dp2 Order Importer Listener Agent

for the

ROES Server

1. Overview

The basic function of the dp2 Order Importer Listener Agent is to import an order that exists in dp2 into the ROES Server.

A Listener Agent you say?

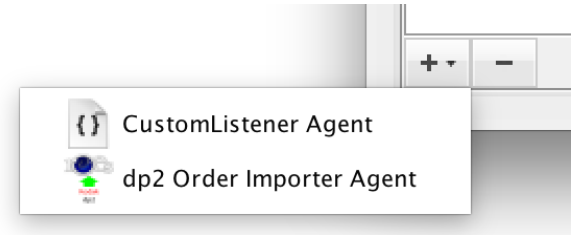
Yes, the dp2 Order Importer is a new type of listener. Up until now the only listeners that existed in the ROES Server were Custom Event Listeners and all other Agents were production agents that draw their work from a Queue. A Listener Agent is a listener in that it listens for a particular event and it's an Agent in that upon hearing that event, it goes and does some work.

In the case of the dp2 Order Importer Listener Agent, it is listening for an "ImportDP2Order" event and when it receives one, it will it will attempt to create a .ro file representing the identified order, currently in dp2, and will then place the created order file in the incoming order directory.

In creating the .ro file the dp2 Order Importer Listener Agent will create an order.xml file by going through all the dp2 jobs belonging to the order and parsing them and creating their equivalent as an item in the order xml.

Adding A dp2 Order Importer Listener Agent

To add a dp2 Order Importer Listener Agent go to the Listeners tab and click on the Plus button in the lower left corner of the window below the list of listeners. Then select the listener you want to add, in this case the dp2 Order Importer Agent



2. Getting Set Up

When you first create your dp2 Order Importer Listener Agent it will be assigned a unique name beginning with Undefined. You can set this name to anything you like.

Unlike the Agents in the Production tab, this Listener Agent is triggered by the broadcasting of an event. Since the event that it is listening for, "ImportDP2Order", is fixed, there is no field in the interface for modifying the event listened for.

2.1. Connecting to the dp2 Database

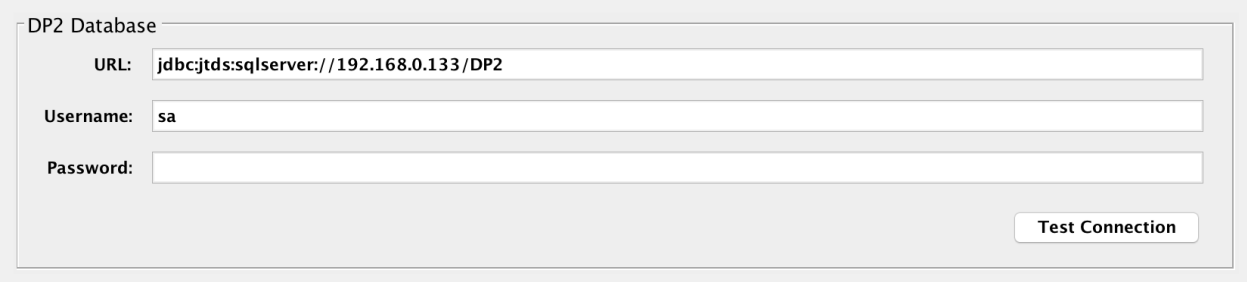
The dp2 Order Importer Listener Agent needs to connect to the dp2 Database in order to perform its operations. To do that we need to define a Java Database Connectivity (JDBC) url to the dp2 database. A JDBC url defines both a driver to use and the network connection path to connect to a database. In the case of dp2 almost universally Microsoft SQL Server (or some variant of MSSQL Server) is what is used for the database server. The ROES Server has a driver to connect to MSSQLS Server built in called jTDS. So your url would look something like this:

