Merative Social Program Management 8.1

Cúram WBPM Integration Cookbook

Note

Before using this information and the product it supports, read the information in Notices on page 25

Edition

This edition applies to Merative™ Social Program Management 8.0.0, 8.0.1, 8.0.2, 8.0.3, and 8.1.

Contents

1 Integrating with IBM WebSphere Business Process Management

Use this information learn how to connect WebSphere® Business Process Management to a Cúram inbound web service by using an existing Cúram web service as an example. Instructions on how to start a WebSphere® Business Process Management web service with a Cúram outbound web service are also provided.

This document is a practical handbook for developers to integrate Cúram web services with IBM® 's WebSphere® Business Process Management (WBPM) family of products. The document provides step by step instructions on how to connect WBPM to a Cúram Inbound web service using an existing Cúram web service as an example. It also provides instructions on how to invoke a WBPM web service using a Cúram Outbound web service. It is intended that the document be used in conjunction with the *Cúram Server Modelling Guide*.

The document is intended for developers using WebSphere Integration Developer (WID) and WebSphere Process Server (WPS) to choreograph Cúram web services, i.e., to develop WID calls to Cúram web services or Cúram web service calls to WID.

1.1 Background

WBPM is an IBM family of products for modeling, assembling, deploying, and managing applications that embody a service-oriented architecture (SOA) and are integrated using an enterprise service bus (ESB). The core set of products in the solution are:

- WebSphere Process Server (WPS) deploys and executes processes that orchestrate services (people, information, systems, and trading partners) within your service-oriented architecture (SOA) or non-SOA infrastructure.
- WebSphere Enterprise Service Bus (WSESB) provides Web services connectivity and JMS messaging.
- WebSphere Integration Developer (WID) is an Eclipse-based tool for building SOA-based BPM and integration solutions across WPS, WESB, and WebSphere Adapters.

Cúram currently offers a set of web services called "Incremental Modernization and Transformation" (IMT) Services to weave together Cúram's SEM (Social Enterprise Management) services which are embedded in Cúram's various business modules (such as Case management, Eligibility and Entitlement Management, etc.)

Cúram have successfully verified that WBPM (WID versions 6.1.2.002 and 6.2) can be used to connect applications together using IMT Services. This involved the following steps:

- Setup of WID and WPS alongside a Cúram installation.
- Import Cúram IMT Services into WID and WPS.
- Connect Cúram IMT Services to other services in WPS.
- Test the import and the connection. Analyse and validate the messages being sent between the various components.

This demonstrates that Cúram supports SOA and is ready for integration with external systems using WebSphere BPM stack. This document outlines the steps taken to verify this proof of concept.

1.2 Prerequisites

To make best use of this guide, you should have a good knowledge of the Cúram web service infrastructure. Ideally, readers should be familiar with chapters 28 through 30 of the *Cúram Server Modelling Guide* and *Common Practices for a Web Service Implementation* before embarking upon WBPM integration.

It is assumed that before embarking on the steps provided in this document, the developer has an installed Cúram, CuramWS, WID and WPS. It is also assumed that the developer has a Cúram development environment set up. It is outside the scope of this document to define the process of creating a Cúram inbound or Cúram outbound web service.

1.3 Creating an Import in WID

Introduction

This chapter defines the steps involved to import the CreateEvidence IMT web service into WID. This provides an example of how to develop a WID call to a Cúram web service.

Steps

Create Project and Dependant Library

1. Select File > New > Project > Module and call it "CuramCreateEvidence"
2. Select File > New > Library and call it "CuramWebServices"
3. Expand the CuramCreateEvidence module and open the Dependencies
4. Click Add and select the CuramWebService library
5. Tick "Deploy with Module"

Import Cúram WSDL

1. Select Physical Resources > CuramWebServices > Properties and note the path to the CuramWebServices library folder
2. Browse to http://localhost:<port>/CúramWS/services/CreateEvidenceWS?wsdl and save the file as CreateEvidenceWS.wsdl in the CuramWebServices library folder
3. Refresh the CuramWebService library to see the WSDL file

Prepare WSDL for Client Generation

1. Right click CreateEvidenceWS.wsdl and open with text editor
2. Update these imports with prefixes if WID shows errors. e.g.

```
            <wsdl:types>
<xs:schema...
 <
             xs:
            import namespace="http://lang.java"/>
```

```
<
        xs:
      import namespace="http://exception.util.curam"/>
<
        xs:
      import namespace="http://message.util.curam"/>
<
        xs:
      import namespace="CreateEvidenceWS"/>
```

