subsequent script invocations by getting it from gSystemData. So the last script could have been written like so:

```
[//]———Begin Report—————
[BEGIN_SCRIPT(BeanShell)]
      String persistentString = gSystemData.get( "persistentString");
      gOutputWriter.write( persistentString );

      // Now, let's remove persistentString from PersistentSystemData
      Hashtable psd = gSystemData.get("PersistentSystemData");
      psd.remove( "persistentString");
[END_SCRIPT]
[//]———End Report—————
```

Note that in order to remove the object that we initially put in PersistentSystemData, we still had to get the PersistentSystemData object.

2.2. Script Agents

Script Agents allow you to insert the execution of scripts into a workflow. Like any Agent, when

they process jobs from their incoming queue, they are handed a batch of jobs to process. The script in the Script Agent can get access to the batch of jobs being processed as well as the XML data of the order that the jobs originated from. The data can be obtained through these additional fields in the gReportMacros Hashtable:

Additional Agent gReportMacros Items

Field	Type	Description
JobBatch	ArrayList<RSServerJob>	This is the list of jobs that are in the batch being processed by the agent.
OrderXMLData	RSXMLElement	The root object of the order's xml data.
AgentName	String	The name of the agent we are running in.

When you create scripts in a Script Agent you do not embed them within a report, but rather you enter them directly into the script text area of the script agent. All the same environment variables will exist when the script is executed: gOutputWriter, gReportMacros and gSystemsData. If you write to the gOutputWriter in a script that runs in a Script Agent, any data written will be sent to the log for that agent. The gReportMacros will have all the macro values defined as if you were at the root level of a report, in addition to the JobBatch and OrdersXMLData.

Let's say we wanted to write to the log how many jobs were in the batch that we are processing we could do it something like:

```
// Begin script
ArrayList<RSServerJob> theBatch = gReportMacros.get("JobBatch");
gOutputWriter.write( "Number of jobs in batch: " + theBatch.size() );
// End Script
```

The output of this script would appear in the Agents log something like:

Number of jobs in batch: 17

2.3. Custom Event Listeners

When the server is running, events will be generated when interesting things happen. Those events will be broadcast to anyone who is listening for them. An event typically consists of an event identifier, a String that is the name of the event, and event data which will be some object that carries data related to the event.

Events

Event Name	Broadcast to	Sent When	EventData
EVENT_DATABASE_CHANGED	Local Listeners	The underlying database connection for the workstation has changed	none
OUR_IP_ADDRESS_CHANGED	Local Listeners	When a particular NIC is selected for this workstation.	<workstation IP Address>
WORKSTATION_ADDED	Local Listeners	An ensemble listener has been added to this workstation	<workstation IP Address>:<port>
WORKSTATION_REMOVED	Local Listeners	An ensemble listener has been removed from this workstation	<workstation IP Address>:<port>
ON_LOCAL_LAUNCH	Local Listeners	The workstation starts up without regard to whether anyone is logged in	none
ON_LOCAL_SHUTDOWN	Local Listeners	The workstation is shutting down	none
ON_WORKSTATION_SHUTDOWN	Remote Listeners	The workstation has shutdown all processing and is just about to exit	<workstation IP Address>:<port>
ON_USER_LOGGED_IN	All Listeners	A user successfully logs into a workstation	<Username>[,]<Workstation Name>
ON_USER_LOGGED_OUT	All Listeners	A user logs out of a workstation	<Username>[,]<Workstation Name>

Event Name	Broadcast to	Sent When	EventData

ON_LOCAL_BATCH_COMPLETE	Local Listeners	A batch is successfully processed by a printer or agent before it is reenqueued elsewhere	A Hashtable Object with:
			{
			"AgentName", <Name of the Agent>
			"BatchID", <The batch ID>
			"OrderID", <The order's ID>
			"TheBatch", ArrayList<RSServerJob> Object
			}

ON_LOCAL_BATCH_ERROR	Local Listeners	A batch has encountered an error while being processed by a printer or agent.	A Hashtable Object with:
			{
			"AgentName", <Name of the Agent>
			"BatchID", <The batch ID>
			"OrderID", <The order's ID>
			"TheBatch", ArrayList<RSServerJob> Object,
			"Exception", <The Exception object thrown>
			}

ON_ORDER_PROCESSED	All Listeners	An order is successfully processed	<The assigned Lab ID of the Order>

ON_LOCAL_ORDER_PROCESSED	Local Listeners	An order is successfully processed on this workstation	A Hashtable Object with:
			{
			"OrderID", <The order's ID>
			"CustomerOrderID", <The client generated order id>
			"OrderFileName", <The name of the order file>
			}

ON_ORDER_PROCESS_ERROR	Local Listeners	An order has erred during initial processing	A Hashtable Object with:
			{
			"OrderID", <The order's ID>
			"CustomerOrderID", <The client generated order id>
			"OrderFileName", <The name of the order file>
			"Exception", <The Exception object thrown>
			}

ON_SEND_LOG	All Listeners	We are directed to send our logs to the error notification recipient	A String with "true" or "false" indicating the logs should also be sent to SoftWorks.

Event Name	Broadcast to	Sent When	EventData
ON_ORDER_DELETED	All Listeners	An order is successfully deleted	<The assigned Lab ID of the Order>

ON_LOCAL_ORDER_DELETED	Local Listeners	An order is successfully deleted on this workstation	A Hashtable Object with:
			{
			"OrderID", <The order's ID>
			}

ON_FLUSH_CACHED_ORDER_DATA	All Listeners	The underlying order data has been modified	<The assigned Lab ID of the Order>

ON_STORE_CUSTOMER_DATA_ERROR	Local Listeners	When an error occurs storing customer data	A Hashtable Object with:
			{
			"OrderID", <The order's ID>
			"CustomerOrderID", <The client generated order id>
			"Exception", <The Exception object thrown>
			}

ON_EPAY_ERROR	Local Listeners	An error occurs while trying to process the credit card for an order on this workstation	A Hashtable Object with:
			{
			"TheOrder", <The RSOrder object for the order>
			"ErrorMessage", <The error message returned>
			}

Custom Event Listeners allow you to listen for events and when those events are heard, to execute a script.

When an event listener invokes a script, unlike with a Script Agent or Report, there is no order context so the gReportMacros will have very few items setup in it; only those that do not relate to order data, like the "CURRENT_TIME", etc. Among those that are defined there will be a couple of additional items:

Additional Listener gReportMacros Items

Field	Type	Description
EventName	String	This is the name of the event that triggered this listener.
EventData	Object	This is the data that was passed with the event.

3. Advance Scripting Concepts

Interacting with the Server environment

Classes the server provides for you to interact with the server environment

The server provides a number of classes that your script can use and all of them will be in the java package named com.softworks.server.scripting. To use these classes in your script you will need to add an import statement like this:

```
import com.softworks.server.scripting.*;
```