jdbc:jtds:sqlserver://<your MSSQL Server address here>/DP2

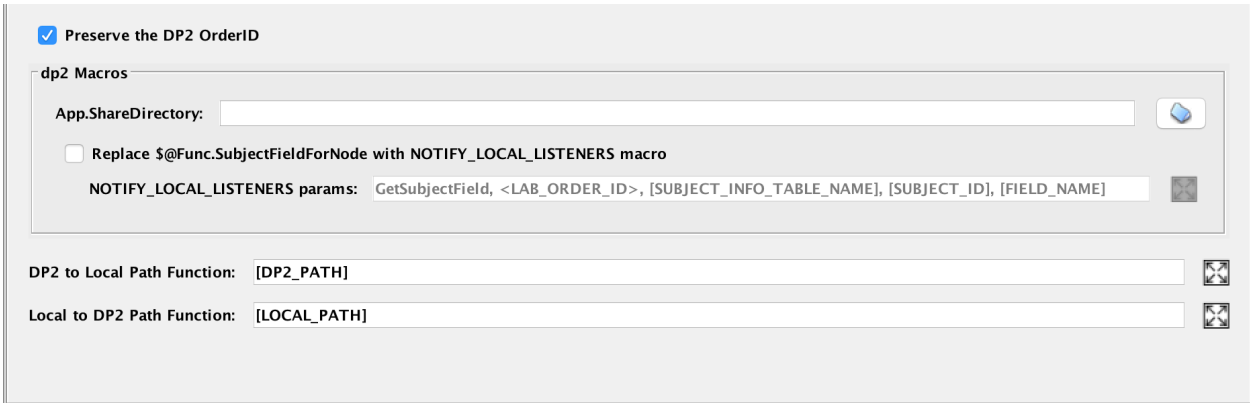
The Username and Password fields should be set to the username and password you have setup for accessing the DP2 database (by default dp2 sets the username is set to “sa” and the password to blank but your administer may well have set it to something different).

Once you have populated the database fields you can click on the “Test Connection” button to see if the setting are connecting properly and make adjustments as necessary.

When you're done it might look something like:



2.2. The Other Settings



Preserve the DP2 Order ID

If you select the "Preserve the DP2 Order ID" option then when the order file (.ro) is created, an additional file will be added that will cause the ROES Server to assign the same order ID that the order has assigned to it in dp2.

2.3. dp2 Macros

In creating the order .ro file, the dp2 Order Importer Listener Agent will have to go through and parse all the jobs belonging to the order being imported. In those jobs may be a number of various dp2 macros. Generally the dp2 Order Importer Listener Agent will evaluate those macros and place the value called for in it's own order data.

App.ShareDirectory

The jobs of the order might make use of the $@App.ShareDirectory directory so dp2 Order Importer Listener Agent need to be told where dp2's App.ShareDirectory is so it can resolve those dp2 macros.

Replace $@Func.SubjectFieldForNode with NOTIFY_LOCAL_LISTENERS

If the order uses subject data and you want to preserve the behavior where subject data is dynamically looked up at rendering time then this option allows you to do that. If you don't need to preserve that behavior then you can skip this option section.

If you are not familiar with the NOTIFY_LOCAL_LISTENERS macro then jump ahead and read the section titled "Getting to know the NOTIFY_LOCAL_LISTENERS report macro" and then come back and read the rest of this section.

By default the dp2 Order Importer Listener Agent will evaluate the $@Func.SubjectFieldForNode macros with the value that they call for. This will statically set the value in the generated order data for the order. If you select the Replace $@Func.SubjectFieldForNode with NOTIFY_LOCAL_LISTENERS option, what it will do instead, is mark that text node as needing to be evaluated at rendering time (by using a "textneedsevaluating" attribute) and setting the text to the macro string "[NOTIFY_LOCAL_LISTENERS(<param data here>)]". The data passed by the NOTIFY_LOCAL_LISTENERS macro is set in the NOTIFY_LOCAL_LISTENERS params field.

By default the NOTIFY_LOCAL_LISTENERS params field will be filled in with the value:

GetSubjectField, <LAB_ORDER_ID>, [SUBJECT_INFO_TABLE_NAME], [SUBJECT_ID], [FIELD_NAME]

This params string is itself evaluated at the time the order data is created.

The first parameter is the event that is broadcast, in this case a GetSubjectField event. The subsequent parameters are passed as the data for that event.

The first data field passed, in this case, is "<LAB_ORDER_ID>" which does not get changed to anything as the string "<LAB_ORDER_ID>" , unless it is passed as a parameter to another macro, is not a macro so it's treated as a literal string and evaluates to "<LAB_ORDER_ID>".

The remaining parameters take advantage of three additional macros that get created at the time the params string is evaluated:

SUBJECT_INFO_TABLE_NAME - The SubjectInfo table name called for in the dp2 macro.
SUBJECT_ID - The subject ID of the subject called for in the dp2 macro.
FIELD_NAME - The field name from the table called for in the dp2 macro.

So let's say, for example, we had a text field where the dp2 macro called for retrieving the FirstName field from the SubjectInfoSports table where the SubjectID was 0001. The evaluated params string would look like:

GetSubjectField, <LAB_ORDER_ID>, SubjectInfoSports, 0001, FirstName

So the resulting text value for the field would be:

[NOTIFY_LOCAL_LISTENERS(GetSubjectField, <LAB_ORDER_ID>, SubjectInfoSports, 0001, FirstName)]

When that product is rendered in the ROES Server it will broadcast that event and look for a response from an event handler that is listening for "GetSubjectField". That response will be swapped into the text for that node and will be rendered.

2.4. Path conversion Functions

Down near the bottom of the dp2 Agent are two fields marked "DP2 to Local Path Function" and "Local to DP2 Path Function".