Add Message Monitoring

1. Find the service port binding within the WSDL and change the port on the address, e.g. from 10102 to 8070

   ```
   <wsdlsoap:address location="http://localhost:8070/CuramWS/
   services/CreateEvidenceWS"/>
   ```
2. Set up TCPMon to forward messages arriving at the new port number on to the Cúram port number (you will see the response message too). Open the Java Applet at https://tcpmon.dev.java.net/tcpmon.jnlp and enter Local Port as 8070, Server Name as localhost and Server Port as 10102.

Decouple Types From Interface

Refactoring the WSDL types into their own, decoupled, definition files allows the developer to reuse these types more easily within WID.

1. Goto Business Integration view and expand CuramWebServices > Interfaces
2. Right click CreateEvidenceWS and select Refactor > Extract WSDL Components, checking the Extaract Business Objects and Use folder structure derived from the Business Object namespace(s) check boxes. Do not select the Separate Port Type (Interface) from other WSDL elements checkbox.
3. If you get a warning about type name collisions, just rename one of the files in the preview box.
4. Create a schema called CreateEvidenceRootType.xsd to define a root node that can contain either the top level request element or the top level response element. Make sure your include statements have the correct path to the xsd files you just extracted:

```
            <?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:ce="http://ws.curam/EvidenceCreateWS"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  attributeFormDefault="unqualified"
  elementFormDefault="qualified"
  targetNamespace="http://ws.curam/EvidenceCreateWS"
  xmlns=""
>
  <xs:include
    schemaLocation=
 "
             curam\ws\EvidenceCreateWS
\EvidenceCreateEvidenceType.xsd
           " />
  <xs:include
    schemaLocation=
    "
```

```
                curam\ws\EvidenceCreateWS
\EvidenceCreateResponse.xsd
              " />
  <xs:complexType name="CreateEvidenceRootType">
      <xs:sequence>
        <xs:element name="evidence"
          type="ce:EvidenceCreateEvidenceType" minOccurs="0"/>
        <xs:element name="response"
          type="ce:EvidenceCreateResponse" minOccurs="0"/>
      </xs:sequence>
  </xs:complexType>
  </xs:schema>
```

5. Refresh the CuramWebService library to see CreateEvidenceRootType.xsd

Wrap Messages

1. Double click the CreateEvidenceWS interface.
2. Click on both instances of anyType in the types column, click Browse and change the type to CreateEvidenceRootType.

Create the Import

1. Open the assembly diagram
2. Drag CreateEvidenceWS interface onto this assembly
3. Select "Import with Web Service binding". Browse for an existing port and select the CreateEvidenceWS port.
4. Select "SOAP1.1/HTTP" and click OK.
5. Save the assembly diagram.

1.4 Testing an Import in WID

Introduction

This chapter provides instructions on how to test the import using WID's own Business Process Choreographer Explorer.

Steps

Create and Wire a Test Process

1. In the assembly diagram, drag and drop a Process from the palette beside your import
2. Rename the Process to something like Test_CreateEvidenceWSImport1
3. Hover over the end of the Process box and an orange wiring line should appear. Hold down the left mouse button to connect this to the import.
4. Hover over the process and add the CreateEvidenceWS interface by selecting the interface button that appears.

Implement Test Process

1. Double click the process and click Yes to implement it. Save it in a useful location such as curam/bpi
2. Click on Invoke in the palette and drop it between the Receive and the Reply
3. Select the Properties tab and the Details page within this.
4. Select the CreateEvidenceWSPartner
5. Select the input and output to "part" and "receiveDocumentResponse" in this case, i.e. to match the Receive and Reply presets which come from the Cúram WSDL.
6. Save everything.

Launch the Server

1. Go to the Servers tab in the bottom pane.
2. Right click on WebSphere Process Server and select Add and Remove Projects. Add our module and click Finish.
3. Start the server. This will take around 5 minutes the first time you do it. It's started when you see "Application started: CuramCreateEvidenceApp" in the console

Run the Test Process

1. Right click the server and select Launch > WebSphere Business Choreographer Explorer
2. Click Yes
3. Logon using admin/admin
4. Click on My Process Templates on the top left
5. Tick your process and click Start Instance
6. Click Edit Source and paste in a valid XML request.