RSScriptUtilities - A class that contains a collection of utility methods

```
public class RSScriptUtilities
{
      public static File locateImageFile( RSXMLElement pImageElement,
                                          Hashtable pSystemData )
      public static File locateImageFile( RSXMLElement pImageElement,
                                          Hashtable pSystemData,
                                          String pOrderID )
      public static RSJobQueue getJobQueueByName( String pName)
      public static RSOrder getOrderByID( String pOrderID)
      public static RSAgent getAgentByName( String pName)
      public static ArrayList<RSAgent> getAgents()
      public static RSListenerAgent getListenerAgentByName( String pName)
      public static ArrayList<RSListenerAgent> getListenerAgents()
      public static void addOrderContextualMenuItem(  String pMenuTitle,
                                                      String pEventName,
                                                      int pEventScope )
      public static void removeOrderContextualMenuItem( String pMenuTitle )
      public static void notifyListeners( String pEventName,
                                          Object pEventData )
      public static void notifyLocalListeners(  String pEventName,
                                                Object pEventData )
      public static void notifyRemoteListeners( String pEventName,
                                                Object pEventData )
      public static void addServerEventListener(
                                          RSServerEventListener pListener,
                                          String pEventName )
      public static void removeServerEventListener(
                                          RSServerEventListener pListener,
                                          String pEventName )
      public static void removeServerEventListener(
                                          RSServerEventListener pListener )
      public static String getServerEnsembleAddressAndPort()
      public static boolean secureControlOf( String pTargetDesc )
      public static boolean secureControlOf( String pTargetDesc,
```

```
                                        long pMaxWaitTime )
public static boolean releaseControlOf( String pTargetDesc )
public static Rectangle getScreenBounds( Window pWindow )
public static Insets getScreenInsets( Window pWindow )
public static void doExec(String[] pCmds)
public static void doExec(String pCmd)
public static RSCatalog getCatalogForOrder( RSOrder pOrder )
public static Dimension getImageDimensions(
                                    InputStream pInputStreamForImage )
public static RSXMLElement readXMLFrom(
                         InputStream pInputStream) throws Exception
public static String getSharedData( String pName) throws SQLException
public static void setSharedData( String pName,
                                  String pValue) throws SQLException
public static String evaluateEmbeddedReport(
                               String pReportTemplate,
                               Hashtable pSystemData ) throws Exception
public static RSXMLElement getXMLFromMacroObject( String pMacroName,
                                                  Hashtable pMacros )
public static void populateTableWithRecordsTableData(
                         String pTableName,
                         RSXMLElement pRecordsTableData,
                         HashMap<String, String> pAdditionalAssignments)
                                                       throws Exception
public static void populateTableWithRecordsTableData(
                         String pTableName,
                         RSXMLElement pRecordsTableData )
                                                       throws Exception
public static void sendEmail( String pSubject,
                              String pFromName,
                              String pFrom,
                              String pTo,
                              String pCc,
                              String pBcc,
                              String pMsgBody ) throws Exception
public static void sendEmail( String pSubject,
                              String pFromName,
                              String pFrom,
                              String pTo,
                              String pCc,
                              String pBcc,
                              String pMsgBody,
                              boolean pMsgBodyIsHTML ) throws Exception
public static boolean workstationIsProcessingIncomingOrders()
public static void startProcessingIncomingOrders()
public static void stopProcessingIncomingOrders()
public static void shutdownTheWorkstation( String pReason )
public static boolean isValidUser( String pUserName, String pPassword)




// The following methods are accessible only from signed scripts
public static String morphCrop( Hashtable pSystemData,
                                File pImageFile,
                                String pOriginalCrop,
                                String pNewNodeBounds )
```

```
                                                    throws Exception
}
```

Description
`public static File locateImageFile( RSXMLElement pImageElement, Hashtable pSystemData )` This method locates an image file that belongs to an order. Parameters: **pImageElement** - The xml element representing the image in the order **pSystemData** - The gSystemData object that was passed to your script, your script must be in either a Report or Script Agent; i.e. the gSystemData must have order contextual data in it. If your script is in an event, you can call the variant below. Returns: A File object pointing to the image file
`public static File locateImageFile( RSXMLElement pImageElement, Hashtable pSystemData, String pOrderID )` This method locates an image file that belongs to an order. Parameters: **pImageElement** - The xml element representing the image in the order **pSystemData** - The gSystemData object that was passed to your script **pOrderID** - The lab ID of the order that the image belongs to Returns: A File object pointing to the image file
`public static RSJobQueue getJobQueueByName( String pName)` This method returns a server job queue object from the server. Parameters: **pName** - Name of the job queue you want returned Returns: An RSJobQueue object representing the queue in the server

Description
`public static RSOrder getOrderByID( String pOrderID)` This method returns an order object from the server. Parameters: **pOrderID** - The ID of the order you want returned Returns: An RSOrder object representing the order in the server
`public static RSAgent getAgentByName( String pName)` This method returns an agent object from the server. Parameters: **pName** - The name of the agent you want returned Returns: An RSAgent object representing the agent in the server
`public static ArrayList<RSAgent> getAgents()` This method returns an array list of agent object for all the agents in the server. Parameters: None. Returns: An ArrayList of RSAgent objects; one for each agent in the server
`public static RSListenerAgent getListenerAgentByName( String pName)` This method returns a listener agent object from the server. Parameters: **pName** - The name of the listener agent you want returned Returns: An RSListenerAgent object representing the listener agent in the server

Description
`public static ArrayList<RSListenerAgent> getListenerAgents()` This method returns an array list of listener agent object for all the listener agents in the server. Parameters: None. Returns: An ArrayList of RSListenerAgent objects; one for each listener agent in the server
`public static void removeOrderContextualMenuItem( String pMenuTitle )` This method removes a contextual menu item that had been added with the addOrderContextualMenuItem method. Parameters: **pMenuTitle** - The label of the contextual menu item to be removed Returns: Nothing

`public static void notifyListeners( String pEventName,`
`Object pEventData )`
 	
This method broadcasts an event to all listeners both local and remote in the system.
 	
Parameters:
 	
pEventName - The name of the event to be broadcast when item is selected
pEventData - An object that will be passed with the event. Currently event data that is to be broadcast to remote listeners is limited to a String
 	
Returns: Nothing

`public static void notifyLocalListeners( String pEventName,`
`Object pEventData )`
 	
This method broadcasts an event to local listeners on the workstation.
 	
Parameters:
 	
pEventName - The name of the event to be broadcast when item is selected
pEventData - An object that will be passed with the event.
Returns: Nothing

Description
`public static void notifyRemoteListeners( String pEventName, Object pEventData )` This method broadcasts an event to remote listeners in the system. Parameters: **pEventName** - The name of the event to be broadcast when item is selected **pEventData** - An object that will be passed with the event. Currently event data that is to be broadcast to remote listeners is limited to a String Returns: Nothing
`public static void addServerEventListener( RSServerEventListener pListener, String pEventName )` This method adds an event listener to the system. Parameters: **pListener** - The listener to be added. **pEventName** - The name of event that this listener is listening for. Returns: Nothing
`public static void removeServerEventListener( RSServerEventListener pListener, String pEventName )` This method removes an event listener from the system for a particular event. Parameters: **pListener** - The listener to be removed. **pEventName** - The name of event that this listener is to removed from listening for. Returns: Nothing
`public static void removeServerEventListener( RSServerEventListener pListener )` This method removes an event listener from the system for all events. Parameters: **pListener** - The listener to be removed. Returns: Nothing

Description
`public static String getServerEnsembleAddressAndPort()` This method returns the Ensemble address (IP address) and port of this workstation separated by a ":". Parameters: None. Returns: A String containing the ensemble address and port of this workstation
`public static boolean secureControlOf( String pTargetDesc )` This method provides a means of coordinating activity among all threads on all the workstations in the system. When you call this method it will not return until the calling thread has secured exclusive control of the target. The target is simply a name that competing threads agree to coordinate on. It is imperative that whenever you no longer need control of the target that you call the releaseControlOf method identifying the target. Parameters: **pTargetDesc** - The name target that threads will coordinate on. Returns: A boolean indicating whether we succeeded in gaining control of the target. A result of false means that an error occurred trying to secure control of the target.

`public static boolean secureControlOf( String pTargetDesc,`
`long pMaxWaitTime)`
 	
This method provides a means of coordinating activity among all threads on all the workstations in the system. When you call this method it will not return until the calling thread has secured exclusive control of the target or the pMaxWaitTime has expired. The target is simply a name that competing threads agree to coordinate on. It is imperative that whenever you no longer need control of the target that you call the releaseControlOf method identifying the target.
 	
Parameters:
 	
pTargetDesc - The name target that threads will coordinate on.
pMaxWaitTime - The number of milliseconds that we will wait to gain control before timing out
 	
Returns:
 	
A boolean indicating whether we succeeded in gaining control of the target. A result of false means that an error occurred trying to secure control of the target or that pMaxWaitTime has passed.

Description
`public static boolean releaseControlOf( String pTargetDesc )` This method releases our exclusive control of the target and allows other threads that are waiting for access to the target to proceed. Parameters: **pTargetDesc** - The name target that threads will coordinate on. Returns: A boolean indicating whether we succeeded in releasing control of the target. A result of false means that an error occurred trying to release control of the target.
`public static Rectangle getScreenBounds( Window pWindow )` This method is a utility function that returns the bounds of a display of the workstation minus the insets for task bar/dock. If the pWindow parameter is null then it will get the size of the default display, otherwise it will get the size of the display that the pWindow is on. Parameters: **pWindow** - A window that is on the display that we want the size of. If this parameter is null, then we will get the size of the default display. Returns: A Rectangle that is the bounds of the display minus the task bar or on OS X, the Dock and menu bar.
`public static Insets getScreenInsets( Window pWindow )` This method is a utility function that returns the insets of a display of the workstation that include the task bar or on OS X, the dock and menu bar. If the pWindow parameter is null then it will get the insets of the default display, otherwise it will get the insets of the display that the pWindow is on. Parameters: **pWindow** - A window that is on the display that we want the insets of. If this parameter is null, then we will get the insets of the default display. Returns: An Insets object that is the insets of the display including the task bar or on OS X, the Dock and menu bar.