These function make it possible for you to morph a path that is defined relative to one system into a path that resolves to the same file but relative to the other system.

When would this ever be useful? If you are running both the ROES Server and dp2 in windows then you probably wont ever need to do anything here; unless maybe you were using mounted directories and the disk were not mounted with the same ID on both systems. If you are running your ROES Server on Mac OS X and dp2 on Windows then the dp2 Agent will need to be able to transform paths from one system into paths on the other.

By default, the "DP2 to Local Path Function" just contains the macro [DP2_PATH] which evaluates to the path, relative to dp2, that you will want to change into a local, that is, ROES Server relative path. Below is an example of how I do this in my test environment.

```
[BEGIN_SCRIPT(BeanShell)]
    import com.softworks.server.scripting.*;
    import java.util.*;
    import com.softworks.server.proxy.*;

    // Defined by the environment for us
    // Hashtable gReportMacros;
    // Hashtable gSystemData;
    // StringWriter gOutputWriter;
```

```
    // Convert Windows path to OS X path
    String thePath = (String) gReportMacros.get( "DP2_PATH" );

    if (thePath.toUpperCase().
           startsWith( "\\\\Macbookpc\\KPDP2\\".toUpperCase() ))
    {
        thePath =
           thePath.substring( "\\\\Macbookpc\\KPDP2\\".length() );
        thePath =
           "/Volumes/C/Eastman Kodak/KPro Applications/KPDP2/" +
           thePath.replace( '\\', '/' );
    }
    else if (thePath.toUpperCase().startsWith( "C:\\".toUpperCase() ))
    {
        thePath = thePath.substring( "C:\\".length() );
        thePath = "/Volumes/C/" + thePath.replace( '\\', '/' );
    }

    gReportMacros.put( "THE_PATH", thePath );
[END_SCRIPT][\]
[THE_PATH]
```

Basically the script is looking for either a preceding “\\Macbookpc\KPDP2\” or “C:\\” which are the two ways that a path might be defined in my dp2 environment, and replacing those with “/Volumes/C/Eastman Kodak/KPro Applications/KPDP2/“ or “/Volumes/C/“ respectively and then swapping any remaining backslashes with forward slashes.

Similarly, the “Local to DP2 Path Function” is populated by default with just the macro [LOCAL_PATH] which will evaluate to the path to be modified as it’s defined relative to the ROES Server environment. Below is an example of how I change the path on my test system.

```
[BEGIN_SCRIPT(BeanShell)]
    import com.softworks.server.scripting.*;
    import java.util.*;
    import com.softworks.server.proxy.*;

    // Defined by the environment for us
    // Hashtable gReportMacros;
    // Hashtable gSystemData;
    // StringWriter gOutputWriter;

    // Convert OS X path to Windows path
    String thePath = (String) gReportMacros.get( "LOCAL_PATH" );

    if (thePath.toUpperCase().startsWith( "/Volumes/C/Eastman Kodak/
KPro Applications/KPDP2/".toUpperCase() ))
    {
        thePath =
```

```
        thePath.substring(
   "/Volumes/C/Eastman Kodak/KPro Applications/KPDP2/".length() );
      thePath =
        "\\\\Macbookpc\\KPDP2\\" + thePath.replace( '/', '\\' );
  }
  else if (thePath.toUpperCase().
              startsWith( "/Volumes/C/".toUpperCase() ))
  {
      thePath = thePath.substring( "/Volumes/C/".length() );
      thePath = "C:\\" + thePath.replace( '/', '\\' );
  }

  gReportMacros.put( "THE_PATH", thePath );
```

[END_SCRIPT][\]
[THE_PATH]

This script basically does the same thing as the prior script stripping the Mac OS X prefix and replacing it with the windows equivalent.

If you wan tot learn more about scripting then you should take a look at our scripting guide “Scripting for the ROES Server”.

3. Related Topics

3.1. Getting to know the NOTIFY_LOCAL_LISTENERS report macro

The NOTIFY_LOCAL_LISTENERS report macro allows you to broadcast an event to listeners on this workstation and by means of a convention, pass data to and receive results back from the event listener.

The NOTIFY_LOCAL_LISTENERS report macro that can be called like so:

[NOTIFY_LOCAL_LISTENERS(HelloFromReport, MyFirstParam, MySecondParam)]

The first parameter to the macro is the event that will be broadcast, "HelloFromReport" in this example and the following comma separated parameters will be passed to the caller in the eventData with keys that are equal to their parameter position. So in the above example the two params would appear in the event data with the keys "1" set to " MyFirstParam" and "2" set to " MySecondParam".

