precisely

# Assure DQ

## Application Programming Interface Guide

# Notices

precisely

Trademarks

Infogix, the Infogix logo, ACR, ACR/Detail, ACR/Summary, ACR/Workbench, ACR/Connector, Infogix Assure, Infogix Insight, ACR/Instream, ACR/File, Infogix ER, Infogix Perceive, Data3Sixty, and Data360 are registered trademarks of Precisely. Data3Sixty Analyze, Data3Sixty Govern, Data3Sixty DQ+, Data360 Analyze, Data360 Govern and Data360 DQ+ are trademarks of Precisely. Any other trademarks or registered trademarks are the property of their respective owners.

1700 District Ave Ste 300
Burlington MA 01803-5231
USA

www.precisely.com

# Contents

## Chapter 5 SOAP Web Service for Control Point Execution



## Chapter 6 SOAP Web Service for Control Point Results



## Chapter 7 REST Web Service for Control Point Execution

## Contents

# 1

# Introduction

Assure DQ control processing can be triggered from the user interface, from the command line where the server or a client is deployed, or by API request. This guide describes how to pass API requests for control processing to Assure DQ. A Web service API and message-oriented API are both supported. Data quality control point processing is not supported.

This guide is for application programmers responsible for developing client programs that pass data to the Assure DQ server, trigger the Assure DQ server to execute control point processing, and (optionally) get back processing results. The guide explains how to form the request and parse the reply using either the Web service or message-oriented APIs.

The *API Guide and Examples* CD includes sample Java™ and COBOL programs demonstrating the message-oriented API, and a Java client program demonstrating use of the Web service.

The guide has the following chapters:

- Chapter 1, "Introduction"—Assure DQ API messaging. supported Web services, and Assure DQ control points supported.
- Chapter 2, "COBOL Messaging Sample"—how to structure and send messages from COBOL-based applications to Assure DQ.
- Chapter 3, "Java Messaging Sample"—JMS messaging from Java-based applications.
- Chapter 4, "API Message Dispatcher"—how to dispatch JMS messages from a queue of your choice directly to the Assure DQ API queue.
- Chapter 5, "SOAP Web Service for Control Point Execution"—explains the Web service for executing control points and how to run the sample Java client.
- Chapter 6, "SOAP Web Service for Control Point Results"—describes the Web service for requesting processing results.
- Chapter 7, "REST Web Service for Control Point Execution"—explains the REST Web service for executing control points and how to run the sample Java client.
- Chapter 8, "REST Web Service for Control Point Results"—explains the REST Web service for requesting processing results.
- Appendix A, "Assure DQ Message Schema"—provides the XML schema.

# Types of Messaging

Two types of messaging are supported by Assure DQ: wait-for-response and fire-and-forget. Wait-for-response messaging is synchronous. An application program sends messages containing data to Assure DQ and waits for corresponding responses indicating processing results. Fire-and-forget messaging is asynchronous. An application program sends messages to Assure DQ but does not expect any response.

# Message-Oriented API

Assure DQ supports both JMS and native messaging operations that can be integrated into a J2EE JMS environment, such as WebSphereMQ. In JMS, an XML (Text) message is supported. In WebSphereMQ native messaging, a MQSTR message is supported. Both types of messages are in XML. See the appendix for the message structure.

Because the message-oriented API capability uses message queues, both synchronous and asynchronous messaging are fully supported. The potential advantage of asynchronous messaging is that if a response to a request is not needed, the application program need not wait for it before sending another request.

To enable the application program to match responses with earlier requests, the Assure DQ server provides correlation IDs supported by both JMS and native WebSphere MQ. If an application sends a correlation ID, Assure DQ will respond with the same correlation ID. If the application program does not supply a correlation ID, Assure DQ copies the message ID of the request to the correlation ID of the response. This enables the application program to use the request message ID as the correlation ID.

When an application requests a response, Assure DQ sends back the request's original <trec> or <brec> record content as part of the response. Requests can also include a callback data element <cb> for Assure DQ to return with the response. The application program can use this element to pass application-specific data that does not require integrity checking. This enables the application to asynchronously process responses without having to internally retain the callback data in the intervening time.

Optionally, the system can populate a runtime field with the standard JMS message timestamp. To enable this feature, configure the JMS_TIMESTAMP_RUNTIME_FIELD property in the api.properties file.

Specify a name of your choice for the field. The name cannot include spaces. The api.properties file is in *install_folder*\Infogix\igx-data\default \IA\config.

To use the timestamp in control processing or include it in results and reports, set up a date type runtime field at the control point level. Use the same field name specified in the api.properties file. See the online Help in the Assure DQ user interface for details on how to define and use runtime fields.

See Chapter 2, "COBOL Messaging Sample" and Chapter 3, "Java Messaging Sample" for more information and sample programs demonstrating the capabilities of the message-oriented API.

# Web Service Support

The control point execution Web Service provides simple messaging capabilities. A client program can pass data to the Assure DQ server, request control point processing, and (optionally) wait for a response indicating processing results. The client program can point to a file to process or pass individual transactions to Assure DQ. The client can also simply request processing of a control point and get back results.

The results Web Service returns processing results for any previous control point run. This can include summary-level information or detailed results that the client program can page through.

To use the Web Services, add code to an existing application or create a new client program to call the Web Services. The Web Services support both Java and .NET clients.

# Types of Control Points Supported

The control point execution API supports all types of data capture control points: Balance, Single Capture, and Multiple Capture. The system can capture and process either a single instance or multiple instances of data from a single API call.

The Web Service and MQ and JMS messaging all support passing single or multiple instances of API text or binary data. Recap Reports and control data export are not available during API invocation of Multiple Capture control points.

The Web Service API also supports direct input of control field name/value pairs. This style of API call must pass only a single instance of data, so only Balance and Single Capture control points are supported.

# Optional Result Catalogs

By default, when an API request invokes Assure DQ, no result catalogs are produced. A rules writer can override this behavior if users want result catalogs. To get result catalogs, set the Generate Report option for a pass/fail/complete validation outcome. When the validation outcome occurs, the control point will produce a result catalog.

# 2

# COBOL Messaging Sample

This chapter describes two sample COBOL programs for sending and receiving messages from Assure DQ on z/OS. The *API Guide and Examples* CD includes these programs and supporting copybooks, JCL, the XML message schema, and a control entity zip file containing sample rules.

IASAMP01 demonstrates wait-for-response messaging. This is a synchronous style of messaging where an application program sends messages containing data to Assure DQ and waits for corresponding responses indicating processing results.

IASAMP02 demonstrates fire-and-forget messaging. This is an asynchronous style of messaging where an application program sends messages to Assure DQ but does not expect any response. The sample also shows how to convert and send binary application data via the messaging API.

## Sample Program 1

IASAMP01 is a sample application program that sends messages to Assure DQ and processes the results sent back by the server. The data sent by the sample program is a <trec> all text record containing a single instance of data for Assure DQ to process. The sample program parses the response from the Assure DQ server and displays the return code or error messages that were returned.

See Chapter 3, “Java Messaging Sample” for an example of a multiple instance API call. COBOL/MQ messaging supports both single and multiple instance calls.

The sample demonstrates how the message ID of the request message can become the correlation ID of the response. When the sample program sends the request message, WebSphere MQ assigns a message ID to it. Assure DQ then uses this message ID as a correlation ID to send back with the response message. The sample program uses the correlation ID to match up the response with the original request.

The sample also demonstrates the use of callback data. The request message includes a callback <cb> element that enables the sample program to correlate the response and the initial request.

The SYSOUT output of IASAMP01 is shown below, including step-by-step tracing and XML sent and received.

```
 START OF TEST
 0000-ACCEPT-PARM ENTRY
  QUEUE MANAGER FOR TEST   = CSQ1
  SEND QUEUE NAME FOR TEST = IA40.REQ.ON.DEV3
  RECV QUEUE NAME FOR TEST = IA40.REPLY
 0000-ACCEPT-PARM EXIT
 1000-CONNECT ENTRY
 1100-CONNECT-QMGR ENTRY
 1100-CONNECT-QMGR EXIT
 1200-OPEN-SEND-QUEUE ENTRY
 1200-OPEN-SEND-QUEUE EXIT
 1300-OPEN-REPLY-QUEUE ENTRY
 1300-OPEN-REPLY-QUEUE EXIT
 1000-CONNECT EXIT
 2000-MAKE-REQUEST ENTRY
 REQUEST = <ex><cp><e>SAMPLE MESSAGE CE</e><p>CP1</
p><w>true</w><cb>RETURN TO ME 1</cb><trec><³›CDATA›IASAMP01
DEFAULT;A;
 100;A;100;!!></trec></cp></ex>
 2000-MAKE-REQUEST EXIT
 3000-PUT-REQUEST ENTRY
 3000-PUT-REQUEST EXIT
 4000-GET-REPLY ENTRY
 4000-GET-REPLY EXIT
 5000-SHOW-RESULTS ENTRY
  XML RETURNED = <rs><cp><e>SAMPLE MESSAGE CE</e><p>CP1</
p><rr>false</rr><w>true</w><t>2007-05-22T12:13:25.970-
05:00</t><
 cb>RETURN TO ME 1</cb><trec>IASAMP01 DEFAULT;A;100;A;100;</
trec></cp><r><rc>0</rc></r></rs>
  RESPONSE CONTROL ENTITY = SAMPLE MESSAGE CE
  RESPONSE CONTROL POINT  = CP1
  RESPONSE CALLBACK DATA  = RETURN TO ME 1
  RESPONSE RETURN CODE    = 0000
 5000-SHOW-RESULTS EXIT
 6000-DISCONNECT ENTRY
 6000-DISCONNECT EXIT
 END OF TEST
```

# Sample Program 2

IASAMP02 demonstrates fire-and-forget messaging. This is an asynchronous style of messaging where an application program sends messages to Assure DQ but does not expect any response. The request is in the form of a binary <brec> message containing a single instance of data for Assure DQ to process.