Description
`public static void doExec(String[] pCmds)` This method is a utility function that executes a command line function. Parameters: **pCmds** - An array of strings that constitute the parts of the command line function; i.e. the function and it's parameters. Returns: Nothing
`public static void doExec(String pCmd)` This method is a utility function that executes a command line function. Parameters: **pCmd** - A String that represents the function to be executed on the command line, the parameters to the function must be space separated. Returns: Nothing
`public static RSCatalog getCatalogForOrder( RSOrder pOrder )` This method locates and returns the catalog for the given order. Parameters: **pOrder** - The order object for which you would like the catalog Returns: An RSCatalog object representing the catalog from which the order was created or null if it could not be located.

`public static Dimension getImageDimensions(`
`InputStream pInputStreamForImage )`
 	
This method returns the dimensions of an image.
 	
Parameters:
 	
pInputStreamForImage - An image input stream to an image whose dimensions we will return
 	
Returns:
 	
The dimensions of the image referenced by pInputStreamForImage.

Description
`public static RSXMLElement readXMLFrom( InputStream pInputStream) throws Exception` This method reads xml data from an InputStream and returns the root element of that xml data. Parameters: **pInputStream** - An input stream to the xml data to be read in Returns: The root element of the xml data read. Throws: If an exception is caught while reading from the input stream it will be thrown out of this call
`public static String getSharedData( String pName) throws SQLException` This method returns a value from the SharedData database table. Parameters: **pName** - The name of the data item to be returned Returns: The value associated with the name passed in. Throws: If an exception is caught while accessing the database it will be thrown out of this call
`public static void setSharedData( String pName, String pValue) throws SQLException` This method returns a value from the SharedData database table. Parameters: **pName** - The name of the data item to be set **pValue** - The value to be associated with the name. Returns: Nothing Throws: If an exception is caught while accessing the database it will be thrown out of this call

Description

```
public static String evaluateEmbeddedReport(
                                String pReportTemplate,
                                Hashtable pSystemData ) throws Exception
```

This method returns the evaluated result of a report template string. The report is evaluated within the context of the script's report.

Parameters:

pReportTemplate - The report string to be evaluated
pSystemData - The system data object.

Returns:

The evaluated result.

Throws:

If an exception is caught while evaluating the report it will be thrown out of this call.

```
public static RSXMLElement getXMLFromMacroObject( String pMacroName,
                                                  Hashtable pMacros )
```

This method returns an RSXMLElement that is associated with an object referenced by a macro in the gMacros table. When is that useful? Sometimes, an object may be placed in the gMacros table as a place holder. For example, when you are iterating through a list in a report using something like [BEGIN_UNIT_LIST] and [END_UNIT_LIST], within the begin and end macros an object representing the current item in the iteration will be placed in the macros table. The name is typically starts with CURRENT_ so in the example where we are iterating through a unit list there will be an entry with a key of CURRENT_UNIT. If you would like to access the corresponding xml for these items you can call this method passing the name of the macro and the macro table itself and it will return an RSXMLElement or null if there is no corollary xml or the macro does not exist.

Parameters:

pMacroName - The macro to extract an RSXMLElement from.
pMacros - The gMacros data object in your script.

Returns:

The xml of the associated macro object as an RSXMLElement or null if there is no associated xml or if the macro named does not exist in the pMacros table.

Description

```
public static void populateTableWithRecordsTableData(
                        String pTableName,
                        RSXMLElement pRecordsTableData,
                        HashMap<String, String> pAdditionalAssignments )
                                                    throws Exception
```

This method will attempt to import the contents of a records table into the identified table in the ROES Servers database. It will also assign static values to each record field that is identified in the pAdditionalAssignments hash map.

Parameters:

pTableName - The name of the table to populate.
pRecordsTableData - The root of the XML of the records table data from the order.
pAdditionalAssignments - Additional fields to set and the values to set them to in each record.

Returns: Nothing

Throws:

If an exception is caught while populating the table it will be thrown out of this call.

```
public static void populateTableWithRecordsTableData(
                        String pTableName,
                        RSXMLElement pRecordsTableData )
                                                    throws Exception
```

This method will attempt to import the contents of a records table into the identified table in the ROES Servers database.

Parameters:

pTableName - The name of the table to populate.
pRecordsTableData - The root of the XML of the records table data from the order.

Returns: Nothing

Throws:

If an exception is caught while populating the table it will be thrown out of this call.

Description

```
public static void sendEmail( String pSubject,
                              String pFromName,
                              String pFrom,
                              String pTo,
                              String pCc,
                              String pBcc,
                              String pMsgBody ) throws Exception
```

This method will send an email using the SMTP server settings from the Server's Email tab.

Parameters:

pSubject - The subject line for the email.
pFromName - The name of the sender of the email.
pFrom - The email address of the sender of the email.
pTo - The email address(es) to send this email to. This can be a comma separated list of email addresses.
pCc - The email address(es) to send this email to as a carbon copy. This can be a comma separated list of email addresses or null if you do not wish to send carbon copies.
pBcc - The email address(es) to send this email to as a blind carbon copy. This can be a comma separated list of email addresses or null if you do not wish to send blind carbon copies.
pMsgBody - The plain text content of the email.

Returns: Nothing

Throws:

If an exception occurs while sending the email it will be thrown out of this call.

Description

```
public static void sendEmail( String pSubject,
                              String pFromName,
                              String pFrom,
                              String pTo,
                              String pCc,
                              String pBcc,
                              String pMsgBody,
                              boolean pMsgBodyIsHTML ) throws Exception
```

This method will send an email using the SMTP server settings from the Server's Email tab.

Parameters:

pSubject - The subject line for the email.
pFromName - The name of the sender of the email.
pFrom - The email address of the sender of the email.
pTo - The email address(es) to send this email to. This can be a comma separated list of email addresses.
pCc - The email address(es) to send this email to as a carbon copy. This can be a comma separated list of email addresses or null if you do not wish to send carbon copies.
pBcc - The email address(es) to send this email to as a blind carbon copy. This can be a comma separated list of email addresses or null if you do not wish to send blind carbon copies.
pMsgBody - The content of the email. This can be html or plain text. If it is html then the next parameter (pMsgBodyIsHTML) should be set to true.
pMsgBodyIsHTML - Identifies whether the message body should be sent as html or plain text.

Returns: Nothing

Throws:

If an exception occurs while sending the email it will be thrown out of this call.

```
public static boolean workstationIsProcessingIncomingOrders()
```

This method returns the current processing state of this workstation.

Parameters:

None.

Returns:

A Boolean set to true if this workstation is currently processing incoming orders; otherwise false.

Description
`public static void startProcessingIncomingOrders()` This method causes this workstation to start processing incoming orders. It's the equivalent of clicking on the "Start Server" button on the workstation. If the server is already processing incoming orders then this call does nothing. Parameters: None. Returns: Nothing
`public static void stopProcessingIncomingOrders()` This method causes this workstation to stop processing incoming orders. It's the equivalent of clicking on the "Stop Server" button on the workstation. If the server is not currently processing incoming orders then this call does nothing. Parameters: None. Returns: Nothing
`public static void shutdownTheWorkstation( String pReason )` This method causes this workstation to: stop processing incoming orders if it is currently doing so, to stop all the running Agents and Listeners on this workstation and then to completely shutdown and terminate the workstation. It is similar to the user quitting the workstation with the exception that a dialog giving the user the option to cancel the shutdown is not presented; only the dialog showing the progress in shutting down the running components. Parameters: **pReason** - The reason for the shutdown, this will be recorded in the log. Returns: Nothing
`public static boolean isValidUser(String pUserName, String pPassword)` This method whether the user name and password passed in are valid credentials for a ROES Server user. Parameters: **pUserName** - The user name. **pPassword** - The password. Returns: A Boolean set to true if there is a record in the Users table with a matching user name and password.

Description

This method is reserved to signed scripts

```
public static String morphCrop( Hashtable pSystemData,
                                File pImageFile,
                                String pOriginalCrop,
                                String pNewNodeBounds ) throws Exception
```

This method returns a modified crop for the new node bounds.

