



AltioLive Developer Training

Version 5.4

7. Using JDBC, SOAP and Java Service Functions

Summary

This module covers Service Function interface types other than the HTTP Service Function used in the Stocks application.

Integra SP

88 Wood Street London
EC2V 7RS
United Kingdom

www.altio.com

tel: +44 (0) 20 8528 1045

Contents

Using JDBC Service Functions	2
Introduction to JDBC Service Functions	2
The currency table in the stocks_currency database	3
Using the Application Manager to Define a JDBC Service Function	3
Defining a GET service function	3
Defining the Datakeys	6
Create an Application View	6
Creating a window to display the data	7
Creating a NEW Service Function	7
Creating an UPDATE Service Function	9
Creating a MARK_AS_DELETED Service Function	10
Creating the SQL_GET_UNDELETED_CURRENCY Service Function	11
Creating the SQL_CURRENCIES datapool	13
Saving the Application configuration	14
Setting the SQL_CURRENCIES Datapool	14
Continue building the application	15
Keeping Altio Consistent with External Data	18
Using SOAP Service Functions	20
Creating a SOAP Service function from the WS Demo Wizard	20
Testing the created SOAP Service Function from the Application Manager	22
Testing the created SOAP Service Function from the Designer	23
Using Java Service Functions	24
Creating a new JAVA Service Function	24
Setting the new Java Service Function	25
Introduction to the settings	25
Practical Exercise: Setting the Java Service Function	25

Using JDBC Service Functions

A previous session introduced HTTP Service Functions. Now we will complete our exploration of the communication methods available for integration with Back end applications: the JDBC Service Functions for integration with an SQL Back end and the SOAP service Functions for Web Services.

Introduction to JDBC Service Functions

Please note: *You need to work with an SQL database to do this exercise.*

In this part we will create JDBC service functions to interact with an SQL database at the Back end.

The client interface for configuring a JDBC service is similar to that used for an HTTP service. Behind the interface however the interaction with the Presentation Server and Back end database is quite different.

Any data returned from an SQL query has to be converted to XML before processing by the Presentation Server. A template is defined in the service function to map SQL columns to XML attributes.

Please be aware that not all of the features of SQL databases can be invoked using JDBC and the AltioLive Service Request interface. However, more complex statements are possible than the examples that follow.

We will use the MySQL database and the Mark Matthews JDBC Driver for MySQL. Altio does not recommend any particular back-end database or JDBC driver; however these products are easily available from <http://dev.mysql.com/downloads/connector/j/5.0.html>. You should have enough experience with the database you are using to be able to integrate it with external connectors such as AltioLive. At the least, you will require a JDBC driver for your database; consult your database documentation for more information.

The JDBC driver will need to be deployed on the application server. For AltioLive Studio Edition, the MySQL driver jar(s) must be deployed in the **WEB-INF\lib** directory.

The currency table in the stocks_currency database

For this exercise we will work with a very simple database. You can follow this exercise by:

- Working with your own SQL database. In this case you will need to adapt the parameters given in the exercise to your own database.
- Creating a simple database in MySQL called **Test** with a **stocks_currency** table populated as follows:

CURRENCY_ID	DESCR	QTY	SUBNAME	TIMESTAMP	ABBR	AL_ACTION
1	Japanese Yen	100	Sen		YEN	ACTIVE
2	US Dollar	100	Cent		USD	ACTIVE
3	Euro	100	Cent		EUR	ACTIVE

- **CURRENCY_ID:** contains a unique number value that is generated by the database. Every time a row is inserted, an incremented number will automatically be put in this column. Most databases provide incrementing number columns. The column should be a primary key.
- **TIMESTAMP:** is a column of type 'timestamp' which is supported by most enterprise databases. Any changes that are made to a row will result in the timestamp entry being updated. The timestamp implementation is not consistent across databases. On MySQL it is a date column; on Microsoft SQL Server it is a binary number (which has to be converted to an integer for any select statement to be readable – this is not covered in the examples here). If your database does not provide timestamps, you can leave the column blank for the purposes of this example.
- **AL_ACTION** This will contain a character string. Its use is explained later. Change the default value to **ACTIVE**.

Using the Application Manager to Define a JDBC Service Function

In this part we will configure a JDBC service function to get data from the database. Then we will define a datapool to handle live updates. Finally we will create a datakey that will define an index reference required for data updates.

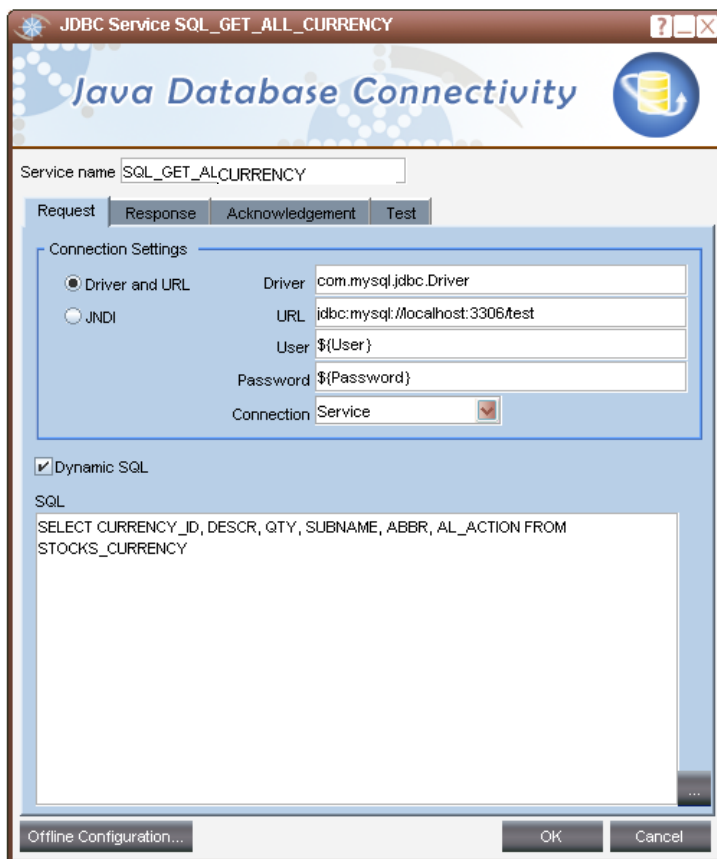
We will start by creating a JDBC Service function that will be used to get the Initial Data in the application.

Defining a GET service function

When creating a GET service function, it's important to have the correct SQL query. Unlike the HTTP service function we cannot test the function and datapool until we return to the Designer; so it is advisable to check the SQL using a utility associated with the database before we start. As an example, for MySQL we could use the `MySqlCommand` to test queries and check results before implementing the command in the service function. (Note: sample data and table creation files can be found in the training directory)

1. Before creating the Service Function, you need to copy your MySQL driver jar(s)(**mysql-connector-java-x.x.x-bin.jar**) in the **lib** folder under altio54\WEB-INF\lib.
2. Restart AltioLlve server.
3. Open the **Application Manager**, using the **STOCKS** application.
4. Press the **JDBC** button to create the new service function.
5. From the **Request** tab, set the parameters as follow:

Service name	SQL_GET_CURRENCY	
	REQUEST TAB	
Parameter	Value	Description
Radio Button: Driver and URL		
Driver	com.mysql.jdbc.Driver	In the driver's README file
URL	jdbc:mysql://localhost:3306/MySchema	Set the URL field to the address of the database. Include the schema name but not the database name.
User	Your Sql user, for example: root	You can also set the user as a parameter (by right-clicking on the Parameters folder in the Application Explorer) and then referring to it with the variable \${User}
Password	Your password	You can also set the password as a parameter (by right-clicking on the Parameters folder in the Application Explorer) and then referring to it with the variable \${Password}
Dynamic SQL	Tick this option	
SQL	SELECT DESCR, CURRENCY_ID, QTY, SUBNAME, ABBR, AL_ACTION FROM STOCKS_CURRENCY	Name of each element in the table (DESCR, CURRENCY_ID,...) and the name of the table (in this example: STOCKS_CURRENCY). Note that we exclude the TIMESTAMP element.



- From the **Response** tab, click on the **Generate Template** button. The template is used to format SQL data received into hierarchical XML format. This should return:


```
<SQL_GET_CURRENCY>
<stocks_currency DESCR="{response.DESCR}" CURRENCY_ID="{response.CURRENCY_ID}" QTY="{response.QTY}"
SUBNAME="{response.SUBNAME}" ABBR="{response.ABBR}" AL_ACTION="{response.AL_ACTION}"/>
</SQL_GET_CURRENCY>
```

- From the **Acknowledgement** tab, set the parameters as follow:

ACKNOWLEDGEMENT TAB		
Parameter	Value	Description
On success: DEFINE	Success	This is the message that will be send to the AltioLive client when the service succeed
On failure: DEFINE	Failure	This is the message that will be send to the AltioLive client when the service failed

- From the **Test** tab, click on the **Test Service** button. The **Test Results** window displays:



- Datakey warnings appear in the test output. Press the **Extract Datakeys** button to create the required keys automatically.
- A window displays the list of the created datakeys: **SQL_GET_CURRENCY** and **stocks_currency**. Click on the **OK** button.
- Save  the Application.