The SYSOUT output of IASAMP02 is shown below, including step-by-step tracing and XML sent.

```
 START OF TEST
 0000-ACCEPT-PARM ENTRY
  QUEUE MANAGER FOR TEST   = CSQ1
  SEND QUEUE NAME FOR TEST = IA40.REQ.ON.DEV3
 0000-ACCEPT-PARM EXIT
 1000-CONNECT ENTRY
 1100-CONNECT-QMGR ENTRY
 1100-CONNECT-QMGR EXIT
 1200-OPEN-SEND-QUEUE ENTRY
 1200-OPEN-SEND-QUEUE EXIT
 1000-CONNECT EXIT
 2000-MAKE-REQUEST ENTRY
 REQUEST = <ex><cp><e>SAMPLE MESSAGE CE</e><p>CP2</
p><w>false</
w><brec>ycHiwdTX8PJAxMXGweTT40BAQEBAQEBAQEBAQEBAQEBAQEBAQE
 BAQMFAQEBAQEBAQEAAAAABwkBAQEBAQEBAQAAAAAAAAAAd</brec></
cp></ex>
 2000-MAKE-REQUEST EXIT
 3000-PUT-REQUEST ENTRY
 3000-PUT-REQUEST EXIT
 6000-DISCONNECT ENTRY
 6000-DISCONNECT EXIT
 END OF TEST
```

# Example Files on the API CD

The table below describes the sample COBOL programs and supporting copybooks, JCL, XML message schema, and control entity .zip file included in the COBOL folder of the *API Guide and Examples* CD.

| COBOL Samples | |
|---|---|
| IASAMP01.cbl | Sample COBOL program that performs wait-for-response messaging. The program generates a text <trec> message. The PARM section of the program accepts overrides for the queue manager name, queue names, and text data to pass. |
| IASAMP02.cbl | Sample COBOL program that performs fire-and-forget messaging. The program generates a binary <brec> message using an internal 01 COBOL structure that has packed and binary data. |
| IASAMP01.out | The SYSOUT output of IASAMP01, including step-by-step tracing and XML sent and received. |
| IASAMP02.out | The SYSOUT output of IASAMP02, including step-by-step tracing and XML sent. |
| IAXMLREQ.cpy | A subprogram in the form of a copybook used by both sample programs. The subprogram provides a COBOL call interface of parameters and returns an XML document suitable for sending to the Assure DQ server over the messaging infrastructure. It will form a text <trec> or binary <brec> message within the basic request structure based on a parameter passed as the *API Guide* recommends. Optionally, it can also include the callback <cb> element. This subprogram uses RCBINB64 to form base 64 binary encoded content. |
| IAXMLRSP.cpy | A subprogram in the form of a copybook used by IASAMP01. The subprogram does a COBOL-based SAX parse of the response message from Assure DQ, returning the information into parameterized, COBOL-friendly fields and arrays. |
| Message.xsd | The XML schema that fully describes the message structure, for reference and for builds. See Appendix A, “Assure DQ Message Schema” for more details. |
| runsamp.jcl | Sample JCL that runs IASAMP01. Demonstrates the use of PARM (see step 7 on page 17) to override defaults. |
| runsamp2.jcl | Sample JCL that runs IASAMP02. |

| COBOL Samples | |
|---|---|
| Sample_Message_CE | A .zip file of the Assure DQ rules for the control entity referenced by the sample programs. CP1 (control point 1) is for IASAMP01 and has capture logic from a delimited text data API layout. CP2 is for IASAMP02 and captures data from a fixed layout corresponding to the 01 in that program. CP3 is for a scenario where control field values are passed directly to Assure DQ via an API. There is no corresponding sample program for this scenario. |

# Uploading the Samples

Three z/OS® XMIT files in the COBOL folder of the *API Guide and Examples* CD contain the sample COBOL messaging programs, copybooks, and supporting JCL. The table below describes the contents of these XMIT files.

| XMIT Files Containing COBOL Samples | |
|---|---|
| apisamp.coblib.xmit | PDS containing the source code for the COBOL sample programs and copybooks used by the sample programs. |
| apisamp.cntl.xmit | PDS containing JCL to compile, create queues, and run the sample programs. |
| apisamp.load.xmit | Compiled executables/loads of the two COBOL sample programs. |

If you would like to try running the sample programs, complete the following procedure to upload the XMIT files to z/OS.

1. Using the file transfer program of your choice (most 3270 emulators use IND$FILE), perform a binary transfer of the XMIT files from a PC with z/OS connectivity, forcing the following parameters:

   ```
   Logical Recordsize=80, Blocksize=3120, Record Format=FB
   (aka LRECL=80,BLKSIZE=3120,RECFM=FB)
   ```

   You can use the default file name or override the name with a data set name of your choice.

2. Receive the files one at a time, as shown in the following list:

   ```
   RECEIVE INDS('APISAMP.COBLIB.XMIT') will create
   'userid.ASSURE.APISAMP.COBLIB' by default.
   RECEIVE INDS('APISAMP.CNTL.XMIT') will create
   'userid.ASSURE.APISAMP.CNTL' by default.
   ```

```
RECEIVE INDS('APISAMP.LOAD.XMIT') will create
'userid.ASSURE.APISAMP.LOAD' by default.
```

At this point you have unloaded all the files. The data set names generated at your site will be influenced by the user ID used to receive these files.

3. Rename the files (optional).

   You can rename any of the files obtained from the CD as necessary.

4. Optionally, delete the XMIT files from z/OS; they will not be used again.

The contents of the apisamp.coblib.xmit are described in the table below.

| Contents of apisamp.coblib.xmit directory | |
|---|---|
| IASAMP01 | Source code for sample COBOL program that performs wait-for-response messaging. |
| IASAMP02 | Source code for sample COBOL program that performs fire-and-forget messaging. |
| IAXMLREQ.cpy | A subprogram in the form of a copybook used by both sample programs. The subprogram provides a COBOL call interface of parameters and returns an XML document suitable for sending to the Assure DQ server over the messaging infrastructure. |
| IAXMLRSP.cpy | A subprogram in the form of a copybook used by IASAMP01. The subprogram does a COBOL-based SAX parse of the response message from Assure DQ, returning the information into parameterized COBOL friendly fields and arrays. |
| RCBINB64 | Contains instructions for downloading RCBINB64, which is free software from Redvers Consulting Limited, licensed under the terms of the GNU General Public License as published by the Free Software Foundation. |

The contents of apisamp.cntl.xmit are described below.

| Contents of apisamp.cntl.xmit directory | |
|---|---|
| runsamp.jcl | Sample JCL that runs IASAMP01. Demonstrates the use of PARM (see step 7 on page 17) to override defaults. |
| runsamp2.jcl | Sample JCL that runs IASAMP02. |

The contents of the apisamp.load.xmit are described in the table below.

| Contents of apisamp.load.xmit directory | |
|---|---|
| IASAMP01 | Compiled sample COBOL program that performs wait-for-response messaging. |
| IASAMP02 | Compiled sample COBOL program that performs fire-and-forget messaging. |

# Running the COBOL Messaging Samples

If you would like to experiment with the COBOL messaging samples, complete the following steps.

1. Transfer and receive the XMIT files for control JCL, source code, and executable code to the host environment (follow the procedure beginning on page 15).
2. Import the Sample_Message_CE.zip file to the Assure DQ server that will process the messages.
3. Establish the MQ Series queues and channels to communicate with the queues and queue manager where the Assure DQ server is running.

   The control JCL library contains member CSQUTIL and member MQSETUP. These members define the queues and channels used in the development of the samples. They may be useful as a starting point.
4. To the extent necessary, start listeners and channels to ensure connectivity to the queue manager and queues that are used by the Assure server.
5. The sample programs in their executable form can be used as is. If you wish to change default values in the programs (for queue manager, queue names, timeouts, etc.), make those changes and recompile.
6. The control JCL library contains a COMPILE JCL member and a COBOL compile and link procedure (PCOBCLMQ) that includes MQ Series source and link libraries. Tailor these as necessary and run to produce newer executable load modules.
7. Edit and run member RUNSAMP in the control JCL library. This JCL executes the IASAMP01 program. The program accepts an optional PARM of comma delimited information.

If you did not edit and recompile the source to contain defaults for the queue manager and queue names you established in step 5 above, you will need to specify some of these values. The PARM delimited content is 1) queue manager (subsystem), 2) send queue name, 3) receive queue name, and 4) text data to send to the control point. Some commented example PARM values are included in the JCL.

If you ran the control point with the default data, the expected return code is zero. You can see the data stored in the Assure DQ database by selecting Results > Control Data in the user interface. Then, select Sample Message CE and click View for IASAMP01 DEFAULT.

If you used the default input data, you should see the following data in the database when you click View.

| Control Entity SAMPLE MESSAGE CE | |
|---|---|
| **Field Name** | **Control Values** |
| Sample_ID_Field | IASAMP01 DEFAULT |
| Text_Field1 | A |
| Number_Field2 | 100 |
| Text_Field3 | A |
| Number_Field4 | 100 |
| rerun | Optional |
| runNoOverride | Optional |
| sourceTimestamp | Required |
| timeout | Optional |
| waitForResponse | Optional |

The default data sent by IASAMP01 does not produce validation results. You can vary the input data with the PARM in order to produce results. See the second sample, below.

You can also cause the control point execution to fail by attempting to supply non-numeric data for numeric fields.

8. If you want to run the IASAMP02 program, download RCBINB64 from Redvers Consulting Limited.

   RCBINB64 is a COBOL subroutine that converts arbitrary binary data in COBOL to XML compliant, base 64 binary coded strings. IASAMP02 uses this subroutine to send binary data.

   At publication time, RCBINB64 was available from http://www.redversconsulting.com/RCBINB64.php.

9. Replace the RCBINB64 member in userid. ASSURE.APISAMP.CNTL with the RCBINB64 source, and compile it.

10. Edit and run member RUNSAMP2 in the control JCL library to execute the IASAMP02 program.

    After running the sample program, you can see the data stored in the Assure DQ database by selecting Results > Control Data in the user interface. Then, select Sample Message CE and click View for IASAMP02 DEFAULT.

    The second sample will produce results—the validation rules will detect problems. To see the result catalog, select Results > By Control Point and then select Sample Message CE. If you used the default input data, you should see the following information on the Results tab.

| Result Catalog SAMPLE MESSAGE CE.CP2 | |
|---|---|
| **Description** | **Severity** |
| Point to point balancing error in SAMPLE_MESSAGE_CE.CP2 for Number_Field2 | Error |
| Point to point balancing error in SAMPLE_MESSAGE_CE.CP2 for Text_Field2 | Error |

## Problem Resolution

It is likely that the first few attempts to run the samples will not succeed. This is most likely due to queue setup. The sample programs report the MQ Series completion codes and the corresponding reason code whenever an MQ call fails. The first sample is likely to fail with a reason code 2033 (a timeout), if listeners, channels, and transmission queues between the queue managers are not working. If the Assure DQ server received a message, but could not parse the XML content, it will record an error in the server-side logs.

# 3

# Java Messaging Sample

This chapter describes how to run a sample Java program that demonstrates sending and receiving JMS messages from Assure DQ. The *API Guide and Examples* CD includes the sample program, which demonstrates wait-for-response or fire-and-forget messaging, depending on parameters specified in a properties file.