Parameters:

pSystemData - The system data object.
pImageFile - The file containing the image the crop is applied to
pOriginalCrop - The original crop that was applied
pNewNodeBounds - The new node bounds of the node into which the image is being rendered

Returns:

The modified crop value.

Throws:

An exception if the calling script is not signed.

RSOrder - This class represents order data in the server

```
public class RSOrder
{
      public RSXMLElement getOrderXMLData();
      public String getOrderID();
      public String getCustomerID();
      public String getCustomerOrderID();
      public Date getDateProcessed();
      public Date getDateRendered();
      public boolean imagesAreColorCorrected();
      public String getEPayTransactionID();
      public ArrayList<RSProductionItem> getProductionItems();
      public ArrayList<File> getImageFiles();
      public RSEventsData getEventsData();
      public void deleteOrder( boolean pDeleteImages,
                               boolean pDeleteAttachments,
                               boolean pDeleteClientGeneratedImages,
                               boolean pDeleteReports );
      public ArrayList<String> getOriginalOrderIDs();

}
```

Description
`public RSXMLElement getOrderXMLData();` This method returns the root element of the order xml data. This is the data that is in the order.xml file. Parameters: None. Returns: An RSXMLElement that represents the root element of the order.
`public String getOrderID();` This method returns the order id of the order. Parameters: None. Returns: A String containing the order id of the order.
`public String getCustomerID();` This method returns the customer id of the customer that placed the order. Parameters: None. Returns: A String containing the customer id of the customer that placed the order.
`public String getCustomerOrderID();` This method returns the order id of the ROES client assigned to the order. Parameters: None. Returns: A String containing the order id that the ROES client assigned to the order.

Description
`public Date getDateProcessed();` This method returns the date that this order was processed in the server. Parameters: None. Returns: A Date that the order was processed in the server.
`public Date getDateRendered();` This method returns the last date that an item from this order was rendered. Parameters: None. Returns: A Date that the order was processed in the server.
`public boolean imagesAreColorCorrected();` This method returns whether the images have been color corrected in the ROES Color Correction system or not. Parameters: None. Returns: True if the image have been color corrected in the ROES Color Correction system; otherwise false.
`public String getEPayTransactionID();` This method returns the credit card transaction id if this order was paid for by credit card. Parameters: None. Returns: A String containing the credit card transaction id if this order was paid for by credit card.

Description
`public ArrayList<RSProductionItem> getProductionItems();` This method returns a list of the production items in this order. Parameters: None. Returns: An ArrayList<RSProductionItem> containing the production items in this order.
`public ArrayList<File> getImageFiles();` This method returns a list of the image files used in this order. Parameters: None. Returns: An ArrayList<File> containing the image files used in this order.
`public RSEventsData getEventsData();` This method returns an object that encapsulates the events data associated with this order. Parameters: None. Returns: An RSEventData object containing the events data included in this order.

Description

```
public void deleteOrder( boolean pDeleteImages,
                         boolean pDeleteAttachments,
                         boolean pDeleteClientGeneratedImages,
                         boolean pDeleteReports );
```

This method deletes this order from the system.

Parameters:

pDeleteImages - Indicates whether the images for this order should be deleted.
pDeleteAttachments - Indicates whether the attachments for this order should be deleted.
pDeleteClientGeneratedImages - Indicates whether the client generated images for this order should be deleted.
pDeleteReports - Indicates whether any generated reports for this order should be deleted.

Returns:

Returns: Nothing

```
public ArrayList<String> getOriginalOrderIDs();
```

This method returns a the list of order IDs that the images of this order are re-ordered from.

Parameters:

None.

Returns:

An ArrayList of String objects containing the order ids of any orders that the images of this order are re-ordering from.

RSXMLElement - This class represents the data of an XML element

---

```
public class RSXMLElement
{
      public String getTag();
      public ArrayList<String> getAttributes();
      public String getAttribute( String pName);
      public String getAttribute( String pName, String pDefault);
      public String getData();
      public String getCData();
      public String getComment();
      public ArrayList<RSXMLElement> getChildren();
      public ArrayList<RSXMLElement> getChildren( String pBaseTag );
      public RSXMLElement getFirstChildofTag( String pTag );
      public List<String> getChildrensTags();
}
```

Description
`public String getTag();` This method returns the tag name of this element. Parameters: None. Returns: A String containing the tag name of this element.
`public ArrayList<String> getAttributes();` This method returns a list of the names of the attributes belonging to this element. Parameters: None. Returns: An ArrayList<String> containing the names of the attributes belonging to this element.
`public String getAttribute( String pName);` This method returns value of an attribute belonging to this element. Parameters: **pName** - The name of the attribute. Returns: A String containing the value of the attributes.
`public String getAttribute( String pName, String pDefault);` This method returns value of an attribute belonging to this element or an alternate value if this element does not have the attribute. Parameters: **pName** - The name of the attribute. **pDefault** - A value to return if the element does not have the attribute. Returns: A String containing the value of the attributes.

Description
`public String getData();` This method returns the data of this element. Parameters: None. Returns: A String containing data name of this element.
`public String getCData();` This method returns the cdata of this element. Parameters: None. Returns: A String containing the cdata of this element.
`public String getComment();` This method returns the comment of this element. Parameters: None. Returns: A String containing the comment of this element.
`public ArrayList<RSXMLElement> getChildren();` This method returns a list of the child elements of this element. Parameters: None. Returns: An ArrayList<String> containing the child elements of this element.

Description
`public ArrayList<RSXMLElement> getChildren( String pBaseTag );` This method returns a list of the child elements of this element that have a given tag name. Parameters: **pBaseTag** - The tag name to look for. Returns: An ArrayList<String> containing the child elements of this element that have the given tag name.
`public RSXMLElement getFirstChildofTag( String pTag );` This method returns a list of the first child element of this element that has a given tag name. Parameters: **pTag** - The tag name to look for. Returns: An RSXMLElement that represents the first child element of this element that has the given tag name.
`public List<String> getChildrensTags();` This method returns a list of the tag names of any child elements of this element. Parameters: None. Returns: A List<String> containing the names of child elements of this element.

RSWritableXMLElement - This class represents a modifiable version of the data of an XML element. Note: This class should only be used on xml elements that come from an Agent or Printer's batch elements. If you modify XML elements in the server other than those in a batch, you may break the operation of the server and lose or damage order data.

```
public class RSWritableXMLElement extends RSXMLElement
{
      public RSWritableXMLElement( RSXMLElement pElement );
      public void setTag(String pTag);
      public void setData(String pData);
      public void setCData(String pCData);
      public void setAttribute(String pAttributeName, String pValue);

      // The following methods are accessible only from signed scripts
```

```
      public static RSWritableXMLElement init(
                                    Hashtable pSystemData,
                                    String pXMLString ) throws Exception
      public String toXMLString( Hashtable pSystemData ) throws Exception

}
```

Description
`public RSWritableXMLElement( RSXMLElement pElement );` The constructor for RSWritableXMLElement. The constructor is passed an RSXMLElement that this object will wrap around. When any data is set, it is pElement's data that will be set. Parameters: **pElement** - The element whose data will be modified by this class.
`public void setTag(String pTag);` This method sets the tag of the underlying element. Parameters: **pTag** - The tag value to be assigned to the underlying element. Returns: Nothing.
`public void setData(String pData);` This method sets the data of the underlying element. Parameters: **pData** - The data value to be assigned to the underlying element. Returns: Nothing.
`public void setCData(String pCData);` This method sets the cdata of the underlying element. Parameters: **pCData** - The cdata value to be assigned to the underlying element. Returns: Nothing.

Description

```
public void setAttribute(String pAttributeName, String pValue);
```

This method sets the value of an attribute of the underlying element.

Parameters:

pAttributeName - The name of the attribute whose value is to be set on the underlying element.
pValue - The value to be assigned to attribute of the underlying element.

Returns:

Nothing.

This method is reserved to signed scripts

```
public static RSWritableXMLElement init(
                                    Hashtable pSystemData,
                                    String pXMLString ) throws Exception
```

This method returns a new RSWriteableXMLElement object that references the root of the xml data passed to it as a string.

Parameters:

pSystemData - The system data object.
pXMLString - The XML data to be parsed

Returns:

This method returns a new RSWriteableXMLElement object that references the root of the xml data that was passed to it as a string.

Throws:

An exception if the calling script is not signed.