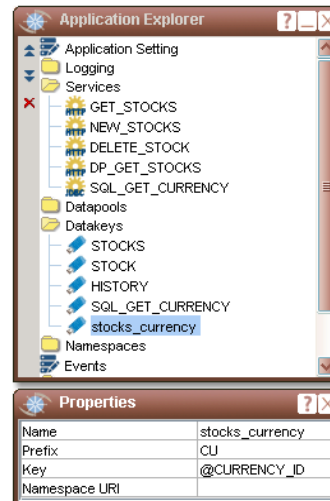
Defining the Datakeys

Datakeys provide unique identifiers to the Presentation Server for use in data updates with the Clients. For more information, see **Developers Training: 5. Getting Live Data using Datapools**.

Still from the **Application Manager**, we need to set the properties of the **stocks_currency** datakey as follows:


Property Value

Name	Stocks_currency
Prefix	CU
Key	@CURRENCY_ID




Create an Application View


Next we will create a view to use the JDBC functionality.

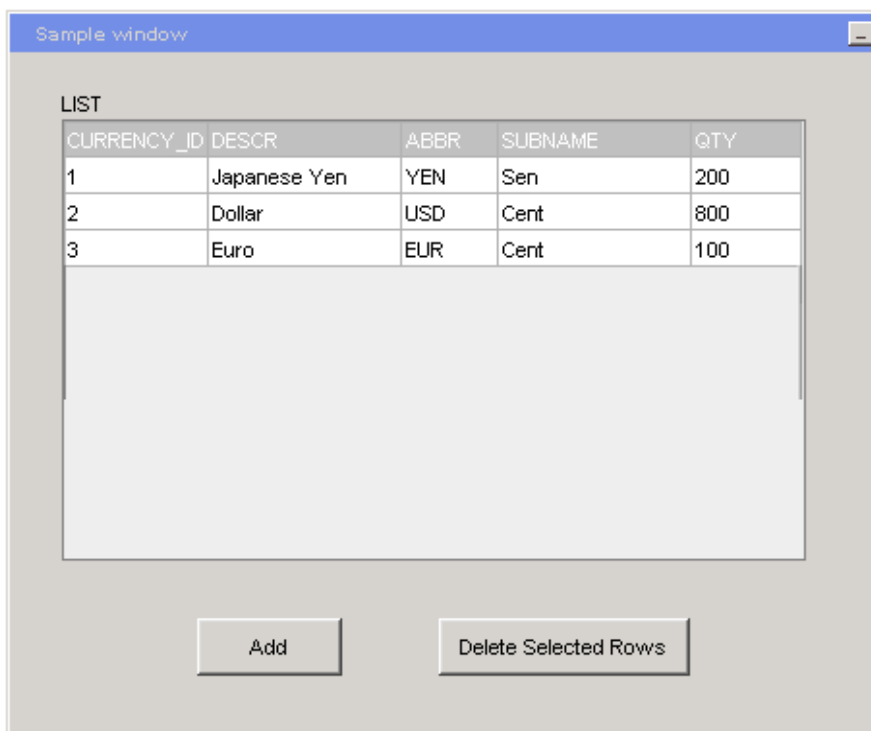
1. From the Altio console, load the Designer, using the application **Stocks**.
2. Create a new view from the **File | New View** menu item.
3. From the **View Explorer** window, right-click on the **Initial Data Service** folder and select **New Initial Data** in the context menu.
4. Select the new initial data and display its properties by clicking on the **Show Properties Window** icon .
5. Expand the **General** properties and from the **Server Command** drop-down menu, select **SQL_GET_CURRENCY**.

Please note: We will set the **Subscribe** property once we have created a Datapool. For now we are using the initial data only.

6. Click on the **Show Datatypes Window** icon.
7. Click on the **Import from Service...** icon  located on the left side of the **Datatypes** window. The **Import datatype information** window is then displayed.
8. From the **Service Function** drop-down menu select **SQL_GET_CURRENCY** and click on the **Import** button.
9. From the **Datatypes** window, expand the **DATA** node, it should now display the **SQL_GET_CURRENCY** element.
10. Save this view with the name **SQLCurrency**.

Creating a window to display the data

1. Still from the **SQLCurrency** view, create a window with a List control and two buttons as shown below.
2. To create easily the columns of the list, remember that you can display the **Datatypes** window and drag and drop attributes directly onto the list.
*For more information about linking datatypes and controls, see **Developer Training: 5. Getting Live Data using datapools.***
3. Set the Data source property of the list to: **SQL_GET_CURRENCY/stock_currency**.
4. Set the name of the first button to **LAUNCH_ADD_BUTTON** and its caption to **Add**.
5. Set the name of the second button to **DELETE_ROW_BUTTON** and its caption to **Delete Selected Row**.
6. Test the application by clicking on the **Run** icon .



7. Save the view .

Creating a NEW Service Function

At this point we have one JDBC service working to provide initial data to the application. Now we will complete the session by adding more functionality. First we need to return to the **Application Manager**.

We will create Service Functions to add, delete and update currencies. We will use a datapool to update the client data from the Back end database. This means that as changes occur on the Back end database, the Presentation Server will forward them automatically to the Client.

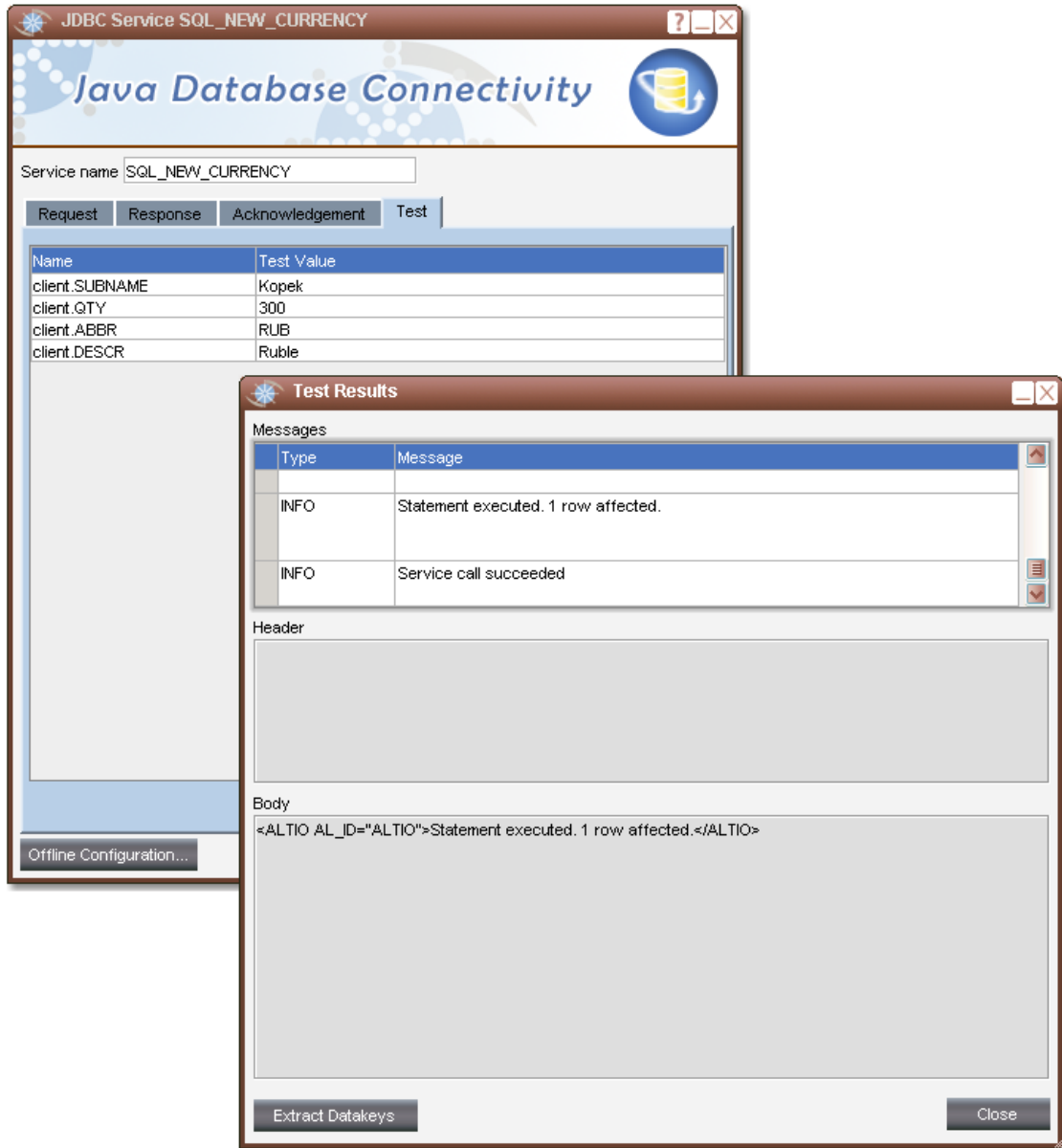
This will permit the user to add a new currency to the database.