## Sample Program

IASAMP01 is a sample application program that sends messages to Assure DQ and processes the results sent back by the server. The sample executes either a Single Capture or Multiple Capture control point. The Single Capture control point captures and validates a single instance of API data. The Multiple Capture control point captures and processes multiple instances passed by a single API call.

The data sent by the sample program is in the form of <trec> all text records, each containing a single instance of data for Assure DQ to process. The sample program parses the response from the Assure DQ server and displays the return code or error messages that were returned.

The sample demonstrates how the message ID of the request message can become the correlation ID of the response. When the sample program sends the request message, JMS assigns a message ID to it. Assure DQ then uses this message ID as a correlation ID to send back with the response message. The sample program uses the correlation ID to match up the response with the original request.

The sample also demonstrates the use of callback data. The request message includes a callback <cb> element that enables the sample program to correlate the response and the initial request.

The sample is highly configurable. The program can send any text data to any control point of any control entity, and (optionally) wait for a response.

The sample is configured to wait for a response—the w (wait) parameter is set to true in the request message:

```
<w>true</w>
```

If you prefer, you can request a response only when an error occurs. In the request message, substitute the re (respond on error) parameter for the w (wait) parameter:

```
<re>true</re>
```

When the re parameter is set to true, the system only returns a response if a control produces a non-zero return code or result. The system will not send a response when capture processing summaries and other system-generated informational results are produced.

# Running the Java Messaging Sample Program

This section explains how to experiment with the Java messaging samples included on the *API Guide and Examples* CD. \Java\AssureMessageSample.zip contains the sample Java messaging program. Although the .zip file is in the form of an Eclipse™ project, you can simply unzip the sample and use it directly by following the steps below. If you prefer to import the sample into an Eclipse workspace, follow the steps for "Running the Sample in Eclipse" on page 23.

## Running the Sample From the Command Line

1. Import the Sample_Message_CE.zip file to the Assure DQ server that will process the messages. The file is in the Java folder of the CD.
2. Unzip AssureMessageSample.zip, also located in the Java folder.
3. In the AssureMessageSample folder you just unzipped, look for compile.bat.
4. Open compile.bat and change the JAVA_HOME environment variable, as needed.
5. Run compile.bat to compile the sample program.
6. Open runsamp.bat and change the JAVA_HOME environment variable, as needed.
7. Open the config folder. In the IASAMP01.properties file, change the following Provider URL setting as needed for your deployment:

```
# Context.PROVIDER_URL
java.naming.provider.url=http-remoting://
<serverhostname>:8080
```

The sample is set up to connect to a WildFly instance produced by a standard deployment of the installer.zip. The setting above connects to an Assure DQ server on the same machine where you unzipped AssureMessageSample.zip. Port 1099 is the Assure DQ and WildFly default.

8. The properties file is configured to run the CP1 Single Capture control point. If you prefer to execute the CP1MANY Multiple Capture control point, modify the properties described below.
   a. Uncomment the CP1MANY control point and comment out the CP1 control point.

```
ControlPoint=CP1MANY
#ControlPoint=CP1
```

   b. Comment out the IASAMP01 input records and uncomment the IASAMP01a record.

```
# the input record (the following will produce an in-
balance result)
#RECORD=IASAMP01;A;100;A;100
# alternate input record that would produce an out-of-
balance result
#RECORD=IASAMP01;A;100;B;100
RECORD=IASAMP01a;A;100;A;100*IASAMP01b;B;200;B;200
*IASAMP01a;C;100;C;110*
```

9. Additional changes to the properties file will be necessary if you are using a different application server or have altered the messaging settings in the Assure DQ server.
10. Execute runsamp.bat to run the sample program.
11. Compare your results to the outputs shown in "Expected Output of the Single Capture Control Point" on page 25 or "Expected Output of the Multiple Capture Control Point" on page 27.

## Running the Sample in Eclipse

If you prefer to import and run the sample program in an Eclipse workspace, complete the following steps:

1. Go to the workspace and click File > Import > Existing Projects Into Workspace.
2. Click Select Archive File and select AssureMessageSample.zip. The file is in the Java folder of the CD.
3. Click Finish.

4. Open the config folder. In the IASAMP01.properties file, change the following Provider URL setting as needed for your deployment:

```
# Context.PROVIDER_URL
java.naming.provider.url=http-remoting://
<serverhostname>:8080
```

   The sample shown above is set up to connect to a WildFly instance of an Assure DQ server on port 1099. This is the Assure DQ and WildFly default.

5. The properties file is configured to run the CP1 Single Capture control point. If you prefer to execute the CP1MANY Multiple Capture control point, modify the properties described below.
   a. Uncomment the CP1MANY control point and comment out the CP1 control point.

```
ControlPoint=CP1MANY
#ControlPoint=CP1
```

   b. Comment out the IASAMP01 input records and uncomment the IASAMP01a record.

```
# the input record (the following will produce an in-
balance result)
#RECORD=IASAMP01;A;100;A;100
# alternate input record that would produce an out-of-
balance result
#RECORD=IASAMP01;A;100;B;100
RECORD=IASAMP01a;A;100;A;100*IASAMP01b;B;200;B;200
*IASAMP01a;C;100;C;110*
```

6. Additional changes to the properties file will be necessary if you are using a different application server or have altered the messaging settings in the Assure DQ server.
7. Right-click IASAMP01.java and run it as a Java application.
8. Compare your results to the outputs shown in "Expected Output of the Single Capture Control Point" on page 25 or "Expected Output of the Multiple Capture Control Point" on page 27.

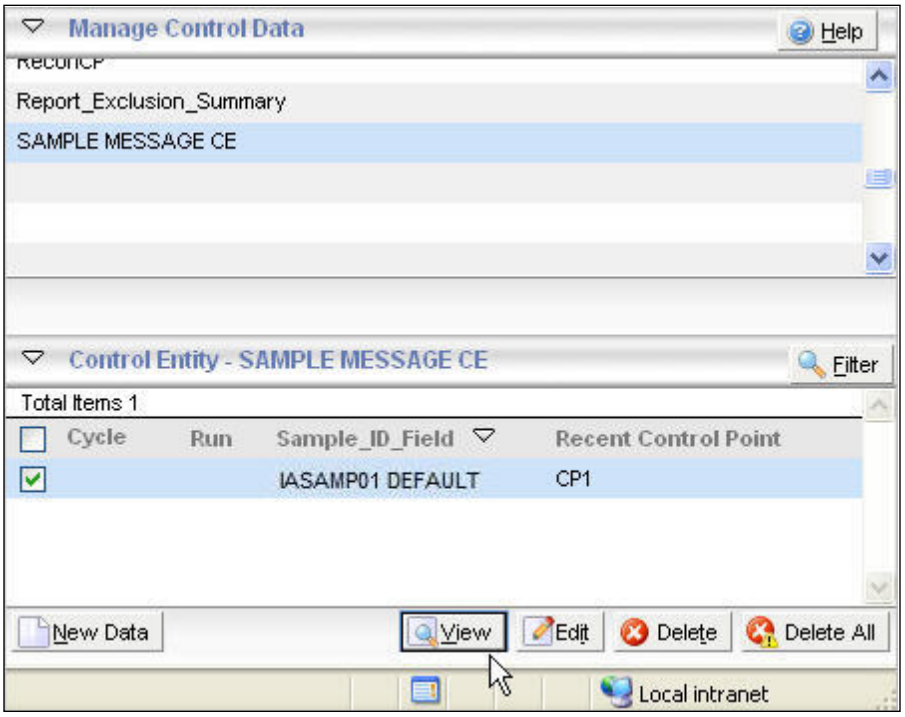
## Expected Output of the Single Capture Control Point

Shown below is the expected output of the CP1 Single Capture control point, including step-by-step tracing and XML sent and received. The sample output is also in SampleOutput.txt in the output folder of AssureMessageSample.

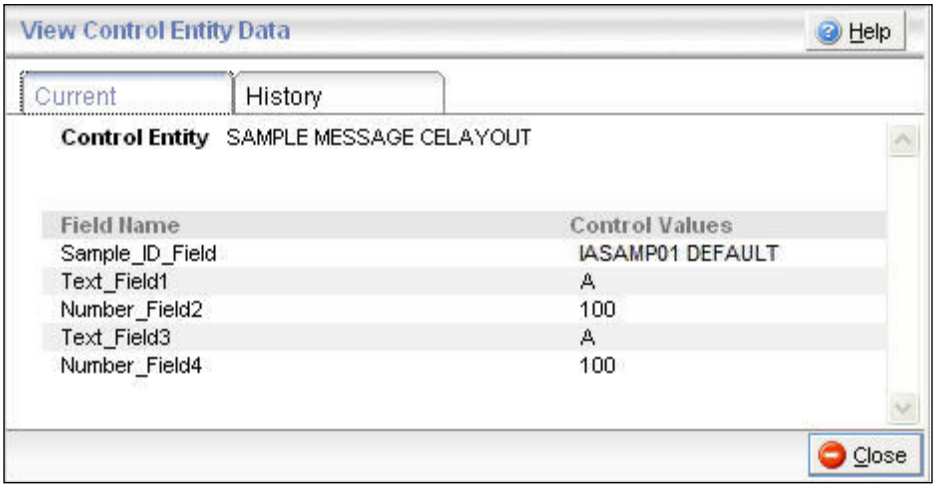
```
IASAMP01 entry
Loading configuration properties from config/
IASAMP01.properties
Establishing Queue Resources...
 Queue Connection Factory name = ConnectionFactory
 Send Queue Name is queue/CASAPIInputQueue
 Response Queue Name is queue/CASAPIOutputQueue
Forming and sending message to Assure...
 Sending message: <ex><cp><e>SAMPLE MESSAGE CE</e><p>CP1</
p><w>true</w><cb><![CDATA[<extra>abc123</extra>]]></
cb><trec><![CDATA[IASAMP01;A;100;A;100]]></trec></cp></ex>
 Message sent
 Retrieving response from Assure...

 Response message from Assure: <rs><cp><e>SAMPLE MESSAGE
CE</e><p>CP1</p><rr>false</rr><w>true</w><t>2007-06-
07T06:54:22.554-05:00</t><cb>&lt;extra&gt;abc123&lt;/
extra&gt;</cb><trec>IASAMP01;A;100;A;100</trec></
cp><r><rc>0</rc></r></rs>
The response:
 Control Entity: SAMPLE MESSAGE CE
 Control Point: CP1
 Rerun: false
 Wait for response: true
 Timestamp: 2007-06-07T06:54:22.554-05:00
 Callback data: <extra>abc123</extra>
 Text Record: IASAMP01;A;100;A;100
 Return code: 0
Closing Queue Resources...
IASAMP01 exit
```

If you ran the control point with the default data, the expected return code is zero. You can see the data stored in the Assure DQ database by selecting Results > Control Data in the user interface. Then, select Sample Message CE and click View for IASAMP01, as shown below.