> **Note:** In this instance, caseID, a participantID and a caseParticipantRoleID are required in order to create evidence. These IDs must be valid and on the Cúram database. At this point, you should create an empty Sport Grant Sample Case using James Smith and note these three IDs to use in your request document.

```
<?xml version="1.0" encoding="UTF-8"?>
<p:CreateEvidenceRootType
  xsi:type="p:CreateEvidenceRootType"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:p="http://ws.curam/EvidenceCreateWS">
  <p:evidence
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://ws.curam/EvidenceCreateWS">
    <p:evidenceData>
      <p:evidenceDetails>
        <p:caseID>83237780112875192 32</p:caseID>
        <p:evidenceType>ET500</p:evidenceType>
        <p:receivedDate>2009-02-23</p:receivedDate>
        <p:effectiveDate>2009-02-23</p:effectiveDate>
        <p:participantID>101</p:participantID>
      </p:evidenceDetails>
      <p:dataObjects>
        <p:dataItem
          name="caseParticipantRoleID">-106960491150049280
        </p:dataItem>
```

```
          <p:dataItem
            name="sportingActivityType">-SA1
          </p:dataItem>
          <p:dataItem
            name="sportingAwardType">SAT1
          </p:dataItem>
          <p:dataItem
            name="paymentAmount">1
          </p:dataItem>
        </p:dataObjects>
      </p:evidenceData>
    </p:evidence>
  </p:CreateEvidenceRootType>
```

7. Click Validate if you want to ensure it's a valid message
8. Click Submit

Analyze Results

1. If everything is successful you will see well formed process input and output messages in the form view.
2. View Source on the output and you will see that it matches the input pasted in 3.2.4:

```
<?xml version="1.0" encoding="UTF-8"?>
<p:CreateEvidenceRootType
  xsi:type="p:CreateEvidenceRootType"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:p="http://ws.curam/EvidenceCreateWS">
  <p:response>
    <p:evidenceCreate success="true">
      <p:evidenceID>-611363649415544832</p:evidenceID>
      <p:evidenceType>ET500</p:evidenceType>
      <p:evidenceDescriptorID>-7600950271094554624
      </p:evidenceDescriptorID>
    </p:evidenceCreate>
  </p:response>
</p:CreateEvidenceRootType>
```

3. Look at TCPMon and you should see the latest message going out and back. The request consists of a SOAP envelope who's body contains the process input. It looks slightly altered as the namespace alias is updated. The response is also a SOAP envelope, who's body contains the process output. Once again, the namespace alias may look different but this is to be expected.
4. If you want further insight into what happens under the hood of Cúram's web service infrastructure, you can add some debugging output to your Cúram inbound service BPO to print out the request that arrives in and the response that gets returned. e.g.

```
<?xml version="1.0" encoding="UTF-8"?>
<ws_1:evidence>
  <ws_1:evidenceData>
    <ws_1:evidenceDetails>
      <ws_1:caseID>8323778011287519232
      </ws_1:caseID>
      <ws_1:evidenceType>ET500
      </ws_1:evidenceType>
      <ws_1:receivedDate>2009-02-23
      </ws_1:receivedDate>
      <ws_1:effectiveDate>2009-02-23
      </ws_1:effectiveDate>
```

```
        <ws_1:participantID>101
        </ws_1:participantID>
      </ws_1:evidenceDetails>
      <ws_1:dataObjects>
        <ws_1:dataItem
          name="caseParticipantRoleID">-106960491150049280
        </ws_1:dataItem>
        <ws_1:dataItem
          name="sportingActivityType">-SA1
        </ws_1:dataItem>
        <ws_1:dataItem
          name="sportingAwardType">SAT1
        </ws_1:dataItem>
        <ws_1:dataItem
          name="paymentAmount">1
        </ws_1:dataItem>
      </ws_1:dataObjects>
    </ws_1:evidenceData>
  </ws_1:evidence>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <evidenceCreate success="true">
    <evidenceID>-611363649415544832</evidenceID>
    <evidenceType>ET500</evidenceType>
    <evidenceDescriptorID>-7600950271094554624
    </evidenceDescriptorID>
  </evidenceCreate>
</response>
```

1.5 Creating an Export and a Mapping in WID

Introduction

This chapter describes how to expose a WID web service to a Cúram client. It also provides a simple example of mapping the WID service. The mapping component allows customers to invoke their own logic, e.g., to kick off workflow web services, etc. In the context of this example the mapping will simply pass the unaltered request directly on to the import.