1. Open the **Application Manager**, using the **STOCKS** application.
2. Clone the **SQL_GET_CURRENCY** Service Function and rename it to **SQL_NEW_CURRENCY**.

3. Set the parameters as follows:

Service name	SQL_NEW_CURRENCY
REQUEST TAB	
Parameter	Value
SQL	INSERT INTO STOCKS_CURRENCY (DESCR, QTY, SUBNAME, ABBR) VALUES ('\${client.DESCR}', `\${client.QTY}`, `\${client.SUBNAME}`, `\${client.ABBR}`)

4. Go to the **Test** tab, enter test values and click on the **Test Service** button.



Creating an UPDATE Service Function

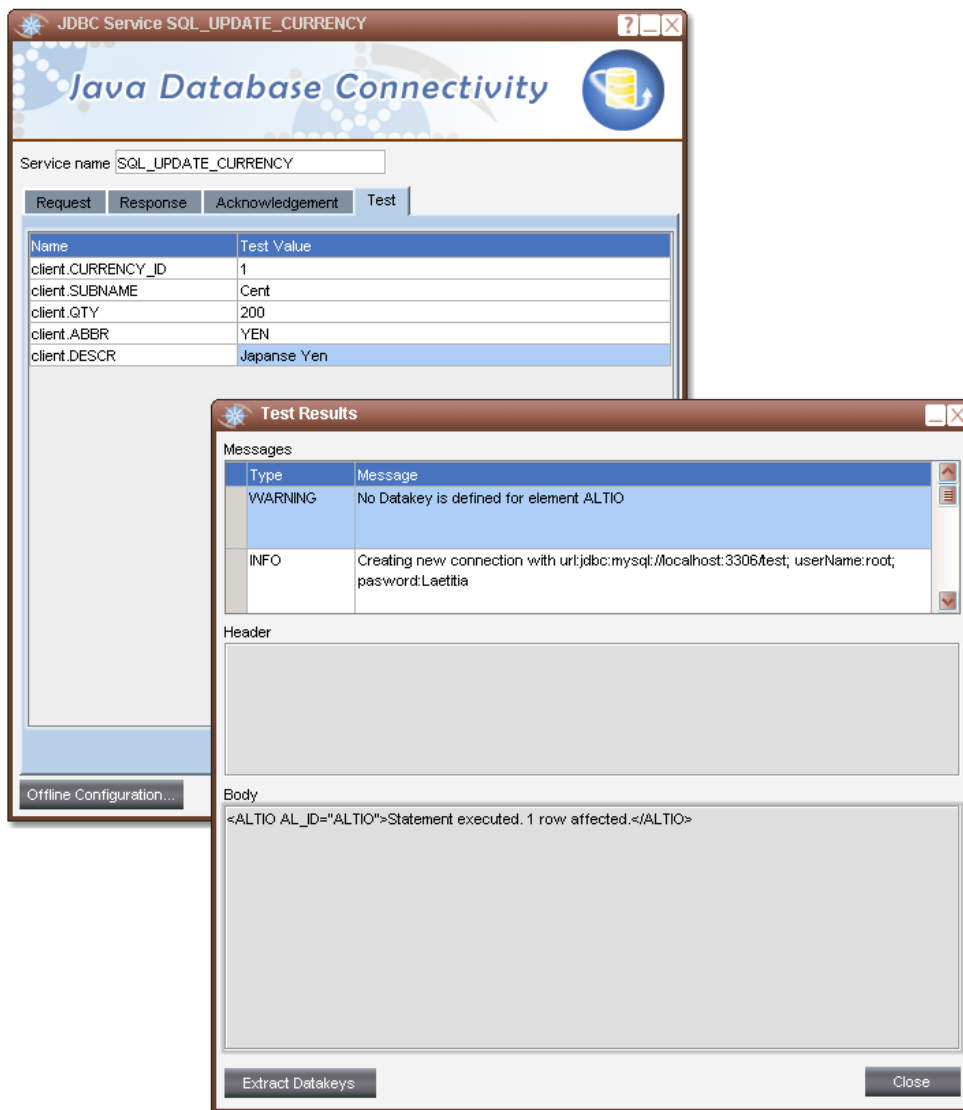
This will permit the user to make changes to the currencies data, which are then saved to the database.

1. Clone another JDBC service function from **SQL_GET_CURRENCY** and set the parameters as follows:

Service name	SQL_UPDATE_CURRENCY
REQUEST TAB	
Parameter	Value
SQL	UPDATE STOCKS_CURRENCY SET DESCR='\${client.DESCR}', QTY='\${client.QTY}', SUBNAME='\${client.SUBNAME}', ABBR='\${client.ABBR}' WHERE CURRENCY_ID='\${client.CURRENCY_ID}'

Please note: We do not need to go on the **Response** tab as no data is returned and no template is required.

2. Go to the **Test** tab, enter test values and click the **Test Service** button.

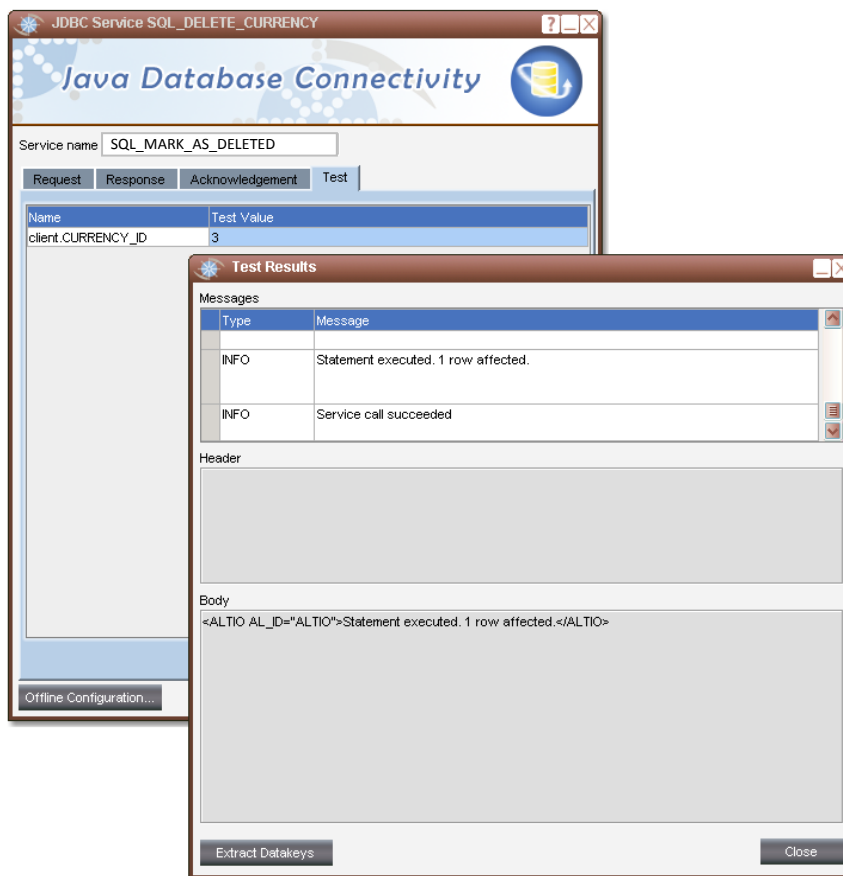


Creating a MARK_AS_DELETED Service Function

1. Open the **Application Manager**, using the **STOCKS** application.
2. Clone the **SQL_GET_CURRENCY** Service Function and rename it to **SQL_MARK_AS_DELETED**.
3. Modify the SQL statement as follows:

Service name	SQL_MARKS_AS_DELETED
REQUEST TAB	
Parameter	Value
SQL	UPDATE STOCKS_CURRENCY SET AL_ACTION='DELETE' WHERE CURRENCY_ID=\${client.CURRENCY_ID}

4. Go to the **Test** tab, enter a test value and click on the **Test Service** button.



Please note: This service function uses the attribute **AL_ACTION**. In the Client, data is marked as deleted using the attribute value **AL_ACTION = 'DELETE'**; Elements are not actually deleted, merely flagged as such:

Example of how the data is marked:

CURRENCY_ID	DESCR	QTY	SUBNAME	TIMESTAMP	ABBR	AL_ACTION
1	Japanese Yen	200	Sen	2009-02-18 10:...	YEN	ACTIVE
2	Dollar	800	Cent	2009-02-18 12:...	USD	ACTIVE
3	Euro	100	Cent	2009-02-17 12:...	EUR	DELETE
17	Ruble	300	Kopek	2009-02-18 12:...	RUB	ACTIVE

See the end of the session for an explanation of this delete implementation.