If you used the default input data, you will see the following data in the database when you click View.



## Expected Output of the Multiple Capture Control Point

If you ran the CP1Many control point, go to Results > By Control Point. In the top pane, make sure Exclude API Catalog is not checked. Click Apply. Select SAMPLE MESSAGE CE - CP1MANY.

On the Reports tab in the bottom pane, select and view the Control Report. The report will look similar to the one shown below.

# Control Report

INFOGIX

## IASAMP01a

| | |
|---|---|
| Created | December 5, 2018 11:24:35 AM CST |
| Control Entity | SAMPLE MESSAGE CE |
| Control Point | CP1MANY |

### Validation Inputs

| Name | Value |
|---|---|
| current | |
| Sample_ID_Field | 'IASAMP01a' |
| Text_Field1 | 'A' |
| Number_Field2 | 200 |
| Text_Field3 | 'C' |
| Number_Field4 | 210 |

### Validation Rule Set:test

Derived Values

None

Validations

| Condition | Outcome |
|---|---|
| current.Text_Field1 = current.Text_Field3<br>A = C<br>false | Failed |
| current.Number_Field2 = current.Number_Field4<br>200 = 210<br>false (*** -10 ***) | Failed |

### Actions

Store Control Data

# 4

# API Message Dispatcher

This chapter describes a sample Java program that reads JMS messages from your queue and sends them to Assure DQ for control processing. The program references a properties file to determine the output queue to access and the Assure DQ control entity and control point to run. The *API Guide and Examples* CD includes the sample program.

## Sample Program

CustomAPIMessageDispatcher is a sample application program that reads JMS messages from a queue of your choice and sends them to the Assure DQ API queue for processing by a designated control entity and control point. The sample executes a Single Capture control point to capture a single instance of API data.

## Running the Sample Program

This section explains how to experiment with the API message dispatcher program included on the CD. \Java\AssureMessageSample.zip contains the sample program. Although the .zip file is in the form of an Eclipse project, you can simply unzip the sample and use it directly by following the steps below. If you prefer to import the sample into an Eclipse workspace, follow the steps for "Running the Sample in Eclipse" on page 31.

### Running the Sample From the Command Line

1. Import the Sample_Message_CE.zip file to the Assure DQ server that will process the messages.

   The file is in the Java folder of the CD.
2. Unzip AssureMessageSample.zip and look for capture.txt.
3. Copy the first line of capture.txt to your message queue to serve as a test message.
4. Open compile.bat and change the JAVA_HOME environment variable, as needed.
5. Run compile.bat to compile the sample program.

6. Open runsamp_dispatcher.bat and change the JAVA_HOME environment variable, as needed.
7. Open the CustomAPIMessageDispatcher.properties file in the config folder.
8. Specify the names of the control entity and control point to run:

   CE_API_Demo_Capture_Text_Data
   CP_API_Demo_Capture_Text_Data
9. To run the program in a WebSphere® environment, comment out all of the WildFly parameters and uncomment the WebSphere parameters.
10. Change the following Provider URL setting as needed for your deployment:

    *WildFly:*

    ```
    java.naming.provider.url=http-remoting://
    <serverhostname>:8080
    ```

    *WebSphere:*

    ```
    java.naming.provider.url=iiop://localhost:2809
    ```

    The sample is set up to connect to a WildFly or WebSphere instance produced by a standard deployment of Assure DQ. The setting above connects to an Assure DQ server on the same machine where you unzipped AssureMessageSample.zip. Port 1099 is the Assure DQ and WildFly default. Port 2809 is the Assure DQ and WebSphere default.
11. Change the FROM_QUEUE_NAME parameter to the name of your message queue.
12. For WebSphere only, change the FROM_CONNECTION_FACTORY parameter to the name of the connection factory for your message queue.
13. Execute runsamp_dispatcher.bat to run the sample program.
14. Compare your output to the sample output shown on page 32.

## Running the Sample in Eclipse

If you prefer to import and run the sample program in an Eclipse workspace, complete the following steps:

1. Place a sample message on your message queue.
2. Go to the Eclipse workspace and click File > Import > Existing Projects Into Workspace.
3. Click **Select Archive File** and select AssureMessageSample.zip. The file is in the Java folder of the CD.
4. Click Finish.
5. Open the CustomAPIMessageDispatcher.properties file in the config folder.
6. Specify the names of the control entity and control point to run.
7. To run the program in a WebSphere environment, comment out all of the WildFly parameters and uncomment the WebSphere parameters.
8. Change the following Provider URL setting as needed for your deployment:

   *WildFly:*

   ```
   java.naming.provider.url=http-remoting://
   <serverhostname>:8080
   ```

   *WebSphere:*

   ```
   java.naming.provider.url=iiop://localhost:2809
   ```

   The sample is set up to connect to a WildFly or WebSphere instance produced by a standard deployment of Assure DQ. The setting above connects to an Assure DQ server on the same machine where you unzipped AssureMessageSample.zip. Port 1099 is the Assure DQ and WildFly default. Port 2809 is the Assure DQ and WebSphere default.
9. Change the FROM_QUEUE_NAME parameter to the name of your message queue.
10. For WebSphere only, change the FROM_CONNECTION_FACTORY parameter to the name of the connection factory for your message queue.
11. Right-click CustomAPIMessageDispatcher.java and run it as a Java application.
12. Compare your output to the sample output shown on page 32.

## Sample Output of the API Message Dispatcher

Shown below is the expected console output of the message dispatcher sample program.

```
Reading configuration from config/
CustomAPIMessageDispatcher.properties
Connecting to queues...
 From Queue Name is queue/Your_source_queue
 To Queue Name is queue/IA_CASAPIInputQueue
Start reading messages to dispatch to Assure DQ...
 timeout in 60 seconds if there are no messages
 There are no more messages
Dispatched 1 messages
```

If you like, you can also log in to the Assure DQ user interface to verify the control processing results. Go to Results > By Control Point. In the top pane, make sure Exclude API Catalog is not checked. Click Apply. Select SAMPLE MESSAGE CE - CP1.

On the Reports tab in the bottom pane, select and view the Control Report. Verify that the control run produced a return code of 101.

# 5

# SOAP Web Service for Control Point Execution

The Assure DQ server includes a Web service that enables a calling client program to pass data to the server, request Assure DQ processing, and get back processing results. The client program can point to a file to process or pass individual transactions to Assure DQ. The client can also simply request processing of a control point and get back results. Both synchronous and asynchronous processing are supported.

Synchronous processing is only recommended when the server processes a low volume of calls or the calls have short wait for response times. By default, no more than 20 calls can wait for a response at the server. Too many calls waiting can result in denial of service to client programs. When this happens, no further calls are processed.

The default wait for response time is five minutes. This overrides wait times specified in client programs. If you need to change the default settings for synchronous processing, see the troubleshooting section at the end of this chapter.

To use the Web service, add code to an existing application or create a new client program to call the Web service. The Web service supports both Java and .NET clients.

The *API Guide and Examples* CD includes a sample Java client program. The sample demonstrates how to point to a file to process, or pass an individual transaction and process it in synchronous or asynchronous mode.

The sample Java client, source code for the client, and libraries and executables necessary to build, compile, and run the client are in the WebServices folder on the CD. Also included are a WSDL (Web Services Description Language) file for the Web service, programmer documentation for the Web service, and sample data and control point rules to use with the sample client.

The contents of the soapwsclientexample folder are described in the table below.

| Contents of soapwsclientexample directory | |
|---|---|
| dist | Contains the configuration file, batch and shell script files, and jar file needed to run the sample client. The jar is already built. To run the sample client, the only requirement is a Java runtime environment. See "Running the Sample Client" on page 34. |
| lib | Contains all third party jars necessary to build and run the sample client. |
| samples | Contains a properties file template, .zip files of the Assure DQ rules for the control entities/control points referenced in the sample, and a sample input file. |
| schema | Contains a copy of the WSDL for the Web service, for reference and for builds. |
| src | Contains source code for the sample Java client. |
| build.xml | Contains an Ant build file to generate the required Axis classes and compile and jar the sample client. |
| jaxws-ri-2.3.2.zip | Contains JAX-WS RI 2.3.2. JAX-WS RI 2.3.2 is a Web Services framework that provides tools and infrastructure to develop Web Services solutions for the end users and middleware developers. Unzip this in the current folder to create the jaxws-ri folder. |

# Running the Sample Client

If you would like to experiment with the sample Java client shipped with Assure DQ, complete the following steps:

1. Import CE_API_Demo_Capture_SourceFile.zip and CE_API_Demo_Capture_Text_Data.zip into Assure DQ using the Control Entities/Import Rules function.

   The .zip files are located in SOAPWebServices\samples\rules\.
2. Copy capture.txt from SOAPWebServices\samples\data\ to a directory that can be accessed by the Assure DQ server.
3. Edit the application.properties file in the SOAPWebServices\dist directory, as needed.
   a. Specify the correct host, port, Assure DQ user ID and password for your installation.

b. If the API is triggering a control point with external data sources, comment out the following line:

   ```
   <target>.TextDataItem=
   ```

   where <target> is the value of the TargetExample property.

c. If the API is triggering a control point with a new external data source that will override the path/URL used by the capture source for the control point, uncomment and use the following lines:

   ```
   <target>.CaptureSource
   <target>.RemoteFileLocation=<path of server file>
   ```

   where <target> is the value of the TargetExample property, and where <path of server file> is the value of the new path/URL for the capture source given.

   In this scenario, in the Assure DQ user interface, do not change the layout data access method or the capture source override data access method to “API Data”.

d. Modify the following POLL properties as needed for long running tasks:

   POLL_ATTEMPT (integer, number of attempts)

   POLL_TIMEOUT (millisecond 60,000 = 1 minute)

   The poll attempt value multiplied by the poll timeout value should be larger than the anticipated longest running task.

Notice that the properties file includes three examples. Example 1 applies controls to data in a sample file, capture.txt. Example 2 passes a single transaction to Assure DQ for synchronous control processing. Example 3 does the same thing as Example 2, but asynchronously.

## Modifying and Compiling the Sample Client

If you modify the sample Java client, you must compile the sample before running it. The Ant software must be installed on the server or a development machine. To compile the sample, complete the following steps:

1. At the command prompt on the machine where you will run the example, change to the folder where you installed \examples\soapwsclientexample\dist.
2. At the command prompt, set the ANT_HOME environment variable to the directory of your Ant home folder.
3. At the command prompt, set the JAVA_HOME environment variable to the directory of your local JDK.