This method is reserved to signed scripts

```
public String toXMLString( Hashtable pSystemData ) throws Exception
```

This method returns the XML hierarchy that it references as an XML string.

Parameters:

pSystemData - The system data object.

Returns:

This method returns the XML hierarchy that it references as an XML string. The XML String will be UTF-8 encoded and it will not contain an XML declaration.

Throws:

An exception if the calling script is not signed.

RSEventsData - This class represents the event data contained in an order

```
public class RSEventsData
{
	public RSEventsRecord getRecord( String pRecordID);
	public ArrayList<RSEventsTable> getTables();
}
```

Description
`public RSEventsRecord getRecord( String pRecordID);` This method returns the record identified by pRecordID Parameters: **pRecordID** - The id of the record you are after. Returns: An RSEventsRecord identified by pRecordID or null if there was no match.
`public ArrayList<RSEventsTable> getTables();` This method adds a batch to this job queue. Parameters: None. Returns: An ArrayList of tables contained within the event data.

RSEventsTable - This class represents a table contained within the event data of an order

```
public class RSEventsTable
{
	public ArrayList<String> getFields();
	public String getTableID();
	public ArrayList<RSEventsRecord> getRecords();
}
```

Description
`public ArrayList<String> getFields();` This method returns the list of names of the fields of this table. Parameters: None. Returns: An ArrayList of Strings containing the names of the fields of this table.
`public String getTableID();` This method returns the identifier of this table. Parameters: None. Returns: An String containing the identifier of this table.
`public ArrayList<RSEventsRecord> getRecords();` This method retrieves the list of records from this table Parameters: None. Returns: An ArrayList of the records contained in this table.

RSEventsRecord - This class represents a record from a table

```
public class RSEventsRecord
{
      public String getIndex();
      public ArrayList<RSProductionItem> getProductionItems( RSOrder pOrder);
      public ArrayList<RSXMLElement> getOrderItemUnits( RSOrder pOrder );
      public ArrayList<RSXMLElement> getOrderItems( RSOrder pOrder );
      public Iterator getFieldNamesIter();
      public String getFieldValue( String pFieldName );
}
```

Description
`public String getIndex();` This method returns the index of the record Parameters: None. Returns: A String containing the index of this record.
`public ArrayList<RSProductionItem> getProductionItems( RSOrder pOrder);` This method returns a list of production items that were created from this record. Parameters: **pOrder** - The order that this event data belongs to. Returns: An ArrayList of production items created from this record.
`public ArrayList<RSXMLElement> getOrderItemUnits( RSOrder pOrder );` This method returns a list of xml elements from the order that correlate to the production items that were created from this record. Parameters: **pOrder** - The order that this event data belongs to. Returns: An ArrayList of xml elements unit templates created from this record.
`public ArrayList<RSXMLElement> getOrderItems( RSOrder pOrder );` This method returns a list of xml elements from the order that correlate to the order items that were created from this record. Parameters: **pOrder** - The order that this event data belongs to. Returns: An ArrayList of item level xml elements templates created from this record.

Description
`public Iterator getFieldNamesIter()` This method returns an iterator for the names of the fields in this record. Parameters: None. Returns: an Iterator for the names of the fields in this record.
`public String getFieldValue( String pFieldName );` This method returns the value of a particular field in the record. Parameters: **pFieldName** - The name of the field whose value you want to retrieve. Returns: A String containing the value of the field.

RSJobQueue - This class represents a job queue in the workstation

---

```
public class RSJobQueue
{
      public String getName();
      public void addBatch( ArrayList<RSServerJob> pBatch );
      public void addBatch( ArrayList<RSProductionItem> items,
                            Long pQuantity ) throws Exception;
      public ArrayList<RSServerJob> getNextBatch();
}
```

Description
`public String getName();` This method returns the name of this job queue. Parameters: None. Returns: A String containing the name of this job queue.

Description
`public void addBatch( ArrayList<RSServerJob> pBatch );` This method adds a batch to this job queue. Parameters: **pBatch** - An ArrayList<RSServerJob> of the jobs to be added as a batch to the job queue. Returns: Nothing.

`public void addBatch( ArrayList<RSProductionItem> items,`
`Long pQuantity ) throws Exception;`
 	
This method adds a batch of production items to this job queue with a given quantity. RSServerJob items will be created from the RSProductionItems and those will be what is actually added to the batch.
 	
Parameters:
 	
items - An ArrayList<RSProductionItem> of the items to be added as a batch to the job queue.
pQuantity - The quantity that should be set for the items in the batch.
 	
Returns:
 	
Nothing.

`public ArrayList<RSServerJob> getNextBatch();` This method retrieves a batch from this job queue and returns it. Parameters: None. Returns: An ArrayList<RSServerJob> that contains the list of jobs of the batch.

RSProductionItem - Represents an atomic produceable item in the server, i.e. an 8x10 unit or a page of a book. This is basically the data that an RSServerJob would be read from.

```
public class RSProductionItem
{
	public RSXMLElement getTemplatesXML();
	public String getProperty( String attributeName );
	public String getOrderID();
	public long getProductionItemID();
}
```

Description
`public RSXMLElement getTemplatesXML();` This method returns the template element of this production item. Parameters: None. Returns: An RSXMLElement that represents the template element of this production item.
`public String getProperty( String attributeName );` This method returns a property of this production item's template element. Parameters: **attributeName** - The name of the attribute to get. Returns: A String containing value of the property of this production item's template element.
`public String getOrderID();` This method returns the order id of the order this production item belongs to. Parameters: None. Returns: An String that contains the order id of the order this production item belongs to.
`public long getProductionItemID();` This method returns the production item id of this item. Parameters: None. Returns: An long that contains the production item id of this item.

RSServerJob - Represents a job in the server that could be in placed in a job queue

```
public class RSServerJob
{
      public String getBatchID()
      public String getOrderID()
      public String getProductionItemID()
      public RSXMLElement getTemplate()
      public File getRenderedFile()
      public int getRenderedWidth()
      public int getRenderedHeight()
      public int getRenderedArea()
      public int getOrdinalInBatch()
      public int getPageCount()
      public int getPageCellCount()
      public String getBatchIDSet()

      // The following methods are accessible only from signed scripts
      public static RSServerJob init(
                                 Hashtable pSystemData,
                                 String pBatchID,
                                 String pOrderID,
                                 String pProductionItemID,
                                 RSXMLElement pTemplateXML ) throws Exception

}
```

Description
`public String getBatchID()` This method returns the id of the batch that this job belongs to. Parameters: None. Returns: A String containing the id of the batch that this job belongs to.
`public String getOrderID()` This method returns the id of the order that this job belongs to. Parameters: None. Returns: A String containing the id of the order that this job belongs to.

Description
`public String getProductionItemID()` This method returns the id of the production item id that this job derived from. Parameters: None. Returns: A String containing the production item id that this job derived from.
`public RSXMLElement getTemplate()` This method returns the root template element of this job. Parameters: None. Returns: An RSXMLElement that represents the root template element of this job
`public File getRenderedFile()` This method returns rendered file of this job. If the job has not been rendered to a file then it returns null. Parameters: None. Returns: An File that represents the rendered file of this job or null if the job has not been rendered to a file.
`public int getRenderedWidth()` This method returns the width of the rendered image of this job in pixels. If the job has not been rendered then this value will be undefined. Parameters: None. Returns: An int that represents the width of the rendered image of this job in pixels.

Description
`public int getRenderedHeight()` This method returns the height of the rendered image of this job in pixels. If the job has not been rendered then this value will be undefined. Parameters: None. Returns: An int that represents the height of the rendered image of this job in pixels.
`public int getRenderedArea()` This method returns the area (width x height) of the rendered image of this job in pixels. If the job has not been rendered then this value will be undefined. Parameters: None. Returns: An int that represents the area (width x height) of the rendered image of this job in pixels.
`public int getOrdinalInBatch()` This method returns the ordinal position of this job in it's batch, i.e. where it falls in the list of jobs in its batch. Parameters: None. Returns: An int that represents the ordinal position of this job in it's batch.
`public int getPageCount()` This poorly named method returns the number of jobs in this job's batch. Parameters: None. Returns: An int that represents the number of jobs in this job's batch.