Creating the SQL_GET_UNDELETED_CURRENCY Service Function

1. Clone the **SQL_GET_CURRENCY** Service Function and rename it to **SQL_GET_UNDELETED_CURRENCY**.
2. Modify the SQL statement to include the timestamp and to return only undeleted currency.(the currencies that have an **AL_ACTION** attribute with the default value **ACTIVE**) :

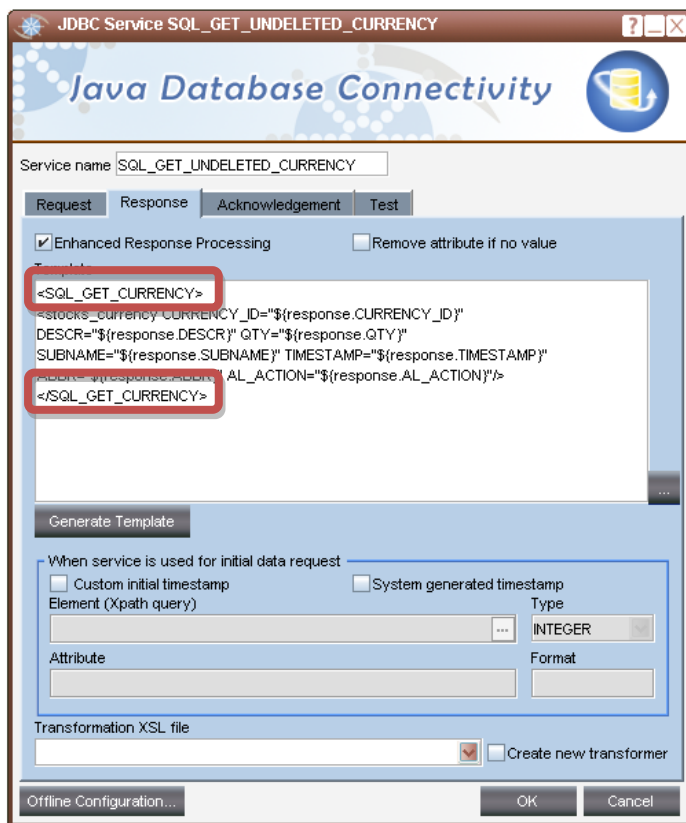
Service name	SQL_GET_UNDELETED_CURRENCY
REQUEST TAB	
Parameter	Value
SQL	SELECT CURRENCY_ID, DESCR, QTY, SUBNAME, TIMESTAMP, ABBR, AL_ACTION FROM STOCKS_CURRENCY WHERE AL_ACTION = 'ACTIVE'

Remember, each datapool has an associated timestamp. The timestamp is set initially when the first client subscribes to the datapool and receives initial data. Once the datapool is initialized, the timestamp changes whenever the Presentation Server receives updated data from the Back end application.

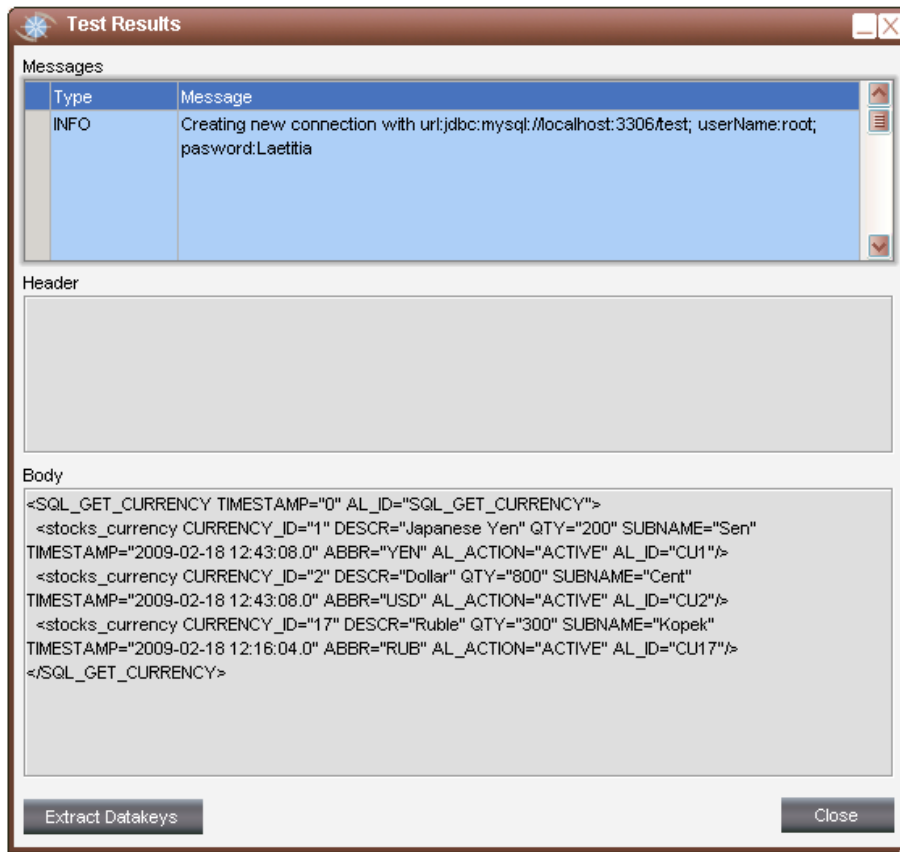
3. From the **Response** Tab, click on the **Generate Template** button. The template is used to format SQL data received into hierarchical XML format. This should return:

```
<SQL_GET_UNDELETED_CURRENCY>
<stocks_currency CURRENCY_ID="{response.CURRENCY_ID}" DESCR="{response.DESCR}" QTY="{response.QTY}"
SUBNAME="{response.SUBNAME}" ABBR="{response.ABBR}" AL_ACTION="{response.AL_ACTION}"/>
</SQL_GET_UNDELETED_CURRENCY>
```

4. Change the element **<SQL_GET_UNDELETED_CURRENCY>** to **<SQL_GET_CURRENCY>** as shown in the screenshot below:



- From the **Test** tab, click on the **Test Service** button. The Test results window displays:

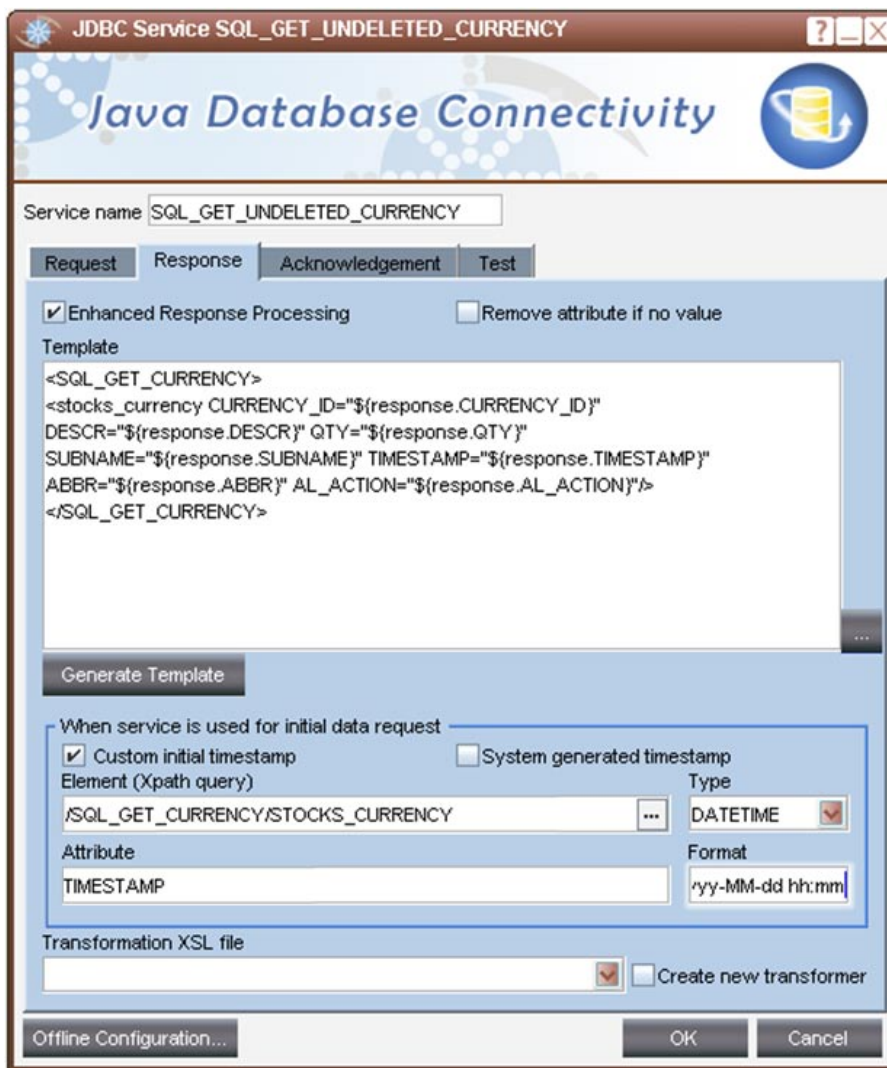


Please note: You can see that the **Euro** currency that we have marked as deleted is not included in the Results.