Use a command in the following format, where JAVA_LOCATION is the directory of your local JDK:

*On Windows:*

```
SET JAVA_HOME=JAVA_LOCATION
```

*On Unix or Linux®:*

```
JAVA_HOME=JAVA_LOCATION
export JAVA_HOME
```

4. At the command prompt, execute the command to compile the sample.

   Use a command in the following format:

   *On Windows:*

   ```
   %ANT_HOME%\bin\Ant -f build.xml dist
   ```

   *On Unix or Linux:*

   ```
   $ANT_HOME\bin]Ant -f build.xml dist
   ```

   The compile command generates the jar file soapwsclientexample.jar.

## Running Example 1

1. To run example 1, verify that the TargetExample parameter in the application.properties file is set to example1.
2. In the Example 1 Properties section of the properties file, substitute the full path of capture.txt (see step 2 on page 34) for RemoteFileLocation.
3. At the command prompt on the machine where you will run the example, set the JAVA_HOME environment variable to the directory of your local JRE.

   Use a command in the following format, where *JAVA_LOCATION* is the directory of your local JRE:

   *On Windows:*

   ```
   SET JAVA_HOME=JAVA_LOCATION
   ```

   *On UNIX or Linux:*

   ```
   #!/bin/sh
   JAVA_HOME=JAVA_LOCATION
   export JAVA_HOME
   ```

4. Change to the SOAPWebServices\dist directory, where run.bat and run.sh are located.
5. Execute run.bat or run.sh as follows:

   `run.bat application.properties` or
   `run.sh application.properties`

6. Review the results of the run in the Assure DQ user interface.

### Running the Other Examples

To run example 2 or example 3, change the TargetExample parameter in the application.properties file to the correct example number.

# Request–Response Operations (Methods)

The Web service provides two methods: ProcessControlPoint and GetControlPointResult.

**ProcessControlPoint** executes a control point. The method offers multiple options for process invocation and responds with errors, a result, or a cookie. Either data capture sources can be specified or data can be passed to Assure DQ via the request. The method supports both synchronous and asynchronous processing.

This method has the following two types of messages:

- **ControlPointRequest**—Input message used to call ProcessControlPoint. See page 38 for more details.
- **ControlPointResponse**—Output message returned when calling ProcessControlPoint. See page 41 for more details.

**GetControlPointResult** is used to access processing results in asynchronous mode and when a request times out. When a request has been processed successfully or times out, Assure DQ will return a cookie for accessing the results. This method has the following two types of messages:

- **ControlPointResultRequest**—Input message used to call GetControlPointResult. Requires a cookie ID from an asynchronous call or from a timed-out request. See page 43 for more details.
- **ControlPointResultResponse**—Output message returned when calling GetControlPointResult. If the control point generated results, the message will include a result code and result catalog ID. Otherwise, it will include a message indicating results do not exist. See page 43 for more details.

# Authentication and Access

By default, basic authentication is enabled for the Assure DQ Web Services. This means that when a client program calls the Web Service, the system authenticates the user ID and password against the security directory specified for Assure DQ. In addition, the user ID must have one of the following Assure DQ roles:

- Superuser
- Remote Executor

HTTP basic authentication is implemented for WildFly and WebSphere deployments. Web Services Security (WSS) is not implemented.

The Web Services security settings are in the IA.properties file located in *infogix.install.home*\Infogix\config*config.name*\IA. If you do *not* want Assure DQ to authenticate Web service calls, set the WS-AUTH_TYPE property to none and redeploy Assure DQ

## Access

| Assure DQ SOAP Web Service for Control Point Execution Properties | |
|---|---|
| Web service name | ControlsAssureService |
| Endpoint URL | http://<host:port>/casapi/services/ControlsAssureService |
| WSDL URL | http://<host:port>/casapi/services/ControlsAssureService?wsdl |
| Supported bindings | SOAP (Simple Object Access Protocol) |
| Port type | ControlsAssureService |

# Web Service Element Dictionary

This section describes the messages exchanged between the client and the Web service, and their data elements.

## ControlPointRequest Message

| ControlPointRequest | | | |
|---|---|---|---|
| **Attribute** | **Use** | **Type** | **Description** |
| controlEntityName | Required | String | The name of the control entity. |
| controlPointName | Required | String | The name of the control point within the control entity to process. |
| controlEntityId | Optional | String | Unique ID for the request and control entity instance. Used for internal queuing of requests and run-to-run identification for sequencing control points. |

| ControlPointRequest | | | |
|---|---|---|---|
| **Attribute** | **Use** | **Type** | **Description** |
| controlReportsLimit | Optional | int | Can be used to override the system and control point settings for the maximum number of Control Reports to output. To permit unlimited reports, specify -1. |
| cycleNoOverride | Optional | Long | Can be used to override the cycle number specified for a control entity. Can also be used with the rerun attribute to process a specific run again. |
| developmentMode | Optional | Boolean | Reserved for future use. Defaults to false. |
| rerun | Optional | Boolean | Indicates if this execution is a rerun. For reruns, control data and findings are deleted and replaced in the database. Defaults to false. |
| resultsLimit | Optional | int | Can be used to override the system and control point settings for the maximum number of results to output. To permit unlimited results, specify -1. |
| runNoOverride | Optional | Long | Can be used to override the run number specified for a control entity. Can also be used with the rerun attribute to process a specific run again. |
| sourceTimestamp | Required | dateTime | Set to the client's current timestamp. Reserved for future use. |
| timeout | Optional | int | Can be used to override how long, in milliseconds, a synchronous call should wait for a response. If a response takes longer, Assure DQ will return a cookie for delayed results checking. Defaults to 10000 milliseconds. |
| waitForResponse | Optional | Boolean | Can be used to override the default processing mode, which is asynchronous. When set to true (synchronous mode), Assure DQ will process the request and then return the processing result, a timeout cookie, or a processing error. In asynchronous mode, Assure DQ will return a cookie (reference) string or a processing error once the request has been queued. Defaults to false (asynchronous mode). |

| ControlPointRequest | | | |
|---|---|---|---|
| **Element** | **Nillable** | **Type** | **Description** |
| ctlrptlmt | True | int | An integer specifying the Control Reports limit for the current control point run. To permit unlimited reports, specify -1. |
| rslmt | True | int | An integer specifying the results limit for the current control point run. To permit unlimited results, specify -1. |
| Specify **only one** of the following four elements: | | | |
| binaryDataItem | True | base64binary | Used when capturing records containing binary data encoded as base64. |
| textDataItem | True | String | Used when capturing records containing text data. |
| captureSourceList | True | CaptureSourceListType–An unbounded sequence of CaptureSourceType elements (see description, below). | Used when capturing data from files in order to override default URLs. List named capture sources and their URL overrides. Specify URLs from the server's context, not that of the calling client. |
| controlValueList | True | ControlValueListType–An unbounded sequence of ControlValueType elements (see description on page 41). | Used when directly inputting data. List name value pairs to populate the control fields for this control entity. |

## CaptureSourceType Element

This element type specifies the name and URL of the data capture source.

| CaptureSourceType | | | |
|---|---|---|---|
| **Attribute** | **Use** | **Type** | **Description** |
| sourceName | Required | String | The name of the capture source, as specified for the control point. |
| sourceUrl | Required | String | The URL of the capture source. Can be the path to a file on the server or a URL. For supported URL formats, please see the Assure DQ online help for data capture sources. |

## ControlValueType Element

This element type specifies control values as name value pairs.

| ControlValueType | | | |
|---|---|---|---|
| **Attribute** | **Use** | **Type** | **Description** |
| name | Required | String | The name of the control field to pass to Assure DQ. Use the exact name specified in the control entity definitions. |
| value | Required | String | The actual value to pass to Assure DQ. The value is passed as a string and will inherit the type and format specified for the control field in the control entity definitions. |

## ControlPointResponse Message

| ControlPointResponse | | | |
|---|---|---|---|
| **Element** | **Nillable** | **Type** | **Description** |
| result | True | ControlPointResultType–See page 42. | The results of control point processing, if processing executed successfully. |

| ControlPointResponse | | | |
|---|---|---|---|
| **Element** | **Nillable** | **Type** | **Description** |
| resultCookie | True | String | A reference ID used to get results in asynchronous mode or when a timeout has occurred. This reference ID can be used with a call to GetControlPointResult. |
| error | True | ProcessingErrorType– An unbounded sequence of error messages in the form of string elements. | If errors are encountered during processing of the control point, will contain a list of error messages. Note that these are not the same as results; these are execution errors. |

## ControlPointResultType Element

This element type specifies the processing return code and catalog ID for a completed control point processing request.

| ControlPointResultType | | | |
|---|---|---|---|
| **Attribute** | **Use** | **Type** | **Description** |
| returnCode | Required | int | The return code of the control point execution as defined in validation results. |
| catalogId | Optional | String | The result catalog ID of the result catalog, if one was produced. By default, control points invoked via the Web service do not produce result catalogs. However, if a pass/fail/complete validation outcome occurs and the Result or Generate Report option is specified for that outcome, the control point will produce a result catalog. |
| instanceCount | Optional | int | The number of instances captured by the control point. For single capture control points, the instance count is 0 or 1. For multiple capture control points, the count is 0 or a positive integer. |

### ControlPointResultRequest Message

| ControlPointResultRequest | | | |
|---|---|---|---|
| **Attribute** | **Use** | **Type** | **Description** |
| resultCookie | Required | String | The reference ID that identifies a request that was made successfully in asynchronous mode or a request that timed out. |

### ControlPointResultResponse Message

| ControlPointResultResponse | | | |
|---|---|---|---|
| **Element** | **Nillable** | **Type** | **Description** |
| result | Yes | ControlPointResultType | The results of control point processing, if results were produced. |
| error | Yes | ProcessingErrorType | A message indicating results do not exist. |

# Troubleshooting

## Too Many Synchronous Calls

Too many synchronous calls waiting for a response at the server can result in denial of service to client programs. When this happens, no further calls are processed.

To prevent this problem, by default, no more than 20 calls can wait for a response at the server and the maximum wait for response time is five minutes. This overrides longer wait times specified in client programs. Additional calls are still processed, but cannot consume resources by waiting at the server.

To change these default settings, open *install_folder*\Infogix\components\ControlsAssure System\conf \api.properties and edit the MAX_CONCURRENT_WAITING _SYNC_CALLS property and/or the MAX_WAITFOR_RESPONSE _TIME_MILLIS property, as needed. If your calls typically have long wait times, consider changing to asynchronous processing.

## Timeout Values Too Low

Assure DQ provides the ability to set up timeout settings for API processing at three levels.