Description
`public int getPageCellCount()` This method returns the number of page cells in the production book that this job derived from. This value will only be defined if this job was created by a Book Agent. Parameters: None. Returns: An int that represents the number of page cells in the production book that this job derived from.
`public String getBatchIDSet()` This method returns the set of batch id's that belong to the same set as this job. This value will only be defined if this job was created by a Book Agent. Parameters: None. Returns: An String that contains a comma separated list of batch id's that constitute all the batches that belong in the set.

Description

This method is reserved to signed scripts

```
public static RSServerJob init(
                                Hashtable pSystemData,
                                String pBatchID,
                                String pOrderID,
                                String pProductionItemID,
                                RSXMLElement pTemplateXML ) throws
                                Exception
```

This method returns a new RSServerJob object that references the template XML that was passed to it.

Parameters:

pSystemData - The system data object.
pBatchID - The ID assigned to the batch to which this RSServerJob belongs.
pOrderID - The ID of the order from which our owning batch originated.
pProductionItemID - The id of the production item that this RSServerJob links back to.
pTemplateXML - The XML data of the template this RSServerJob wraps.

Returns:

This method returns a new RSServerJob object.

Throws:

An exception if the calling script is not signed.

RSCatalog - Represents the catalog templates data as the client would see if for this order, including customer specific catalog's etc. This is the catalog in the broadest sense, not to be confused with the catalog xml element within the…catalog.

```
public class RSCatalog
{
      public RSXMLElement getCatalogData()
      public ArrayList<RSXMLElement> getLeafTemplates()
      public Hashtable<String, RSXMLElement> getLeafTemplatesIndexed()
      public RSXMLElement getTemplateByUID( String pUID )
}
```

Description
`public RSXMLElement getCatalogData();` This method returns the root element of the catalog xml hierarchy. Parameters: None. Returns: An RSXMLElement that represents the root element of the catalog xml hierarchy.
`public ArrayList<RSXMLElement> getLeafTemplates();` This method returns a list of template elements from the catalog that contain no other templates. Parameters: None. Returns: An ArrayList of RSXMLElement's that are leaf templates in the catalog.
`public Hashtable<String, RSXMLElement> getLeafTemplatesIndexed();` This method returns a hash table of template elements from the catalog that contain no other templates. The keys to the hash table are the u_id's of templates. Parameters: None. Returns: A Hashtable of RSXMLElement's that are leaf templates in the catalog keyed to their u_id's.
`public RSXMLElement getTemplateByUID( String pUID );` This method returns the element from the catalog with the given u_id. Parameters: **pUID** - The u_id attribute of the template we are looking for. Returns: An RSXMLElement element who's u_id attribute or null if there are no matches.

RSAgent - This class represents an Agent in the server

```
public class RSAgent
{
      public String getName();
      public void stop();
      public void start();
      public boolean isRunning();
      public boolean isStopped();
      public void reprocessErrorBatches( String[] pBatchIDs );
      public void removeErrorBatches( String[] pBatchIDs );
}
```

Description
`public String getName();` This method returns the name assigned to this agent. Parameters: None. Returns: A String containing the name of the agent.
`public void stop();` This method stops the execution of the agent. If the agent is not currently running, this call will be ignored. Parameters: None. Returns: Nothing.
`public void start();` This method starts the execution of the agent. If the agent is not currently stopped, this call will be ignored. Parameters: None. Returns: Nothing.

Description
`public boolean isRunning();` This method returns true if the agent is currently running; otherwise it returns false. Parameters: None. Returns: A boolean indicating if the agent is currently running.
`public boolean isStopped();` This method returns true if the agent is currently stopped; otherwise it returns false. Parameters: None. Returns: A boolean indicating if the agent is currently stopped.
`public void reprocessErrorBatches( String[] pBatchIDs );` This method will cause the agent to reprocess any batches identified in pBatchIDs that are currently in the agents erred batches list. Parameters: **pBatchIDs** - An array of Strings containing the batch IDs to be reprocessed. Returns: Nothing.
`public void removeErrorBatches( String[] pBatchIDs );` This method will cause the agent to delete any batches identified in pBatchIDs that are currently in the agents erred batches list. Parameters: **pBatchIDs** - An array of Strings containing the batch IDs to be deleted. Returns: Nothing.

RSListenerAgent - This class represents an Listener Agent in the server

```
public class RSListenerAgent
{
	public String getName();
	public void stopListening();
	public void startListening();
	public boolean isListening();
}
```

Description
`public String getName();` This method returns the name assigned to this listener agent. Parameters: None. Returns: A String containing the name of the listener agent.
`public void stopListening();` This method stops the agent from listening for any events. If the agent is not currently listening, this call will be ignored. Parameters: None. Returns: Nothing.
`public void startListening();` This method causes the agent to listen for events. If the agent is already listening for events, this call will be ignored. Parameters: None. Returns: Nothing.

Description
`public boolean isListening();` This method returns true if the agent is currently listening for its events; otherwise it returns false. Parameters: None. Returns: A boolean indicating if the agent is currently listening.

RSWebHandler - This class is a utility class that makes it easy setup a web server in your script. The functionality is loosely pattered after nodejs and Express. It allows a user to install "route handlers". A route is a path to an endpoint on the web server. The elements of the path can contain literal values or variables. If the element is a variable then it will be preceded with a ":". Variable elements of a route path are automatically parsed and placed into a structure that can be accessed by a route handler. Route handlers can be added to respond to GET, POST, PUT or DELETE requests. A user can add multiple route handlers for the same route. The route handlers are saved in the order in which they were added and when a request arrives, the first route handler that matches the request will be executed. A route handler can satisfy the request or it can act as a middleware component and only do some portion of the processing and then pass execution onto the next matching route handler.

```
public class RSWebHandler
{
	public RSWebHandler( int pPort );
	public void setErrorRouteHandler( IRSRouteHandler pErrorRouteHandler );
	public void get( String pRoute, IRSRouteHandler pHandler );
	public void post( String pRoute, IRSRouteHandler pHandler );
	public void put( String pRoute, IRSRouteHandler pHandler );
	public void delete( String pRoute, IRSRouteHandler pHandler );
	public void use( String pRoute, IRSRouteHandler pHandler );
	public void start() throws IOException;
	public void stop( int pMaxWaitTime );
}
```

Description
`public RSWebHandler( int pPort );` This constructor returns an RSWebHandler that when started, will listen for incoming connections on the port that it was passed. Parameters: **pPort** - An integer that represents the IP port that you want this RSWebHandler to listen on, Returns: The constructed RSWebHandler object.
`public void setErrorRouteHandler( IRSRouteHandler pErrorRouteHandler );` This method sets the route handler that is to be executed in the case where another route handler has thrown an exception out of its execute method. By default, the WebHandler creates an error route handler that returns a 500 status code and a JSON object containing an error attribute that is set to the message from the exception. Using this method, the user can control how errors are to be handled. Parameters: **pErrorRouteHandler** - An IRSRouteHandler that is to be used for error handling. Returns: Nothing.
`public void get( String pRoute, IRSRouteHandler pHandler );` This method adds a route handler for a given “route”, or path, that will be checked for a match when a GET request is made to the server. Parameters: **pRoute** - A String representing the path to the endpoint as a regular expression. **pHandler** - An IRSRouteHandler that is to be executed when a request is made to this route . Returns: Nothing.

Description
`public void post( String pRoute, IRSRouteHandler pHandler );` This method adds a route handler for a given “route”, or path, that will be checked for a match when a POST request is made to the server. Parameters: **pRoute** - A String representing the path to the endpoint as a regular expression. **pHandler** - An IRSRouteHandler that is to be executed when a request is made to this route . Returns: Nothing.
`public void put( String pRoute, IRSRouteHandler pHandler );` This method adds a route handler for a given “route”, or path, that will be checked for a match when a PUT request is made to the server. Parameters: **pRoute** - A String representing the path to the endpoint as a regular expression. **pHandler** - An IRSRouteHandler that is to be executed when a request is made to this route . Returns: Nothing.
`public void delete( String pRoute, IRSRouteHandler pHandler );` This method adds a route handler for a given “route”, or path, that will be checked for a match when a DELETE request is made to the server. Parameters: **pRoute** - A String representing the path to the endpoint as a regular expression. **pHandler** - An IRSRouteHandler that is to be executed when a request is made to this route . Returns: Nothing.