Steps

Create and Implement an Export

1. Open the Assembly Diagram and drop an Export onto it from the palette.
2. Add a new interface to it called SimpleWIDCreateEvidenceService
3. On the interface definition screen, click the icon for Add Request Response Operation
4. You can leave the operation name and request and response node names as the default, but you will need to change the type from string to CreateEvidenceRootType
5. Save the interface and go back to the assembly and save it too.

Create and Implement Mapping Component

1. Drop an Interface Map onto the assembly from the palette.
2. Wire the export to the map and wire the map to the import with in a 1:1 relationship
3. Double click the map to implement it.
4. Save it to curam/maps.
5. Wire the export's operation to the import's operation.
6. Now the parameter mappings should appear. If they don't, just click on the wire.
7. Wire the parameters with a Move operation from input1 in the export operation to part in the inport operation and another Move operation from the receiveDocumentReturn parameter to the output1 parameter in the export operation.
8. Save this and save the assembly.

Generate the WSDL

1. Right click the export in the assembly and select Generate Binding > Web Service Binding > SOAP/HTTP.
2. Save everything.

Add Message Monitoring

1. Selecting the export, go to the properties tab and click on Binding. The port number is usually defaulted to 9080.
2. Browse to the generated WSDL file, CuramCreateEvidence_SimpleWIDCreateEvidenceExport.wsdl and copy it to CuramAppBuild\EJBServer\components\core\wsdl along with all the other schema files that it is dependant upon.
3. Edit the port number on the Cúram version of this file from 9080 (for example to 1000) and add another monitor to TCPMon

Write a Cúram Outbound Service

Follow the Cúram Server Modelling Guide chapter on outbound web services.

Note: In brief, you must:

- Write a EJBServer\project\config\webservices_config.xml to point to this wsdl.
- Run the 'wsconnector' build target from EJBServer. This uses Axis2's WSDL2Java to generate a client application to consume the service hosted by WID.
- Add 'build\svr\wsc\jav' to your classpath as well as jars for commons-logging, commons-discovery and wsdl4j (specifically 1.5.1) from the CÚRAMSDEJ's lib directory.

The chapter also outlines how to call this client from within your regular code, but this may be a more useful example:

```
/**
 * This method tests WID's exposed Evidence export
 */
public void testCreateEvidenceViaWID()
  throws AppException, InformationalException,
  ParserConfigurationException, SAXException, IOException {

  boolean success = false;

  // The service locator gets instances of the web service.
  final
    WIDExport_WIDServiceHttpServiceLocator
    serviceLocator =
    new WIDExport_WIDServiceHttpServiceLocator();

  try {

    // get an instance of client stub
    // by casting from interface:
    final WIDExport_WIDServiceHttpBindingStub
      myService =
      (WIDExport_WIDServiceHttpBindingStub)
        serviceLocator
        .getWIDExport_WIDServiceHttpPort();

    // getRequestObject builds up the request using
    // Axis generated classes to represent the wsdl types
    final ResponseType response =
      myService.operation1(getRequestObject());

    // this evaluation of success is specific to the
    // structure of this particular service
    success =
      response.getResponse().getEvidenceCreate().isSuccess();

  } catch (javax.xml.rpc.ServiceException se) {
    System.out.println(se.getLocalizedMessage());
    System.out.println(se.fillInStackTrace());
  }

  assertTrue(success);

}
```

1.6 Message Flow in Completed Integration Example

Introduction

This chapter provides the messages from a working end to end integration example.