To change the default settings, edit *install_folder*\Infogix\components \ControlsAssure System\conf\api.properties.

### System Maximum Timeout

The system times out when calls take longer than the max wait for response time, which defaults to five minutes (300,000 milliseconds). The max wait for response time is a system-wide limit on timeout duration that takes precedence over any other parameter. The property that controls this behavior is MAX_WAITFOR_RESPONSE_TIME_MILLIS.

### System Default When Client API Does Not Set Timeouts

The system times out when calls take longer than the system wait for response time, which defaults to 10 seconds (10,000 milliseconds). This parameter is overridden by the max wait for response parameter and by the timeout value specified in the control point request message. This property should always be less than or equal to the max wait for response time. The property that controls this behavior is SYSTEM_WAITFOR_RESPONSE_TIME_MILLIS.

### API Control Point Request Message

The client API can set the timeout property for each control point request message. This attribute value takes precedence over the system wait for response time parameter. For more information, refer to the description of the “timeout” attribute in the Webservice Element Dictionary on page 39.

### Backward Compatibility

For backward compatibility to previous versions of the API:

- Set MAX_WAITFOR_RESPONSE_TIME_MILLIS to be the maximum time any client call should wait for a response. It is not possible to set an infinite time.
- Set SYSTEM_WAITFOR_RESPONSE_TIME_MILLIS to be less than the value set for MAX_WAITFOR_RESPONSE_TIME_MILLIS.

# 6

# SOAP Web Service for Control Point Results

The Assure DQ server includes a Web Service that enables a calling client program to request processing results for any previous control runs.

To use the Web Service, add code to an existing application or create a new client program to call the Web Service. The Web Service supports both Java and .NET clients.

The *API Guide and Examples* CD includes a sample Java client program. The sample demonstrates how to access and display control point processing results given a result catalog ID.

The sample Java client, source code for the client, and libraries and executables necessary to build, compile, and run the client are in the WebServices folder on the CD. The folder also contains a WSDL (Web Services Description Language) file and programmer documentation for the Web Service, and sample data and control point rules to use with the sample client.

The table below describes the contents of the soapwsclientexample folder.

| Contents of soapwsclientexample directory | |
|---|---|
| dist | Contains the configuration file, batch and shell script files, and jar file needed to run the sample client. The jar is already built. To run the sample client, the only requirement is a Java runtime environment. See "Running the Sample Client" on page 46. |
| lib | Contains all third party jars necessary to build and run the sample client. |
| samples | Contains a properties file template, zip files of the Assure DQ rules for the control entities/control points referenced in the sample, and a sample input file. |
| schema | Contains a copy of the WSDL for the Web service, for reference and for builds. |
| src | Contains source code for the sample Java client. |
| build.xml | Contains an Ant build file to generate the required classes and compile and jar the sample client. |
| jaxws-ri-2.3.2.zip | Contains JAX-WS RI 2.3.2. JAX-WS RI 2.3.2 is a Web Services framework that provides tools and infrastructure to develop Web Services solutions for the end users and middleware developers. Unzip this in the current folder to create the jaxws-ri folder. |

# Running the Sample Client

If you would like to experiment with the sample Java client shipped with Assure DQ, complete the following steps:

1. In the Assure DQ user interface, select Results > By Control Point.
   The Results by Control Point list displays.
2. Select a processing result from the list.
   On the Summary tab in the bottom pane, look for the Catalog ID of the result.
3. Run the example results script in the dist directory:
   `results application.properties catalog_ID`

## Modifying and Compiling the Sample Client

If you modify the sample Java client, you must compile the sample before running it. The Ant software must be installed on the server or a development machine. To compile the sample, complete the following steps:

1. At the command prompt on the machine where you will run the example, change to the folder where you installed \examples \soapwsclientexample\dist.
2. At the command prompt, set the ANT_HOME environment variable to the directory of your Ant home folder.
3. At the command prompt, set the JAVA_HOME environment variable to the directory of your local JDK.
   Use a command in the following format, where JAVA_LOCATION is the directory of your local JDK:

   *On Windows:*

   ```
   SET JAVA_HOME=JAVA_LOCATION
   ```

   *On UNIX or Linux®:*

   ```
   JAVA_HOME=JAVA_LOCATION
   export JAVA_HOME
   ```
4. At the command prompt, execute the command to compile the sample.
   Use a command in the following format:

   *On Windows:*

   ```
   %ANT_HOME%\bin\Ant -f build.xml dist
   ```

   *On UNIX or Linux:*

   ```
   $ANT_HOME\bin]Ant -f build.xml dist
   ```

   The compile command generates the jar file, soapwsclientexample.jar.

# Request–Response Operations (Methods)

The Web Service provides three methods: readResultCatalogSummary, listResults and listReports.

**ReadResultCatalogSummary** takes a result catalog ID and returns a response containing summary-level information about the processing results.

**ListResults** returns the detailed results for a result catalog. Your client program can page through these results. The client must pass three parameters: the result catalog ID, the starting index number for the result list, and the number of detailed results to return.

**ListReports** takes a result catalog ID and returns a response containing the list of reports produced for a result catalog.

# Authentication and Access

By default, basic authentication is enabled for the Assure DQ Web Services. This means that when a client program calls the Web Service, the system authenticates the user ID and password against the security directory specified for Assure DQ. In addition, the user ID must have one of the following Assure DQ roles:

- Superuser
- Remote Executor

HTTP basic authentication is implemented for WildFly and WebSphere deployments. Web Services Security (WSS) is not implemented.

The Web Services security settings are in the IA.properties file located in *infogix.install.home*\Infogix\config*config.name*\IA. If you do *not* want Assure DQ to authenticate Web service calls, set the WS-AUTH_TYPE property to none and redeploy Assure DQ.

## Access

| Assure DQ SOAP Web Service For Control Point Results Properties | |
|---|---|
| Web service name | ControlsAssureResultsService |
| Endpoint URL | http://<host:port>/casapi/services/ControlsAssureResultsService |
| WSDL URL | http://<host:port>/casapi/services/ControlsAssureResultsService?wsdl |
| Supported bindings | document literal |
| Port type | ControlsAssureResultsServicePort |

# 7

# REST Web Service for Control Point Execution

The Assure DQ server includes a Web service that enables a calling client program to pass data to the server, request Assure DQ processing, and get back processing results. The client program can point to a file to process or pass individual transactions to Assure DQ. The client can also simply request processing of a control point and get back results. Both synchronous and asynchronous processing are supported.

Synchronous processing is only recommended when the server processes a low volume of calls or the calls have short wait for response times. By default, no more than 20 calls can wait for a response at the server. Too many calls waiting can result in denial of service to client programs. When this happens, no further calls are processed.

The default wait for response time is five minutes. This overrides wait times specified in client programs. If you need to change the default settings for synchronous processing, see the troubleshooting section at the end of this chapter.

To use the Web service, add code to an existing application or create a new client program to call the Web service. The Web service supports both Java and .NET clients.

The *API Guide and Examples* CD includes a sample Java client program. The sample demonstrates how to point to a file to process, or pass an individual transaction and process it in synchronous or asynchronous mode.

The sample Java client, source code for the client, and libraries and executables necessary to build, compile, and run the client are in the WebServices folder on the CD. Also included are a WSDL (Web Services Description Language) file for the Web service, programmer documentation for the Web service, and sample data and control point rules to use with the sample client.

The contents of the restwsclientexample folder are described in the table below.

| Contents of restwsclientexample directory | |
|---|---|
| dist | Contains the configuration file, batch and shell script files, and jar file needed to run the sample client. The jar is already built. To run the sample client, the only requirement is a Java runtime environment. See "Running the Sample Client" on page 50. |
| lib | Contains all third party jars necessary to build and run the sample client. |
| samples | Contains a properties file template, .zip files of the Assure DQ rules for the control entities/control points referenced in the sample, and a sample input file. |
| schema | Contains a copy of the WSDL for the Web service, for reference and for builds. |
| src | Contains source code for the sample Java client. |
| build.xml | Contains an Ant build file to generate the required Axis classes and compile and jar the sample client. |
| jaxws-ri-2.3.2.zip | Contains JAX-WS RI 2.3.2. JAX-WS RI 2.3.2 is a Web Services framework that provides tools and infrastructure to develop Web Services solutions for the end users and middleware developers. Unzip this in the current folder to create the jaxws-ri folder. |

# Running the Sample Client

If you would like to experiment with the sample Java client shipped with Assure DQ, complete the following steps:

1. Import CE_API_Demo_Capture_SourceFile.zip and CE_API_Demo_Capture_Text_Data.zip into Assure DQ using the Control Entities/Import Rules function.

   The .zip files are located in RESTWebServices\samples\rules\.
2. Copy capture.txt from RESTWebServices\samples\data\ to a directory that can be accessed by the Assure DQ server.
3. Edit the application.properties file in the RESTWebServices\dist\ directory, as needed.
   a. Specify the correct host, port, Assure DQ user ID and password for your installation.

b. If the API is triggering a control point with external data sources, comment out the following line:

   ```
   <target>.TextDataItem=
   ```

   where <target> is the value of the TargetExample property.

c. If the API is triggering a control point with a new external data source that will override the path/URL used by the capture source for the control point, uncomment and use the following lines:

   ```
   <target>.CaptureSource
   <target>.RemoteFileLocation=<path of server file>
   ```

   where <target> is the value of the TargetExample property, and where <path of server file> is the value of the new path/URL for the capture source given.

   In this scenario, in the Assure DQ user interface, do not change the layout data access method or the capture source override data access method to “API Data”.

d. Modify the following POLL properties as needed for long running tasks:

   POLL_ATTEMPT (integer, number of attempts)

   POLL_TIMEOUT (millisecond 60,000 = 1 minute)

   The poll attempt value multiplied by the poll timeout value should be larger than the anticipated longest running task.

Notice that the properties file includes three examples. Example 1 applies controls to data in a sample file, capture.txt. Example 2 passes a single transaction to Assure DQ for synchronous control processing. Example 3 does the same thing as Example 2, but asynchronously.

## Modifying and Compiling the Sample Client

If you modify the sample Java client, you must compile the sample before running it. The Ant software must be installed on the server or a development machine. To compile the sample, complete the following steps:

1. At the command prompt on the machine where you will run the example, change to the folder where you installed \examples\restwsclientexample\dist.
2. At the command prompt, set the ANT_HOME environment variable to the directory of your Ant home folder.
3. At the command prompt, set the JAVA_HOME environment variable to the directory of your local JDK.