Description
`public void use( String pRoute, IRSRouteHandler pHandler );` This method adds a route handler for a given “route”, or path, that will be checked for a match when any type of request: GET, POST, PUT or DELETE, is made to the server. Parameters: **pRoute** - A String representing the path to the endpoint as a regular expression. **pHandler** - An IRSRouteHandler that is to be executed when a request is made to this route . Returns: Nothing.
`public void start() throws IOException;` This method starts the web server listening on its port. Parameters: None. Returns: Nothing. Throws: An exception if one is thrown while starting the server.
`public void stop( int pMaxWaitTime );` This method stops the server from listening for any additional incoming requests and waits for any current requests to finish processing, up to a maximum of the number of second identified. Parameters: **pMaxWaitTime** - The maximum time to wait for current requests to complete in seconds. Returns: Nothing.

IRSRouteHandler - An interface that defines the execute method for a route handler

---

```
public interface IRSRouteHandler
{
      public void execute( HashMap<String, Object> pParams,
                           HttpExchange pHttpExchange,
                           IRSRouteUtil pUtil ) throws Exception;
}
```

Description

```
public void execute( HashMap<String, Object> pParams,
                              HttpExchange pHttpExchange,
                              IRSRouteUtil pUtil ) throws Exception;
```

This method is called when the route handler is to be executed. This happens when a request is made on the route that this handler is associated with. It is passed a HashMap containing all the variables that were parsed as part of the route plus any other additions that route handlers that have executed before us made. It is possible for us to add additional items to the HashMap that are for use by subsequent route handlers.

Parameters:

pParams - A HashMap containing any variables parsed from the route plus additions.
pHttpExchange - An HttpExchange object that provides access to the request and response components.
pUtil - An IRSRouteUtil object that provides handy functions like “next()”

Returns:

Nothing.

Throws:

An exception if one is thrown while executing.

IRSRouteUtil - An interface that defines utility methods for a route handler

---

```
public interface IRSRouteUtil extends ISWRouteUtil
{
      public void next();
      public void writeResponseData( int pResponseCode,
                                    String pContentType,
                                    byte[] pResponseData ) throws IOException;
}
```

Description
`public void next();` This method will cause the server to search for the next matching route handler and if it finds one, then it will execute it. This method would typically be called by a route handler when it wants to pass on the execution of a request to the next route handler. Parameters: None. Returns: Nothing.

`public void writeResponseData( int pResponseCode,`
`String pContentType,`
`byte[] pResponseData ) throws IOException;`
 	
This method will write out a response to the current request; allowing the caller to set the response code, the Content-Type response header field and associated data. This method would typically be called by a route handler as a convenient way to send back a response to a request.
 	
Parameters:
 	
pResponseCode - An int value that will be used as the response code.
pContentType - A String containing value to be used as the Content-Type.
pResponseData - A byte array containing the data to be send back with the response.
 	
Returns:
 	
Nothing.
 	
Throws:
 	
An exception if one is thrown while writing the response data

---

RSServerEventListener - An abstract class that represents an event listener. You must subclass this class and implement the notified method.

```
public abstract class RSServerEventListener
{
      public abstract void notified( String pEventName, Object pEventData );
}
```

Description

```
public abstract void notified( String pEventName, Object pEventData );
```

This method is called when an event that this listener is listening for occurs.

Parameters:

pEventName - The name of the event that we are being notified about.
pEventData - Data that belongs to the event

Returns:

Nothing.

RSServerEvents - A class that contains constants for event related data, e.g. ID's of event generated by the system

---

```
public class RSServerEvents
{
      public static final String EVENT_DATABASE_CHANGED       =
                                                    "DATABASE_CHANGED";
      public static final String EVENT_OUR_IP_ADDRESS_CHANGED =
                                                    "OUR_IP_ADDRESS_CHANGED";
      public static final String WORKSTATION_ADDED            =
                                                    "WORKSTATION_ADDED";
      public static final String WORKSTATION_REMOVED          =
                                                    "WORKSTATION_REMOVED";
      public static final String ON_LOCAL_LAUNCH              =
                                                    "ON_LOCAL_LAUNCH";
      public static final String ON_LOCAL_SHUTDOWN            =
                                                    "ON_LOCAL_SHUTDOWN";
      public static final String ON_USER_LOGGED_IN            =
                                                    "ON_USER_LOGGED_IN";
      public static final String ON_USER_LOGGED_OUT           =
                                                    "ON_USER_LOGGED_OUT";
      public static final String ON_LOCAL_BATCH_COMPLETE      =
                                                    "ON_LOCAL_BATCH_COMPLETE";
      public static final String ON_LOCAL_BATCH_ERROR         =
                                                    "ON_LOCAL_BATCH_ERROR";
      public static final String ON_ORDER_PROCESSED           =
                                                    "ON_ORDER_PROCESSED";
      public static final String ON_LOCAL_ORDER_PROCESSED     =
                                                    "ON_LOCAL_ORDER_PROCESSED";
      public static final String ON_ORDER_PROCESS_ERROR       =
                                                    "ON_ORDER_PROCESS_ERROR";
      public static final String ON_SEND_LOG                  =
                                                    "ON_SEND_LOG";
      public static final String ON_ORDER_DELETED             =
                                                    "ON_ORDER_DELETED";
      public static final String ON_LOCAL_ORDER_DELETED       =
                                                    "ON_LOCAL_ORDER_DELETED";
      public static final String ON_FLUSH_CACHED_ORDER_DATA   =
```

```
                                                    "ON_FLUSH_CACHED_ORDER_DATA";
      public static final String ON_STORE_CUSTOMER_DATA_ERROR =
                                                "ON_STORE_CUSTOMER_DATA_ERROR";
      public static final String ON_EPAY_ERROR                  =
                                                    "ON_EPAY_ERROR";

      // Special events only injected into a Custom Listener, not broadcast
      public static final String STARTED_LISTENING              =
                                                    "STARTED_LISTENING";
      public static final String STOPPED_LISTENING              =
                                                    "STOPPED_LISTENING";

       // Event Scope Constants
       public static final int EVENT_SCOPE_LOCAL = 0;
       public static final int EVENT_SCOPE_REMOTE = 1;
       public static final int EVENT_SCOPE_ALL = 2;
}
```

Description
`public static final String EVENT_DATABASE_CHANGED;` This event is broadcast to the listeners on the local workstation when the database settings have been changed. Associated data: None.
`public static final String EVENT_OUR_IP_ADDRESS_CHANGED;` This event is broadcast to the listeners on the local workstation when the ensemble ip address of this workstation changes. Associated data: A String containing the ip address (not port) of this workstation.
`public static final String WORKSTATION_ADDED;` This event is broadcast to the listeners on the local workstation whenever another workstation has been detected. Associated data: A String containing the ip address and port, separated by a colon, of the added workstation.
`public static final String WORKSTATION_REMOVED;` This event is broadcast to the listeners on the local workstation whenever another workstation has been removed by either shutting down or by lost communication with that workstation. Associated data: A String containing the ip address and port, separated by a colon, of the added workstation.
`public static final String ON_LOCAL_LAUNCH;` This event is broadcast to the listeners on the local workstation when the server is launched on this workstation. Associated data: None.

Description
`public static final String ON_LOCAL_SHUTDOWN;` This event is broadcast to the listeners on the local workstation when this workstation is shutting down. It is issued just prior to the processing, running agents and listeners being stopped. Associated data: None.
`public static final String ON_USER_LOGGED_IN;` This event is broadcast to listeners on all workstations when a user has logged into a workstation. Associated data: A String containing the name of the user that logged in followed by "[,]" followed by the name of the workstation they logged in on.
`public static final String ON_USER_LOGGED_OUT;` This event is broadcast to listeners on all workstations when a user has logged out of workstation. Associated data: A String containing the name of the user that logged in followed by "[,]" followed by the name of the workstation they logged off.
`public static final String ON_LOCAL_BATCH_COMPLETE;` This event is broadcast to the listeners on the local workstation when any Printer/Agent completes the processing of a batch. Associated data: A Hashtable with the following keys: **AgentName** - A String containing the name of the agent that completed processing. **BatchID** - A String containing the id of the batch that was processed. **OrderID** - A String containing the id of the order that the jobs in the batch belong to. **TheBatch** - An ArrayList<RSServerJob> representing the batch that was processed.
`public static final String ON_LOCAL_BATCH_ERROR;` This event is broadcast to the listeners on the local workstation when any Printer/Agent completes the processing of a batch. Associated data: A Hashtable with the following keys: **AgentName** - A String containing the name of the agent that completed processing. **BatchID** - A String containing the id of the batch that was processed. **OrderID** - A String containing the id of the order that the jobs in the batch belong to. **TheBatch** - An ArrayList<RSServerJob> representing the batch that was processed. **Exception** - The exception that was thrown while processing the batch.
`public static final String ON_ORDER_PROCESSED;` This event is broadcast to listeners on all workstations when an order is initially processed by a workstation. Associated data: A String containing the order ID of the processed order.