Creating the SQL_CURRENCIES datapool



1. Now create a new Datapool called **SQL_CURRENCIES**.
2. From the **Service** drop-down menu, select the **SQL_GET_UNDELETED_CURRENCY** service function you have created.
3. Set the **Pollrate** to **5** seconds. This controls how often the Presentation Server will interrogate the back-end database for changes to data.
4. Close the **Datapool** window.

Please note: Should you have problems with your polling service functions, you might need to specify the format of your timestamp coming from your database. This can be done in the response tab of both your initial service functions and your polling service functions as follows:



Saving the Application configuration


Now that we have made changes to the application configuration, we must save them.

1. Validate your changes by clicking on the **Validate Application** icon .
2. If the configuration is valid, save by clicking on the **Save** icon .

We have now finished the definition of the Service Requests, Datapool and Datakey required for the application.

Setting the SQL_CURRENCIES Datapool

First we need to use the Datapool:

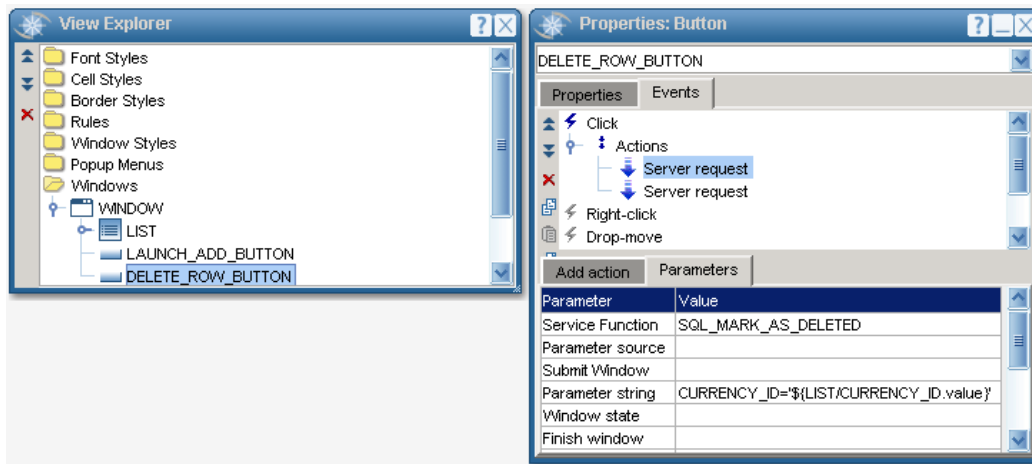
1. From the AltioLive Studio, expand the **STOCK** node and the **views** node under it.
2. Double-click on the **SQLCurrency.xml** view to launch the Designer.
3. On the **Initial Data Services** node, select the **SQL_GET_ALL_CURRENCY**, and display its properties by clicking on the **Show Properties** icon .
4. Change the **Server Command** property to the new Service Function: **SQL_UNDELETED_CURRENCY**.
5. Set the **Subscribe** property to the new Datapool **SQL_CURRENCIES**.

Continue building the application

We now need to continue building the application. Here are some guidance notes:

Please note: If you need more help, refer to the previous modules. You can also open the other view we created for the Stock application as the settings and parameters are similar.

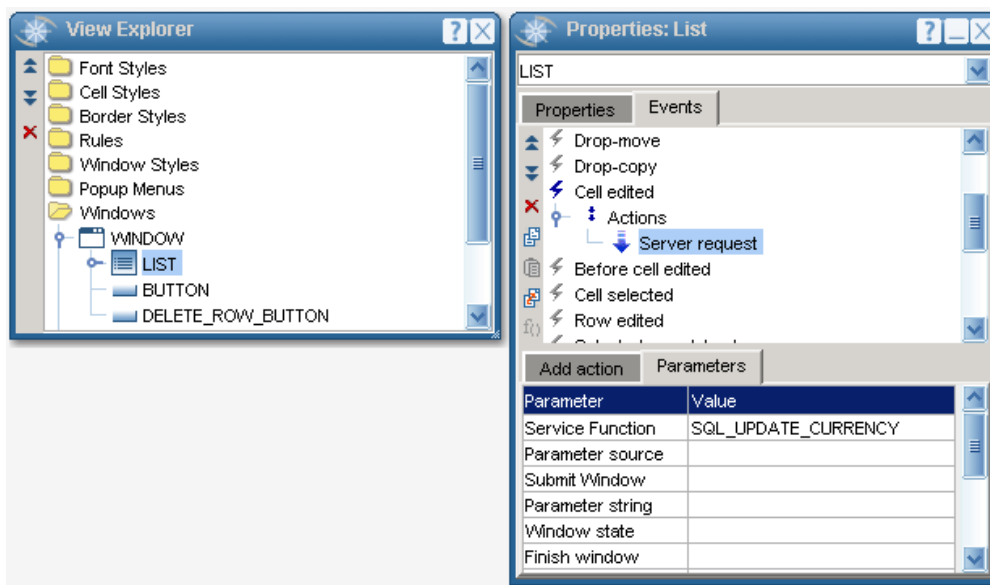
1. For the **Delete Selected Row** button, you should use a **Click** event which triggers a **Server Request** action that uses the **SQL_DELETE_CURRENCY** service function. This Service function needs a parameter: you have to specify which **CURRENCY_ID** is deleted by the Service Function. In our case, it is the one selected in the List control. The way to specify it in Altio is: **CURRENCY_ID='\${LIST/CURRENCY_ID.value}'**.



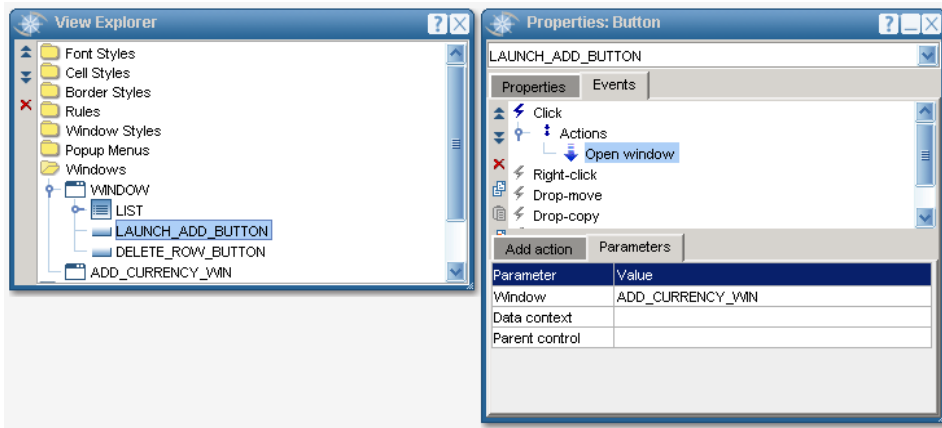
2. Still for the **Click** event, add another **Server Request** to call the **SQL_GET_CURRENCY** Service Function. This will update the List that will display only the elements that are not marked as deleted.

For more information on this delete implementation, see the end of the session.

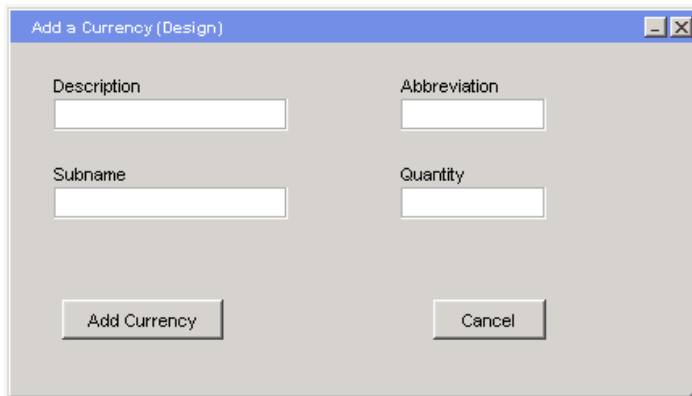
3. Modify the List so that cells can be edited. For this, you will need to set the **Editable** property of the list to **Y** and set the list columns to have their **Type** property set to **TEXT**. The **Type** property can be found under the group of properties called **Editing**.
4. For the list, use a **Cell edited** event which triggers a **Server Request** action that uses the **SQL_UPDATE_CURRENCY** service function.