Use a command in the following format, where JAVA_LOCATION is the directory of your local JDK:

*On Windows:*

```
SET JAVA_HOME=JAVA_LOCATION
```

*On Unix or Linux®:*

```
JAVA_HOME=JAVA_LOCATION
export JAVA_HOME
```

4. At the command prompt, execute the command to compile the sample.

   Use a command in the following format:

   *On Windows:*

   ```
   %ANT_HOME%\bin\Ant -f build.xml dist
   ```

   *On Unix or Linux:*

   ```
   $ANT_HOME\bin]Ant -f build.xml dist
   ```

   The compile command generates the jar file restwsclientexample.jar.

## Running Example 1

1. To run example 1, verify that the TargetExample parameter in the application.properties file is set to example1.
2. In the Example 1 Properties section of the properties file, substitute the full path of capture.txt (see step 2 on page 50) for RemoteFileLocation.
3. At the command prompt on the machine where you will run the example, set the JAVA_HOME environment variable to the directory of your local JRE.

   Use a command in the following format, where *JAVA_LOCATION* is the directory of your local JRE:

   *On Windows:*

   ```
   SET JAVA_HOME=JAVA_LOCATION
   ```

   *On UNIX or Linux:*

   ```
   #!/bin/sh
   JAVA_HOME=JAVA_LOCATION
   export JAVA_HOME
   ```

4. Change to the RESTWebServices\dist directory, where run.bat and run.sh are located.
5. Execute run.bat or run.sh as follows:

   `run.bat application.properties` or
   `run.sh application.properties`

6. Review the results of the run in the Assure DQ user interface.

### Running the Other Examples

To run example 2 or example 3, change the TargetExample parameter in the application.properties file to the correct example number.

# Request–Response Operations (Methods)

The Web service provides two methods: ExecuteControlPoint and ExecuteControlPoint/Result in /casapi/services.

**ExecuteControlPoint** executes a control point. The method offers multiple options for process invocation and responds with errors, a result, or a cookie. Either data capture sources can be specified or data can be passed to Assure DQ via the request. The method supports both synchronous and asynchronous processing.

This method has the following two types of messages:

- **ControlPointRequest**—Input message used to call ProcessControlPoint. See page 54 for more details.
- **ControlPointResponse**—Output message returned when calling ProcessControlPoint. See page 57 for more details.

**ExecuteControlPoint/Result** is used to access processing results in asynchronous mode and when a request times out. When a request has been processed successfully or times out, Assure DQ will return a cookie for accessing the results. This method has the following two types of messages:

- **ControlPointResultRequest**—Input message used to call GetControlPointResult. Requires a cookie ID from an asynchronous call or from a timed-out request. See page 59 for more details.
- **ControlPointResultResponse**—Output message returned when calling GetControlPointResult. If the control point generated results, the message will include a result code and result catalog ID. Otherwise, it will include a message indicating results do not exist. See page 59 for more details.

# Authentication and Access

By default, basic authentication is enabled for the Assure DQ Web Services. This means that when a client program calls the Web Service, the system authenticates the user ID and password against the security directory specified for Assure DQ. In addition, the user ID must have one of the following Assure DQ roles:

- Superuser
- Remote Executor

HTTP basic authentication is implemented for WildFly and WebSphere deployments. Web Services Security (WSS) is not implemented.

The Web Services security settings are in the IA.properties file located in *infogix.install.home*\Infogix\config*config.name*\IA. If you do *not* want Assure DQ to authenticate Web service calls, set the WS-AUTH_TYPE property to none and redeploy Assure DQ

## Access

| Assure DQ REST Web Service for Control Point Execution Properties | |
|---|---|
| Endpoint URL | http://<host:port>/casapi/services/ExecuteControlPoint |
| Content Type | application/json |
| EndpointURL | http://<host:port>/casapi/services/ExecuteControlPoint/Result |
| Content Type | application/json |

# Web Service Element Dictionary

This section describes the messages exchanged between the client and the Web service, and their data elements.

## ControlPointRequest Message

| ControlPointRequest | | | |
|---|---|---|---|
| **Attribute** | **Use** | **Type** | **Description** |
| controlEntityName | Required | String | The name of the control entity. |
| controlPointName | Required | String | The name of the control point within the control entity to process. |
| controlEntityId | Optional | String | Unique ID for the request and control entity instance. Used for internal queuing of requests and run-to-run identification for sequencing control points. |

| ControlPointRequest | | | |
|---|---|---|---|
| **Attribute** | **Use** | **Type** | **Description** |
| controlReportsLimit | Optional | int | Can be used to override the system and control point settings for the maximum number of Control Reports to output. To permit unlimited reports, specify -1. |
| cycleNoOverride | Optional | Long | Can be used to override the cycle number specified for a control entity. Can also be used with the rerun attribute to process a specific run again. |
| developmentMode | Optional | Boolean | Reserved for future use. Defaults to false. |
| rerun | Optional | Boolean | Indicates if this execution is a rerun. For reruns, control data and findings are deleted and replaced in the database. Defaults to false. |
| resultsLimit | Optional | int | Can be used to override the system and control point settings for the maximum number of results to output. To permit unlimited results, specify -1. |
| runNoOverride | Optional | Long | Can be used to override the run number specified for a control entity. Can also be used with the rerun attribute to process a specific run again. |
| sourceTimestamp | Required | dateTime | Set to the client's current timestamp.Reserved for future use. |
| timeout | Optional | int | Can be used to override how long, in milliseconds, a synchronous call should wait for a response. If a response takes longer, Assure DQ will return a cookie for delayed results checking. Defaults to 10000 milliseconds. |
| waitForResponse | Optional | Boolean | Can be used to override the default processing mode, which is asynchronous. When set to true (synchronous mode), Assure DQ will process the request and then return the processing result, a timeout cookie, or a processing error. In asynchronous mode, Assure DQ will return a cookie (reference) string or a processing error once the request has been queued. Defaults to false (asynchronous mode). |

| ControlPointRequest | | | |
|---|---|---|---|
| **Element** | **Nillable** | **Type** | **Description** |
| ctlrptlmt | True | int | An integer specifying the Control Reports limit for the current control point run. To permit unlimited reports, specify -1. |
| rslmt | True | int | An integer specifying the results limit for the current control point run. To permit unlimited results, specify -1. |
| Specify **only one** of the following four elements: | | | |
| binaryDataItem | True | base64binary | Used when capturing records containing binary data encoded as base64. |
| textDataItem | True | String | Used when capturing records containing text data. |
| captureSourceList | True | CaptureSourceListType–An unbounded sequence of CaptureSourceType elements (see description, below). | Used when capturing data from files in order to override default URLs. List named capture sources and their URL overrides. Specify URLs from the server's context, not that of the calling client. |
| controlValueList | True | ControlValueListType–An unbounded sequence of ControlValueType elements (see description on page 57). | Used when directly inputting data. List name value pairs to populate the control fields for this control entity. |

## CaptureSourceType Element

This element type specifies the name and URL of the data capture source.

| CaptureSourceType | | | |
|---|---|---|---|
| **Attribute** | **Use** | **Type** | **Description** |
| sourceName | Required | String | The name of the capture source, as specified for the control point. |
| sourceUrl | Required | String | The URL of the capture source. Can be the path to a file on the server or a URL. For supported URL formats, please see the Assure DQ online help for data capture sources. |

## ControlValueType Element

This element type specifies control values as name value pairs.

| ControlValueType | | | |
|---|---|---|---|
| **Attribute** | **Use** | **Type** | **Description** |
| name | Required | String | The name of the control field to pass to Assure DQ. Use the exact name specified in the control entity definitions. |
| value | Required | String | The actual value to pass to Assure DQ. The value is passed as a string and will inherit the type and format specified for the control field in the control entity definitions. |

## ControlPointResponse Message

| ControlPointResponse | | | |
|---|---|---|---|
| **Element** | **Nillable** | **Type** | **Description** |
| result | True | ControlPointResultType–See page 58. | The results of the control point processing, if processing executed successfully. |

| ControlPointResponse | | | |
|---|---|---|---|
| **Element** | **Nillable** | **Type** | **Description** |
| resultCookie | True | String | A reference ID used to get results in asynchronous mode or when a timeout has occurred. This reference ID can be used with a call to GetControlPointResult. |
| error | True | ProcessingErrorType– An unbounded sequence of error messages in the form of string elements. | If errors are encountered during processing of the control point, will contain a list of error messages. Note that these are not the same as results; these are execution errors. |

## ControlPointResultType Element

This element type specifies the processing return code and catalog ID for a completed control point processing request.

| ControlPointResultType | | | |
|---|---|---|---|
| **Attribute** | **Use** | **Type** | **Description** |
| returnCode | Required | int | The return code of the control point execution as defined in validation results. |
| catalogId | Optional | String | The result catalog ID of the result catalog, if one was produced. By default, control points invoked via the Web service do not produce result catalogs. However, if a pass/fail/complete validation outcome occurs and the Result or Generate Report option is specified for that outcome, the control point will produce a result catalog. |
| instanceCount | Optional | int | The number of instances captured by the control point. For single capture control points, the instance count is 0 or 1. For multiple capture control points, the count is 0 or a positive integer. |

### ControlPointResultRequest Message

| ControlPointResultRequest | | | |
|---|---|---|---|
| **Attribute** | **Use** | **Type** | **Description** |
| resultCookie | Required | String | The reference ID that identifies a request that was made successfully in asynchronous mode or a request that timed out. |

### ControlPointResultResponse Message

| ControlPointResultResponse | | | |
|---|---|---|---|
| **Element** | **Nillable** | **Type** | **Description** |
| result | Yes | ControlPointResultType | The results of control point processing, if results were produced. |
| error | Yes | ProcessingErrorType | A message indicating results do not exist. |

# Troubleshooting

## Too Many Synchronous Calls

Too many synchronous calls waiting for a response at the server can result in denial of service to client programs. When this happens, no further calls are processed.

To prevent this problem, by default, no more than 20 calls can wait for a response at the server and the maximum wait for response time is five minutes. This overrides longer wait times specified in client programs. Additional calls are still processed, but cannot consume resources by waiting at the server.

To change these default settings, open *install_folder*\Infogix\components\ControlsAssure System\conf \api.properties and edit the MAX_CONCURRENT_WAITING _SYNC_CALLS property and/or the MAX_WAITFOR_RESPONSE _TIME_MILLIS property, as needed. If your calls typically have long wait times, consider changing to asynchronous processing.

## Timeout Values Too Low

Assure DQ provides the ability to set up timeout settings for API processing at three levels.

To change the default settings, edit *install_folder*\Infogix\components \ControlsAssure System\conf\api.properties.

### System Maximum Timeout

The system times out when calls take longer than the max wait for response time, which defaults to five minutes (300,000 milliseconds). The max wait for response time is a system-wide limit on timeout duration that takes precedence over any other parameter. The property that controls this behavior is MAX_WAITFOR_RESPONSE_TIME_MILLIS.