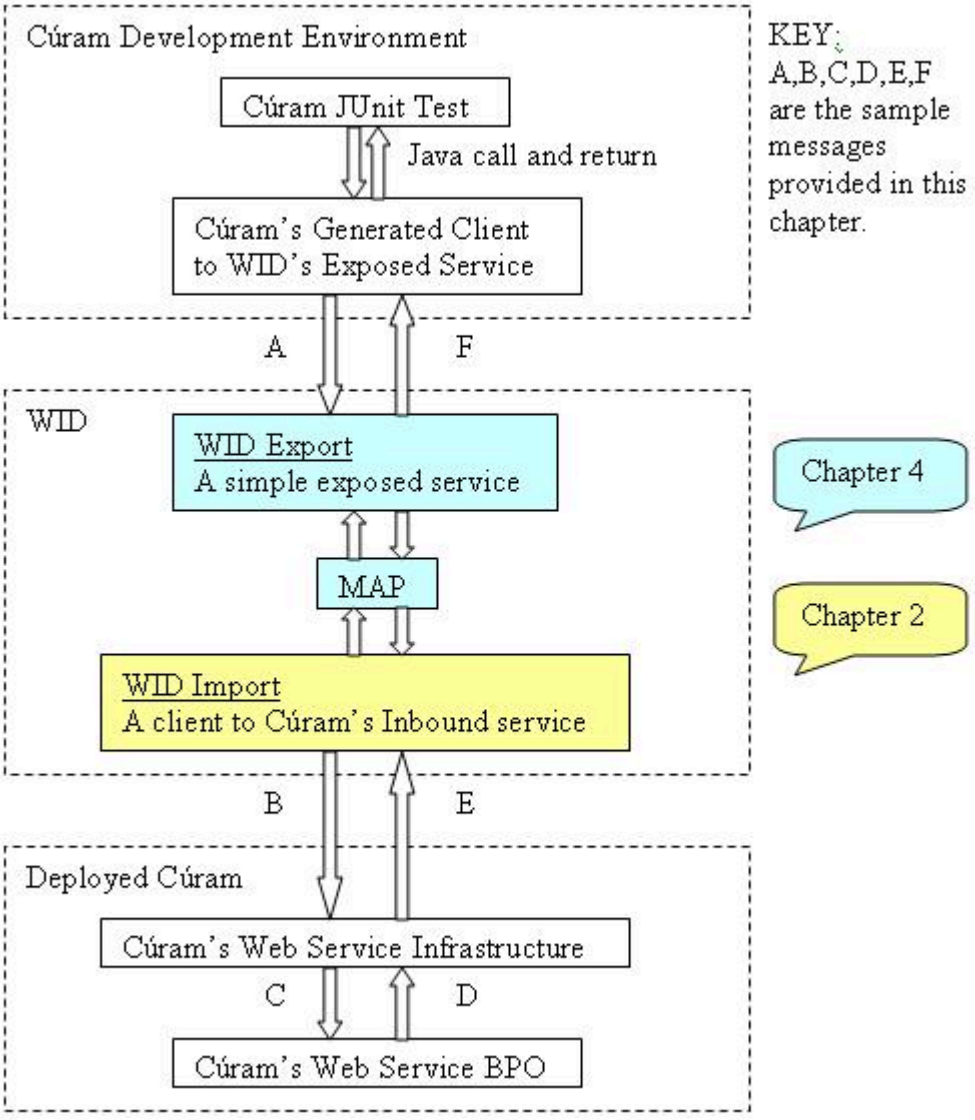

Figure 1: WID Integration Scenario

Messages

Cúram Outbound Client Calls WID Export (A)

This is an example of the message that gets sent when the Cúram outbound web service client invokes the exposed WID Export web service. Creating the JUnit test and the Cúram Outbound Client is outlined in Write a Cúram Outbound Service on page 16. Creating the WID Export is covered in Create and Implement an Export on page 15.

```
POST /CúramWS/services/CreateEvidenceWS HTTP/1.1
Host: localhost:8070
Accept: application/soap+xml,multipart/related,text/*
User-Agent: IBM WebServices/1.0
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Connection: Keep-Alive
Content-Type: text/xml; charset=utf-8
Content-Length: 1082
Date: Thu, 26 Feb 2009 10:28:25 GMT

<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
  <soapenv:Header />
  <soapenv:Body>
    <ws:CreateEvidenceRootType_element
 xmlns:ws="CreateEvidenceWS"
      xmlns:ws_1="http://ws.curam/EvidenceCreateWS"
    >
      <ws_1:evidence>
        <ws_1:evidenceData>
          <ws_1:evidenceDetails>
            <ws_1:caseID>5594596637100998656</ws_1:caseID>
            <ws_1:evidenceType>ET500</ws_1:evidenceType>
            <ws_1:receivedDate>2009-02-26
            </ws_1:receivedDate>
            <ws_1:effectiveDate>2009-02-26
            </ws_1:effectiveDate>
            <ws_1:participantID>8837188368807755776
            </ws_1:participantID>
          </ws_1:evidenceDetails>
          <ws_1:dataObjects>
            <ws_1:dataItem name="caseParticipantRoleID">0
            </ws_1:dataItem>
            <ws_1:dataItem name="sportingActivityType">SA2
            </ws_1:dataItem>
            <ws_1:dataItem name="sportingAwardType">SAT1
            </ws_1:dataItem>
            <ws_1:dataItem name="paymentAmount">50
            </ws_1:dataItem>
          </ws_1:dataObjects>
        </ws_1:evidenceData>
      </ws_1:evidence>
    </ws:CreateEvidenceRootType_element>
  </soapenv:Body>
</soapenv:Envelope>
```