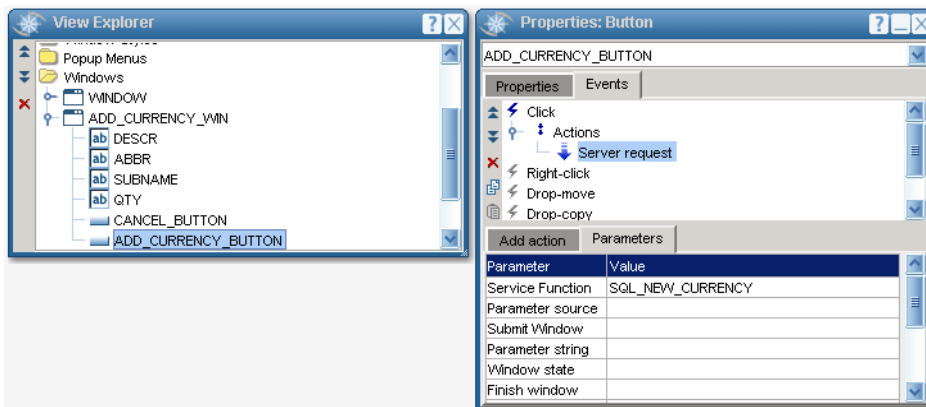
5. Create a new window called **ADD_CURRENCY_WIN**. Set its caption to **Add a Currency** and set its **Show on Startup** property to **N**.
6. Select the **LAUNCH_ADD_BUTTON** on the first window and use a **Click** event which triggers an **Open window** action set to the new **ADD_CURRENCY_WIN** window.



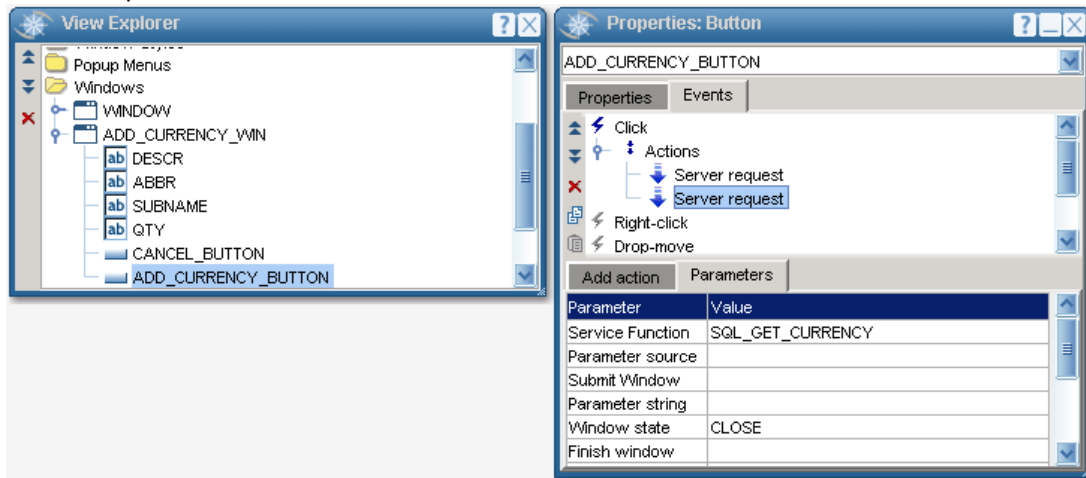
7. Add four text controls and two buttons on the **ADD_CURRENCY_WIN** window (see the screenshot below).
8. Rename the text controls to **DESCR**, **ABBR**, **SUBNAME**, and **QTY** to match the name of the Datatypes.
9. Change the **Caption** of the two buttons to **Add Currency** and **Cancel**.
10. Reposition the controls and set the captions so that the window looks like the window shown below.





11. On the **Add Currency** button, use a **Click** event which triggers a **Server Request** action that uses the **SQL_NEW_CURRENCY** service function.



- Still on the **Click** event add another **Server Request** that calls the **SQL_GET_CURRENCY** Service Function. This will update the List that will display the new currency. For the **Window state** property, select **CLOSE** from the drop-down menu. This setting closes the window, once the Server Request is called.

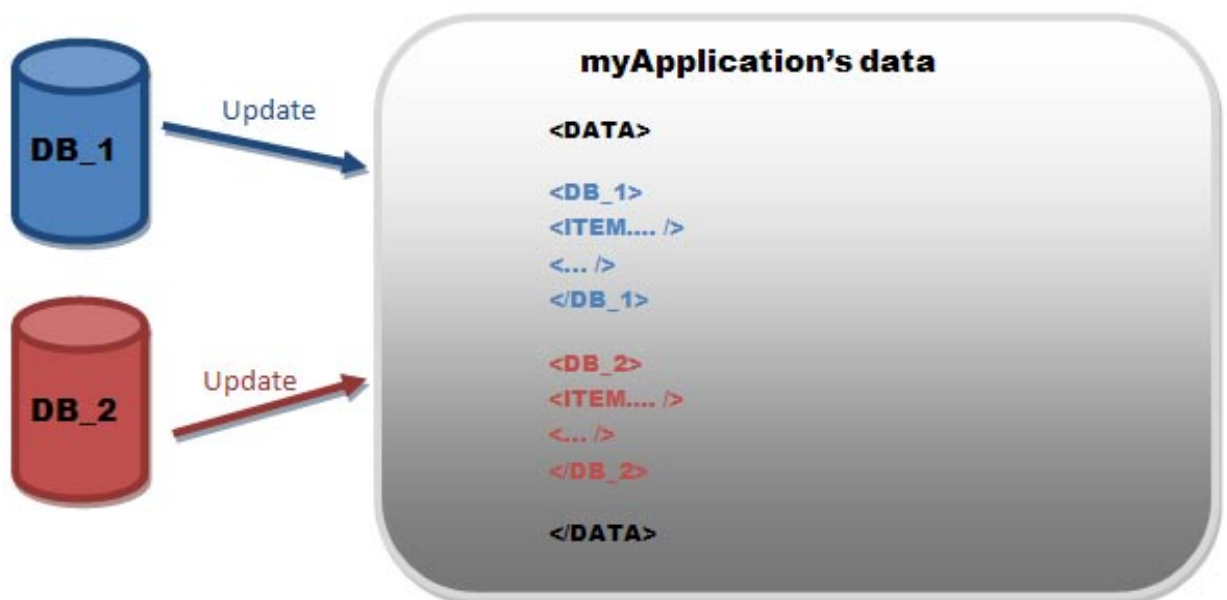


- For each text controls, set the **User input required** property to **Y**.
- For the **Cancel** button use a **Click** event which triggers a **Close window** action.
- Save  and click on the **Run icon**  to test the View. Try to edit, add and delete currencies.
- Try modifying a currency on the SQL database (for example, change the **DESCR** value) to check that the datapool is getting new data from the Back end.

Keeping Altio Consistent with External Data

We should discuss how to keep consistency between Altio and an external data source. Altio stores the data returned to it from service functions and treats this data as an update, not a wholesale replacement for the existing data. This means that if a row is removed from a database and Altio has a service function that returns all the rows, Altio will add any new rows to its copy of the data and update changes to existing ones but will not remove ones that the database no longer has. Why does Altio do this? Because if it didn't Altio would end up removing perfectly good data from different service functions whenever an update was received.

For example, if the application called **myApplication** uses two databases: **DB_1** and **DB_2**. Altio can manage the two data sources:



If Altio deleted elements that are not in the data sent by the Updating Service Function then when **DB_2** sends updates to Altio, the elements belonging to **DB_1** would be deleted, and vice versa, when **DB_1** sends its updates, **DB_2**'s elements would be deleted. To avoid that, Altio only creates new elements and updates existing ones but does not delete them.

The way to tell Altio to remove elements is to use a special attribute called **AL_ACTION**, when this attribute is set to **DELETE** then Altio automatically deletes that element if it already exists in data.

That's why we use a service function called **SQL_MARK_AS_DELETED**, this service function sets the **AL_ACTION** attribute to **DELETE** then we use a second Service Function called **GET_CURRENCY** to reload the data.

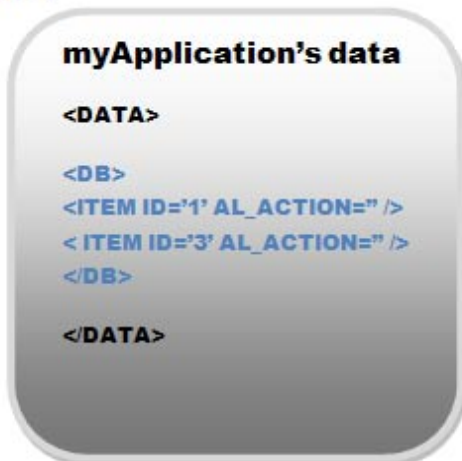
STEP 1



STEP 2



STEP 3



Using SOAP Service Functions

AltioLive 5.4 provides access to Remote Procedure Calls (RPC) using standard SOAP (Simple Object Access Protocol) encoding via HTTP data transfer.