### System Default When Client API Does Not Set Timeouts

The system times out when calls take longer than the system wait for response time, which defaults to 10 seconds (10,000 milliseconds). This parameter is overridden by the max wait for response parameter and by the timeout value specified in the control point request message. This property should always be less than or equal to the max wait for response time. The property that controls this behavior is SYSTEM_WAITFOR_RESPONSE_TIME_MILLIS.

### API Control Point Request Message

The client API can set the timeout property for each control point request message. This attribute value takes precedence over the system wait for response time parameter. For more information, refer to the description of the “timeout” attribute in the Webservice Element Dictionary on page 55.

### Backward Compatibility

For backward compatibility to previous versions of the API:

- Set MAX_WAITFOR_RESPONSE_TIME_MILLIS to be the maximum time any client call should wait for a response. It is not possible to set an infinite time.
- Set SYSTEM_WAITFOR_RESPONSE_TIME_MILLIS to be less than the value set for MAX_WAITFOR_RESPONSE_TIME_MILLIS.

# 8

# REST Web Service for Control Point Results

The Assure DQ server includes a Web Service that enables a calling client program to request processing results for any previous control runs.

To use the Web Service, add code to an existing application or create a new client program to call the Web Service. The Web Service supports both Java and .NET clients.

The *API Guide and Examples* CD includes a sample Java client program. The sample demonstrates how to access and display control point processing results given a result catalog ID.

The sample Java client, source code for the client, and libraries and executables necessary to build, compile, and run the client are in the WebServices folder on the CD. The folder also contains a WSDL (Web Services Description Language) file and programmer documentation for the Web Service, and sample data and control point rules to use with the sample client.

The table below describes the contents of the restwsclientexample folder.

| Contents of restwsclientexample directory | |
|---|---|
| dist | Contains the configuration file, batch and shell script files, and jar file needed to run the sample client. The jar is already built. To run the sample client, the only requirement is a Java runtime environment. See "Running the Sample Client" on page 62. |
| lib | Contains all third party jars necessary to build and run the sample client. |
| samples | Contains a properties file template, zip files of the Assure DQ rules for the control entities/control points referenced in the sample, and a sample input file. |
| schema | Contains a copy of the WSDL for the Web service, for reference and for builds. |
| src | Contains source code for the sample Java client. |
| build.xml | Contains an Ant build file to generate the required classes and compile and jar the sample client. |
| jaxws-ri-2.3.2.zip | Contains JAX-WS RI 2.3.2. JAX-WS RI 2.3.2 is a Web Services framework that provides tools and infrastructure to develop Web Services solutions for the end users and middleware developers. Unzip this in the current folder to create the jaxws-ri folder. |

# Running the Sample Client

If you would like to experiment with the sample Java client shipped with Assure DQ, complete the following steps:

1. In the Assure DQ user interface, select Results > By Control Point.
   The Results by Control Point list displays.
2. Select a processing result from the list.
   On the Summary tab in the bottom pane, look for the Catalog ID of the result.
3. Run the example results script in the dist directory:
   `results application.properties catalog_ID`

## Modifying and Compiling the Sample Client

If you modify the sample Java client, you must compile the sample before running it. The Ant software must be installed on the server or a development machine. To compile the sample, complete the following steps:

1. At the command prompt on the machine where you will run the example, change to the folder where you installed \examples \restwsclientexample\dist.
2. At the command prompt, set the ANT_HOME environment variable to the directory of your Ant home folder.
3. At the command prompt, set the JAVA_HOME environment variable to the directory of your local JDK.
   Use a command in the following format, where JAVA_LOCATION is the directory of your local JDK:

   *On Windows:*

   ```
   SET JAVA_HOME=JAVA_LOCATION
   ```

   *On UNIX or Linux®:*

   ```
   JAVA_HOME=JAVA_LOCATION
   export JAVA_HOME
   ```
4. At the command prompt, execute the command to compile the sample.
   Use a command in the following format:

   *On Windows:*

   ```
   %ANT_HOME%\bin\Ant -f build.xml dist
   ```

   *On UNIX or Linux:*

   ```
   $ANT_HOME\bin]Ant -f build.xml dist
   ```

   The compile command generates the jar file, restwsclientexample.jar.

# Request–Response Operations (Methods)

The Web Service provides three methods within /casapi/services/ GetControlPointResult: catalogSummary, listResults and listReports.

**catalogSummary** takes a result catalog ID and returns a response containing summary-level information about the processing results.

**listResults** returns the detailed results for a result catalog. Your client program can page through these results. The client must pass three parameters: the result catalog ID, the starting index number for the result list, and the number of detailed results to return.

**listReports** takes a result catalog ID and returns a response listing the reports produced for a result catalog.

# Authentication and Access

By default, basic authentication is enabled for the Assure DQ Web Services. This means that when a client program calls the Web Service, the system authenticates the user ID and password against the security directory specified for Assure DQ. In addition, the user ID must have one of the following Assure DQ roles:

- Superuser
- Remote Executor

HTTP basic authentication is implemented for WildFly and WebSphere deployments. Web Services Security (WSS) is not implemented.

The Web Services security settings are in the IA.properties file located in *infogix.install.home*\Infogix\config*config.name*\IA. If you do *not* want Assure DQ to authenticate Web service calls, set the WS-AUTH_TYPE property to none and redeploy Assure DQ.

## Access

| Assure DQ REST Web Service For Control Point Results Properties | |
|---|---|
| Endpoint URL | http://<host:port>/casapi/services/GetControlPointResult/catalogSummary/{catalogId} |
| Accept | application/json or application/xml |
| Endpoint URL | http://<host:port>/casapi/services/GetControlPointResult/listResults/?catalogId=n&offset=0&limit=100 |
| Accept | application/json or application/xml |
| Endpoint URL | http://<host:port>/casapi/services/GetControlPointResult/listReports/{catalogId} |
| Accept | application/json or application/xml |

# A

# Assure DQ Message Schema

The following are sample request (ex) and optional response (rs) messages that demonstrate the XML schema for sending and receiving messages from Assure DQ.

The sample is configured to wait for a response—the w (wait) parameter is set to true in the request message:

```
<w>true</w>
```

If you prefer, you can request a response only when a control produces a non-zero return code or result. In the request message, substitute the re (respond on error) parameter for the w (wait) parameter:

```
<re>true</re>
```

## Sample Request

```
<ex>
      <cp>
            <e>SAMPLE MESSAGE CE</e>
            <p>CP1</p>
            <w>true</w>
            <cb>RETURN TO ME 1</cb>
            <trec>IASAMP01 DEFAULT;A;100;A;100;</trec>
      </cp>
</ex>
```

## Sample Response

```
<rs>
      <cp>
            <e>SAMPLE MESSAGE CE</e>
            <p>CP1</p>
            <rr>false</rr>
            <w>true</w>
            <t>2007-05-22T12:13:25.970-05:00</t>
            <cb>RETURN TO ME 1</cb>
            <trec>IASAMP01 DEFAULT;A;100;A;100;</trec>
      </cp>
      <r>
            <rc>0</rc>
      </r>
</rs>
```

A schema for the XML request and response messages expected by Assure DQ is in message.xsd in both the Java and COBOL folders on the CD. This schema applies to both types of messages. The tables in this appendix list and describe the elements of these messages.

## Execute API Element ex

| ExecuteAPI | | | |
|---|---|---|---|
| **Children** | **Use** | **Type** | **Description** |
| Control Point Request Element cp | Required | String | Execute control point. |
| Result Query Element qry | Optional | String | Query control point execution result from previous asynchronous call. |

## Control Point Response Element rs

| ControlPointResponseType | | | |
|---|---|---|---|
| **Children** | **Use** | **Type** | **Description** |
| Control Point Request Element cp | Required | String | Execute control point. |
| Control Point Result Element r | Optional | ControlPoint Result | Results from synchronous processing. |
| Processing Error Element e | Optional | Processing Error | Error messages. |

## Control Point Request Element cp

| ControlPointRequestType | | | |
|---|---|---|---|
| **Children** | **Use** | **Type** | **Description** |
| e | Required | String | Entity name |
| p | Required | String | Control point name |
| rr | Optional | Boolean | Rerun indicator |
| cy | Optional | Long | Cycle number override |
| rn | Optional | Long | Run number override |

| ControlPointRequestType | | | |
|---|---|---|---|
| **Children** | **Use** | **Type** | **Description** |
| rslmt | Optional | Integer | Results limit override |
| ctlrptlmt | Optional | Integer | Control Reports limit override |
| id | Required | String | Transaction identifier |
| w | Optional | Boolean | Wait for response |
| re | Optional | Boolean | Only send a response if an error occurs (that is, if a control returns a non-zero return code or result). The system does not send a response when capture processing summaries and other system-generated informational results are produced. |
| to | Optional | Integer | Timeout if wait for response |
| k | Required | String | Cookie key used to retrieve results in asynchronous mode |
| t | Optional | Date Time | Source timestamp |
| d | Optional | Boolean | Development mode |
| Runtime Field Value Elements cv and rt | Optional | RunTime ValueType | Runtime field values |
| cb | Optional | String | Optional callback data to be sent back as part of the response message |
| brec | Optional | Base 64 Binary | Binary record |
| trec | Optional | String | Text record |
| Capture Source List Element s | Optional | Capture SourceType | Capture sources URL overrides |
| cv | Optional | Control ValueType | Control field name and value |

## Runtime Field Value Elements cv and rt

| RunTimeValueType | | | |
|---|---|---|---|
| **Children** | **Use** | **Type** | **Description** |
| n | Required | String | Runtime field name |
| v | Required | String | Runtime field value |

## Processing Error Element e

| ProcessingErrorType | | | |
|---|---|---|---|
| **Element** | **Use** | **Type** | **Description** |
| m | Required | String | Error message |

## Control Point Result Element r

| ControlPointResultType | | | |
|---|---|---|---|
| **Children** | **Use** | **Type** | **Description** |
| rc | Required | Integer | Return code |
| cat | Optional | String | Result catalog ID |
| ic | Optional | Integer | Instance count |

## Capture Source List Element s

| CaptureSourceListType | | | |
|---|---|---|---|
| **Children** | **Use** | **Type** | **Description** |
| n | Required | String | Capture source name |
| url | Required | String | Capture source URL |

## Result Query Element qry

| ControlPointResultRequestType | | | |
|---|---|---|---|
| **Children** | **Use** | **Type** | **Description** |
| k | String | ControlPointResult Request | Cookie key for retrieving control point execution results from a previous asynchronous call |

# Index

## W



## X