By convention, listeners that handle events broadcast from the NOTIFY_LOCAL_LISTENERS report macro, can return data to the caller which will be used as the evaluated value for the macro by adding a key "result" to the eventData and setting it to the value to be returned. The event listener must not be run on a separate thread or it will not be able to return its result. If an exception occurs in the handler you can return that to the notifier by adding an "Exception" key to the eventData and setting it to the Exception object thrown. For example:

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

Hashtable eventData = gReportMacros.get( "EventData" );

// Get the first param
String p1 = eventData.get( "1" );
String p2 = eventData.get( "2" );

try
{
     eventData.put( "result", "Hello to you too, the first param
passed was:" + p1 + " and  the second param passed was:" + p2 );
}
catch (Exception e)
{
     eventData.put( "Exception", e );
}
//-----------------------------------------------------
```

3.2. Implementing a Custom Script Listener to handle GetSubjectField

Now you will want to add a CustomListener Agent to deal with the GetSubjectField events. Again go to the Listeners Tab and select the Plus button in the lower left corner and this time select “Add CustomListener Agent”. Name the listener as you like and set the “Listen for events named” field to “GetSubjectField”. Make sure the “Run in separate thread” option is not selected and select BeanShell as the script language.

Let’s say that we have the order’s subject data in tables matching the dp2 SubjectInfo tables (perhaps we used the OVERRIDE_HANDLE_SUBJECT_TABLE…not familiar with that? See the section below titled Importing Subject Data in ROES Server)

and then enter something like the script below.

```
//-----------------------------------------------------
// Called with [NOTIFY_LOCAL_LISTENERS( GetSubjectField,
//                                      <LAB_ORDER_ID>,
//                                      [SUBJECT_INFO_TABLE_NAME],
//                                      [SUBJECT_ID],
//                                      [FIELD_NAME] )]
import com.softworks.server.scripting.*;
import java.util.*;
import java.sql.*;

String eventName = (String) gReportMacros.get( "EventName" );
Hashtable eventData = (Hashtable) gReportMacros.get( "EventData" );
```

```
// Get the parameter values that were passed
String orderID = eventData.get( "1" );
String tableName = eventData.get( "2" );
String subjectID = eventData.get( "3" );
String fieldName = eventData.get( "4" );

try
{
    result = "";
    Connection dbConnection = DriverManager.getConnection(
                                gSystemData.get( "DATABASE_URL" ),
                                gSystemData.get( "DATABASE_UN" ),
                                gSystemData.get( "DATABASE_PW" ) );

    Statement aStatement = dbConnection.createStatement();
    String query = "Select " + fieldName + " from " + tableName +
                     " where OrderID='" + orderID +
                     "' and SubjectID='" + subjectID + "'";
    gOutputWriter.write( "Executing query:" + query );
    ResultSet rs = aStatement.executeQuery( query );
    if ( rs.next() )
        result = rs.getString( 1 );
    rs.close();
    aStatement.close();
    dbConnection.close();

    gOutputWriter.write( "Returning result:" + result );
    eventData.put( "result", result);
}
catch ( Exception err )
{
    // OK, let's put the exception back in the eventData
    // to let the caller know we encountered an exception
    eventData.put( "Exception", err );
}
//-----------------------------------------------------
```

3.3. Importing Subject Data in ROES Server

You can easily import subject data from a ROES order into tables that you have defined in the ROES Server database by adding a CustomListener Script for OVERRIDE_HANDLE_SUBJECT_TABLE events.

When an order containing subject data is processed by the ROES Server an OVERRIDE_HANDLE_SUBJECT_TABLE event will be broadcast for each table in the subject data within that order. The event passes a reference to the recordsfile element from the order data which the script can use to import that data into a table; e.g

```
//-------------------------------------------------------------------
// OVERRIDE_HANDLE_SUBJECT_TABLE script
import com.softworks.server.scripting.*;

Hashtable eventData = gReportMacros.get( "EventData" );
RSXMLElement recordsfileData = eventData.get( "recordsfile" );

// Get the tables
ArrayList<RSXMLElement> tables = recordsfileData.getChildren(
                                                    "recordstable" );

for (RSXMLElement aTable : tables)
{
  String tableName = aTable.getAttribute( “fromtable”,
                                aTable.getAttribute( “tablelabel” );
  if ( tableName != null )
    RSScriptUtilities.populateTableWithRecordsTableData(
                                 tableName.toUpperCase(), aTable );
}
//-------------------------------------------------------------------
```

In the script above we are looking for the name of the table. If this order was imported from dp2 using the dp2 Order Importer Listener Agent then the name of the dp2 Subject Info table that the data came from will be assigned to an attribute of the table element called “fromtable”. If the order was created using the ROES Client Events module then the table that the lab has associated with that table will be assigned to an attribute of the table element called “tablelabel”. If neither of those attributes have been defined on the table element then the script will have to be modified to determine what table the data should be added to.