The RPC-based web service must use a Web Service Definition Language file that includes a SOAP binding for the service definition to be invoked. WSDL is an XML-based language used to define Web services and describes how to access them.

The arguments for a SOAP RPC call are limited to simple types – arrays and other complex types are not supported. The return types from a SOAP RPC call are limited to simple types and arrays.

Creating a SOAP Service function from the WS Demo Wizard.

AltioLive Application Manager provides the WS Demo Wizard to help with building SOAP Service Requests. Click on **File | WS Demo Wizard** menu item to access it:

The screenshot shows the 'Web Services Demo Wizard' window. It has a title bar with a star icon and the text 'Web Services Demo Wizard'. The window is divided into several sections:

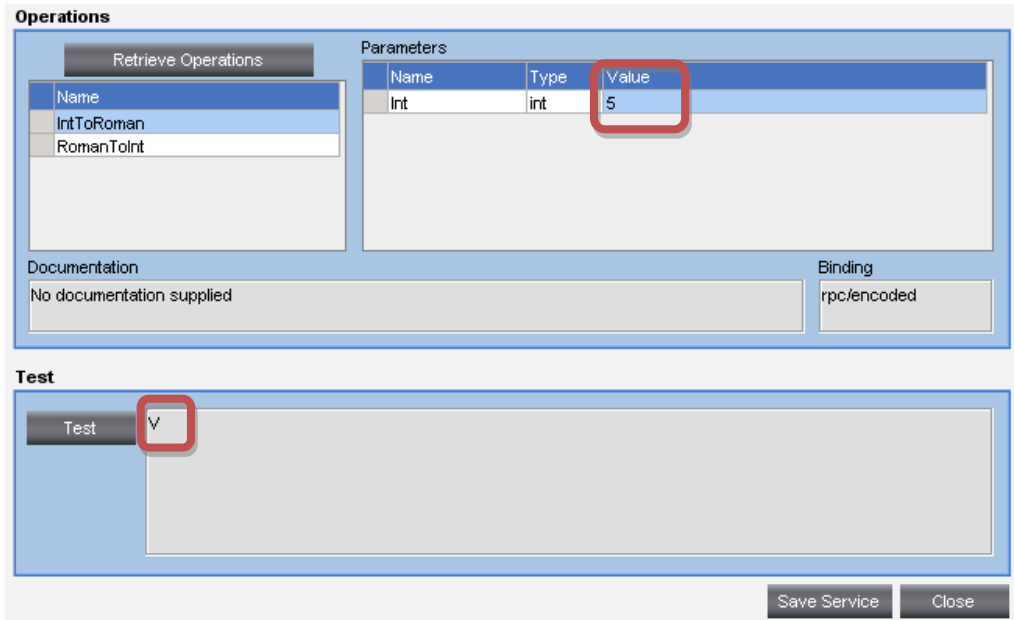
- WSDL Source:** Contains a table with two columns: 'Examples' and 'URL (and args)'. The first row has 'Euro Conversion' in the 'Examples' column and 'http://www.drBob42.co.uk/cgi-bin/Euro42/wsdl/Euro' in the 'URL (and args)' column. Below the table is a text area containing the text: 'Use currencies that are now merged with the Euro, e.g. DEM, NLG, ESP'.
- Operations:** Contains a 'Retrieve Operations' button. Below it is a table with a 'Name' column. To the right is a 'Parameters' table with columns 'Name', 'Type', and 'Value'. Below these are 'Documentation' and 'Binding' text areas.
- Test:** Contains a 'Test' button and a large empty text area.

At the bottom right of the window are two buttons: 'Save Service' and 'Close'.

Some examples are provided, but you can also enter any URL and arguments.

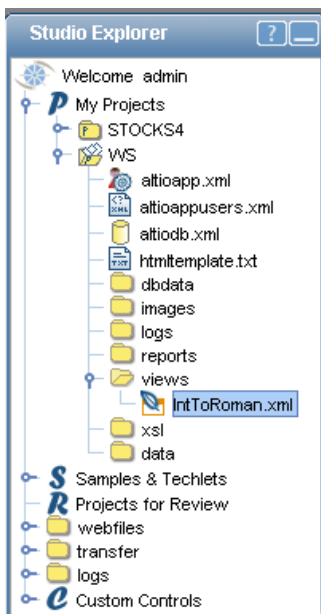
1. From the **Examples** drop-down menu, select **Roman Numeral Conversion**.
2. Press the **Retrieve Operations** button.
3. Select the **IntToRoman** operation and enter **5** in the Value field.
4. Click on the **Test** button.

The result of the conversion displays:



5. Click on **Save Service** button.
6. Close the Application Manager.

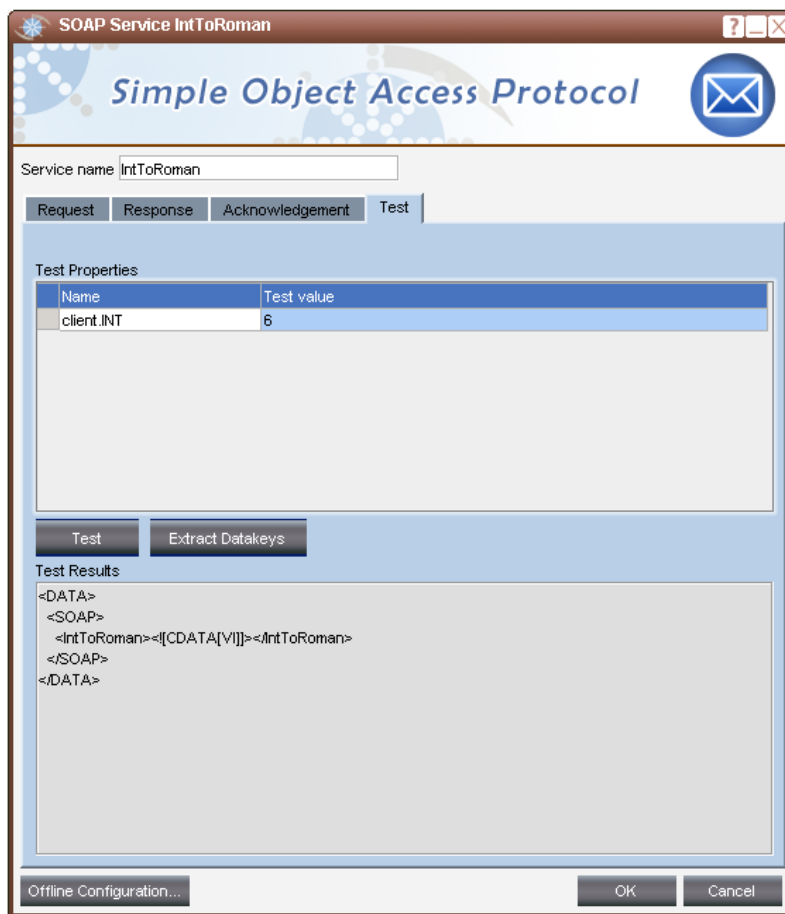
From AltioLive Studio you can see that a SOAP Service Function and a View are created in the **WS** application folder:



Testing the created SOAP Service Function from the Application Manager

1. Launch the Application Manager for the **WS** application.
2. Expand the **Services** folder from the **Application Explorer** and double-click on the **IntToRoman** service function.
3. Examine the four tabs: **Request**, **Response**, **Acknowledgement**, and **Test**.
4. From the **Test** tab enter **6** in the **Test value** field.
5. Click on the **Test** button, the **Test Results** field should display:

```
<DATA>
<SOAP>
<IntToRoman><![CDATA[VI]]></IntToRoman>
</SOAP>
</DATA>
```



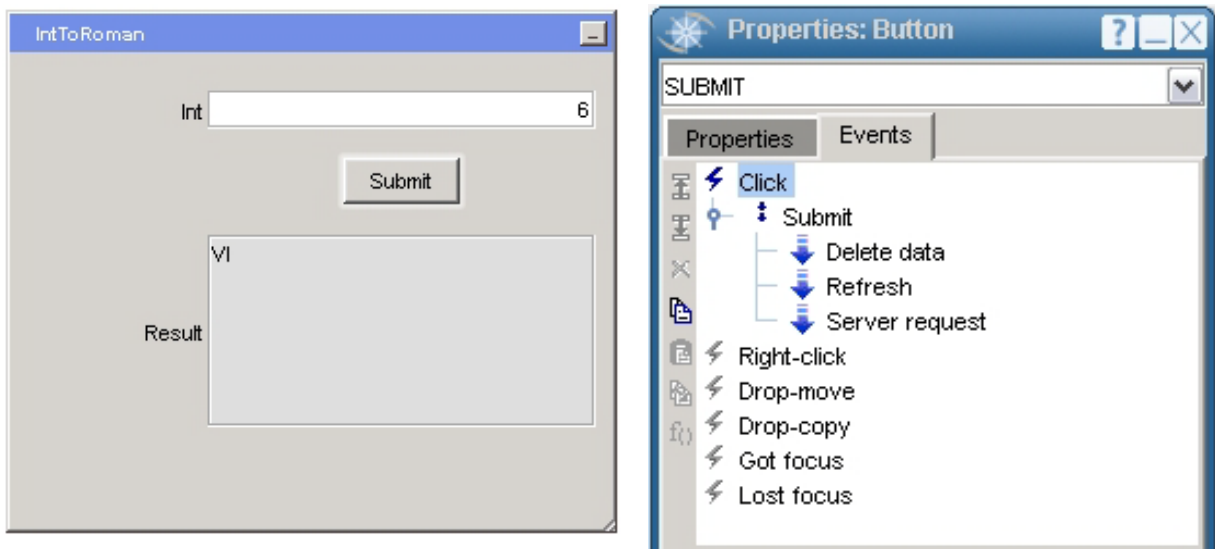
Testing the created SOAP Service Function from the Designer

1. From the AltioLive Studio, expand the **WS** node and double-click on the **IntToRoman** view to launch the Designer.

There is one window called **IntToRoman**, with controls for each of the parameters:

- an editable text control called **Int**,
- a **Submit** button,
- a text control for the result.