Description
`public static final String ON_LOCAL_ORDER_PROCESSED;` This event is broadcast to the listeners on the local workstation when an order is successfully processed by this workstation. Associated data: A Hashtable with the following keys: **OrderID** - A String containing id of the order processed. **CustomerOrderID** - A String containing the client generated order id for the processed order. **OrderFileName** - A String containing the client generated order id for the processed order.
`public static final String ON_ORDER_PROCESS_ERROR;` This event is broadcast to the listeners on the local workstation when an order errs while being processed by this workstation. Associated data: A Hashtable with the following keys: **OrderID** - A String containing id of the order processed. **CustomerOrderID** - A String containing the client generated order id for the processed order. **OrderFileName** - A String containing the client generated order id for the processed order. **Exception** - The exception object thrown during processing.
`public static final String ON_SEND_LOG;` This event is broadcast to the listeners on all workstations when the workstations are being directed to send their logs to the email address that is set up to receive processing failure notifications. Associated data: A String containing “true” or “false” indicating whether the logs should also be sent to SoftWorks.
`public static final String ON_ORDER_DELETED;` This event is broadcast to the listeners on all workstations when an order has been deleted from the system. Associated data: A String indicating the lab assigned id of the order that was deleted.
`public static final String ON_LOCAL_ORDER_DELETED;` This event is broadcast to the listeners on the local workstation when an order has been deleted on this workstation. Associated data: A Hashtable with the following keys: **OrderID** - A String containing id of the order that was deleted.
`public static final String ON_FLUSH_CACHED_ORDER_DATA;` This event is broadcast to the listeners on all workstations when the system wants all cached data about an order to be forgotten. This might happen when an order is modified in some way or when a user explicitly select the “Flush Cached Data” from the orders table contextual menu. Associated data: A String containing the order ID of the order whose cached data is to be flushed.

Description
`public static final String ON_STORE_CUSTOMER_DATA_ERROR;` This event is broadcast to the listeners on the local workstation when an error occurs while the Customer data of an order is being stored in the Customers table. Associated data: A Hashtable with the following keys: **OrderID** - A String containing id of the order processed. **CustomerOrderID** - A String containing the client generated order id for the processed order. **Exception** - The exception object thrown during processing.
`public static final String ON_EPAY_ERROR;` This event is broadcast to the listeners on the local workstation when an error occurs while trying to process the credit card on this workstation. Associated data: A Hashtable with the following keys: **TheOrder** - An RSOrder object representing the order. **ErrorMessage** - A String containing the error message returned by the payment gateway.
`public static final String STARTED_LISTENING;` This event is not broadcast but rather injected into a Script Listener Agent when the Agent starts listening for events if that Agent is set to receive listening state change events. Associated data: None.
`public static final String STOPPED_LISTENING;` This event is not broadcast but rather injected into a Script Listener Agent when the Agent stops listening for events if that Agent is set to receive listening state change events. Associated data: None.
`public static final int EVENT_SCOPE_LOCAL;` An identifier that indicates an event should be broadcast to the local listeners on this workstation only.
`public static final int EVENT_SCOPE_REMOTE;` An identifier that indicates an event should be broadcast to listeners on all the remote workstation only.
`public static final int EVENT_SCOPE_ALL;` An identifier that indicates an event should be broadcast to the all listeners, local and remote.

Script Agent

Modifying the batch that is passed on to another Agent is reserved to signed scripts

It is possible in a signed script to modify the batch that was handed to the Script Agent such that the modified batch is then passed on the next agent. This is accomplished by adding an item to the gMacros object with a key called "SWNewJobBatch". The object should be defined as an ArrayList<RSServerJob>. The jobs referenced in the list will replace the list of jobs in the original batch passed to the Script Agent and be passed on the next agent.

It is also possible, in a signed script, to create additional batches of RSServerJob objects that will get enqueued, possibly to multiple queues. This is accomplished by adding an item to the gMacros object with a key called "SWNewJobBatches". The object should be defined as a Hashtable<String, ArrayList<ArrayList<RSServerJob>>>. This is a Hashtable where the key is the name of the queue that a list of batches should be placed into. The value is an ArrayList of batches; each batch is an ArrayList of RSServerJob object. These batches will get enqueued in addition to the original batch getting passed on to the next agent or dropped if this Script Agent is not set to re-enqueue the batch to another queue.

Exceptions

Generally exceptions are handled as they normally are depending on the language the script is using but what happens when an exception is thrown out of my script? The answer depends on the context of the script:

Throwing exceptions out of a script in a Report

In a report, Report generation will be interrupted and a stack trace will be written to system output. If the report was in a Report Agent then the exception will be written to the Server log as well as the log for that agent.

Throwing exceptions out of a script in a Script Agent

In a script agent, the exception will be written to the server log and the log for the agent.

Throwing exceptions out of a script in a Custom Listener

In a Listener, if the script was triggered by a user clicking on the "Test Now" button then a dialog will presented with the exception information, otherwise the exception information will be written to the server log.

Injecting your own Events into the system

From within a script it is possible to inject your own events into the system. Using the RSScriptUtilities class's notifyListeners, notifyLocalListeners or notifyRemoteListeners, you can broadcast your own event. notifyListeners will broadcast the event to any listeners on all workstations including the one we are running on. notifyLocalListeners will only broadcast the event to listeners on the workstation that the script is running on. notifyRemoteListeners will broadcast the event to all listeners on all workstations other than the one on which the script is running. Each of these methods take two parameters; the name of the event and an object to be passed to listeners for this event. **Note that there is currently a limitation on event data that is to be sent to workstations other than the one sending the event; the event data must be a String.**

So if we knew that there were listeners listening for a "ShowInDialog" event and upon receipt of that event would put the event data up in a dialog, we could trigger those scripts by placing this in our own script:

```
// Begin script
RSScriptUtilities.notifyListeners( "ShowInDialog", "Hello World");
// End Script
```

Debugging

Simple debugging of your script can be done by using the gOutputWriter but once you have more than a handful of lines of code it quickly becomes clear that something more capable would be helpful.

If you are writing your scripts in Java it is possible to be able to debug your scripts using an integrated development environment (IDE) that supports remote debugging. To do this you will first need a debug ROES Web Start server launch for your server (SoftWorks will have to create this launch for you). This launch is configured such that it will allow an IDE to connect to it debugging purposes. The send thing you will need is an IDE that supports remote debugging. we recommend using IntelliJ IDEA which you can get for free at http://www.jetbrains.com/idea/. Download and install the Community Edition which is free.

Once you have your IDE installed then you can create a new project in which you will develop your code that you wish to execute as a script. Before you start writing your code it will be helpful to download our ServerScriptStubs.jar (http://[tbd]/ServerScriptStubs.jar) and add it as a dependency in your project. The ServerScriptStubs.jar contains definitions of all the classes that the server makes public and available to scripts (see above). The stubs jar does not contain the implementation of these classes, merely definitions and empty methods so that your own code that utilizes these classes can compile correctly.

Now, create your classes and code that you want to run from a script in your project. Configure your project to produce a jar file. For example, let's say the jar produced is called ServerScriptTest.jar.

When you are ready to test your code, launch the ROES Server using your RWS debugging launch. Use your IDE to remotely debug the server. In your script in the server, add your jar to the class path. Now you can create instances of the classes you built into your jar in your script and call their methods. Your script would look something like this:

```
// Begin script

// Add our jar to the class path
addClassPath( "/Path/to/your/jar/ServerScriptTest.jar");

// import the package containing your class
import com.softworks.scripttest.*;

// Create an instance of my class
ServerScriptTest scriptTest = new ServerScriptTest();

// Call a method of the object, you will be able
// to set breakpoints in your code
scriptTest.doSomethingInteresting();

// End Script
```

In your IDE you can set break points in your code. Now, exercise the server to cause your script to be executed. When the script runs, the break points you've set in your code will stop execution and you will be able to do all the things that you would normally do when debugging code (examining and setting variable values, setting conditional break points ,etc.).