The **Submit** button has a **Click** action which deletes existing data, refreshes the text box then calls the Soap Service Request.




Using Java Service Functions

A Java Request provides the ability to call a method within a class available to AltioLive Presentation server and remote methods using **RMI**. For a method to be called locally the class must be available in the class path either **WEB-INF/lib** or **WEB-INF/classes**.

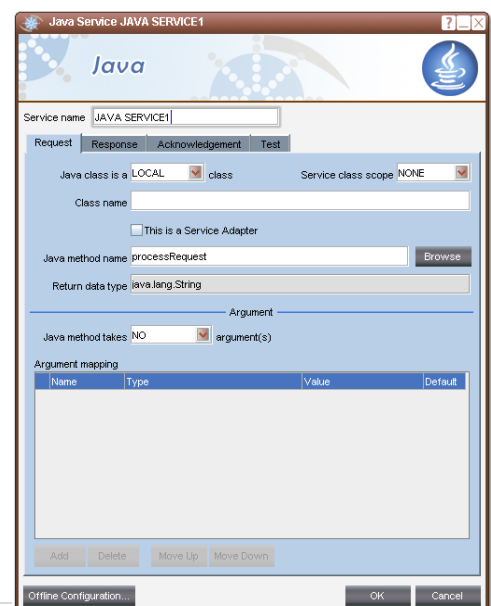
When looking for or developing a Java class to be used for your Java Request, you should consider the following limitation.

1. You can call any Java method which expects any number of arguments, but these arguments must be either Java primitive types or Objects with a constructor which takes a String.
2. When selecting a Java method for the **Java Request**, the specified Java class must have a default constructor (i.e. takes no argument), except that the Java method is a “static” method.
3. Any return data is supported. Presentation server converts return data to a String, because **toString()** is defined in Object class, any Java Object can be converted to String. The only question is whether the String is the right representation of the return data. If you need to return data back to the AltioLive client, make sure you format the data as valid XML. When processing the return data, Presentation server tries its best to convert return data to a String. By default it calls **toString()** to convert the data, but for primitive types, (Array, java.util.Map, java.lang.Collection, org.w3c.dom.Document and org.w3c.dom.Element, etc,) Presentation server uses internal utility functions to convert the data. If the return data is a remote object, you also need to define a remote method to convert the data.

Creating a new JAVA Service Function

1. From the Application Manager either:
 - Click on the **New JAVA Service** icon  on the toolbar.
 - Select the **Services** node in the **Application Explorer** and click on the **Edit | New JAVA service** menu item.
 - Right-click on the **Services** node in the **Application Explorer** and select **New JAVA service**.

A Service Function is added to the list of Services with the default name of **JAVA_SERVICEn**. The following Service details window will open:



Setting the new Java Service Function

Introduction to the settings

- On the **Request** tab specify the type of the Java class: **LOCAL** or **REMOTE**.
 - LOCAL** class: specify a Java class which exists in the same JVM with Presentation Server. (for example: **com.altio.demo.services.DateService**).
 - REMOTE** class: specify the reference of the remote class (for example: **rmi://localhost:1099/RmiDateServer**).
- Specify your **Java method name** defined in your Java class.
- If the Java method takes argument, you also need to set the argument type: (**NO** | **OBJECT** | **HASHTABLE**) and add all required arguments.
- Go to the **Test** tab. If the Java method takes argument from client, you will be able to specify test value for these arguments. Once you set test value, click on the **Test Service** button to test the Java method. You can then see the Output data (data that will be sent to AltioLive client) and Original data (data that is returned from Java method).

If there is a problem with your setup, appropriate messages will be displayed to help you correct the problem before you can test the Java service. Once the Java service works and returns data, you can then fine tune the return XML data structure from the "Response" tab.

Please note: To use RMI you must install a security manager within the Altio JVM.

Practical Exercise: Setting the Java Service Function

In this exercise you will create a call to a local java class which will return a formatted date.

- On the **Request** tab set the following parameters:

Service name	NEW_JAVA	
REQUEST TAB		Description
Parameter	Value	
Java Class is a	LOCAL	
Class name	com.altio.demo.services.DateService	<i>As this is a local class it must be available within the JVM AltioLive Presentation Server is running in.</i>
Java method name	getFormattedDate	You can select Browse to see the list of methods available in the class
Java method takes	OBJECT	

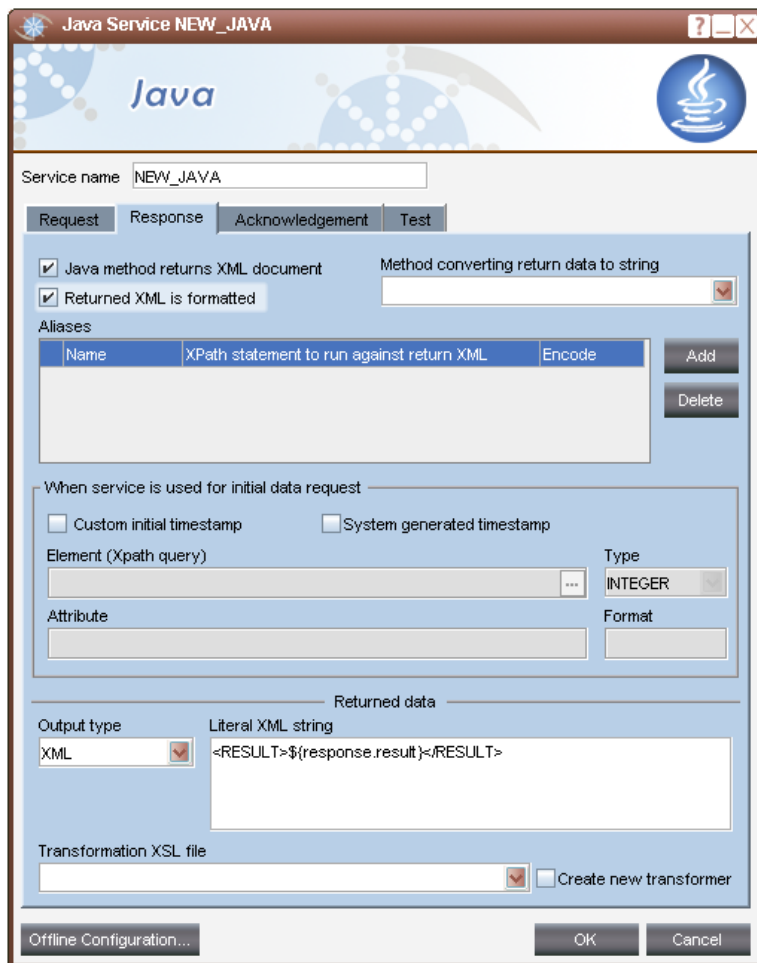
- Enter the following for the **Argument mapping**:

Name	Type	Value
PARAM1	java.lang.String	\${client.PARAM1}

Your screen should look like the one below.



3. Select the **Response** tab and make the following changes:



4. Go to the **Test** tab.

5. Enter a value for the date format like this: **yyyy-MM-dd** (for example: 2008-08-19)
6. Click on the **Test Service** button.



The **Output Data** tab shows the XML data that will be sent back to AltioLive client. You can also click **Original Data** tab to see the original data returned from the Java method.

Next Step: Using the Prototyping Wizard

AltioLive Developer Training

Document Information

KEYWORDS: Java, SOAP, JDBC Service Functions, SQL.

Integra SP – Altio

Telephone: +44 (0) 20 8528 1045

Internet: www.altio.com

Copyright © 2010 Integra SP

Copyright in this document is vested in Integra SP. The contents of the document (wholly or in part) must not be reproduced, distributed, used or disclosed without the prior written permission of Integra SP.

Integra recognizes the trademarks or registered trademarks of any third party product or company name referenced in this document at the time of its publication.