WID Import calls Cúram Inbound Web Service (B)

This is an example of the message that gets sent when the WID Import (a web service client) invokes the Cúram Inbound web service. Creating the WID Import is covered in 1.3 Creating an

Import in WID on page 10. Creating the Cúram Inbound Web Service is outside the scope of this document.

```
POST
  /CuramCreateEvidenceWeb/sca/SimpleWIDCreateEvidenceExport
 HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept:
  application/soap+xml, application/dime, multipart/related,
 text/*
User-Agent: Axis/1.4
Host: localhost:1000
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 1036

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
 <soapenv:Body>
  <operation1
   xmlns="http://CuramCreateEvidence/
SimpleWIDCreateEvidenceService"
  >
    <input1 xmlns="">
      <ns1:evidence
        xmlns:ns1="http://ws.curam/EvidenceCreateWS"
      >
        <ns1:evidenceData>
          <ns1:evidenceDetails>
            <ns1:caseID>5594596637100998656</ns1:caseID>
            <ns1:evidenceType>ET500</ns1:evidenceType>
            <ns1:receivedDate>2009-02-26
            </ns1:receivedDate>
            <ns1:effectiveDate>2009-02-26
            </ns1:effectiveDate>
            <ns1:participantID>8837188368807755776
            </ns1:participantID>
          </ns1:evidenceDetails>
          <ns1:dataObjects>
            <ns1:dataItem name="caseParticipantRoleID">0
            </ns1:dataItem>
            <ns1:dataItem name="sportingActivityType">SA2
            </ns1:dataItem>
            <ns1:dataItem name="sportingAwardType">SAT1
            </ns1:dataItem>
            <ns1:dataItem name="paymentAmount">50
            </ns1:dataItem>
          </ns1:dataObjects>
        </ns1:evidenceData>
      </ns1:evidence>
    </input1>
  </operation1>
 </soapenv:Body>
```

```
</soapenv:Envelope>
```

XML request message arrives at Cúram Inbound BPO (C)

The Cúram Inbound Web Service infrastructure extracts a subset of the SOAP message that comes in over the wire. It passes this into the service 's BPO. Here is an example of this XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<ws_1:evidence>
  <ws_1:evidenceData>
    <ws_1:evidenceDetails>
      <ws_1:caseID>5594596637100998656</ws_1:caseID>
      <ws_1:evidenceType>ET500</ws_1:evidenceType>
      <ws_1:receivedDate>2009-02-26</ws_1:receivedDate>
      <ws_1:effectiveDate>2009-02-26</ws_1:effectiveDate>
      <ws_1:participantID>8837188368807755776
      </ws_1:participantID>
    </ws_1:evidenceDetails>
    <ws_1:dataObjects>
      <ws_1:dataItem name="caseParticipantRoleID">0
      </ws_1:dataItem>
      <ws_1:dataItem name="sportingActivityType">SA2
      </ws_1:dataItem>
      <ws_1:dataItem name="sportingAwardType">SAT1
      </ws_1:dataItem>
      <ws_1:dataItem name="paymentAmount">50
      </ws_1:dataItem>
    </ws_1:dataObjects>
  </ws_1:evidenceData>
</ws_1:evidence>
```

XML response message returned from Cúram Inbound BPO (D)

This is an example of the message that the Cúram service BPO returns to the Cúram Inbound Web Service infrastructure. The infrastructure will then convert this into a SOAP message and send it back to the client.

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <evidenceCreate success="true">
    <evidenceID>-36377825990085181144</evidenceID>
    <evidenceType>ET500</evidenceType>
    <evidenceDescriptorID>-5439222449956716544
    </evidenceDescriptorID>
  </evidenceCreate>
</response>
```

Cúram Inbound Web Service responds to WID Import request (E)

This is an example of the message that gets sent when the Cúram Inbound Web Service responds to the WID Import (web service client).

```
HTTP/1.0 200 OK
Content-Type: text/xml; charset=utf-8
Content-Language: en-IE
Content-Length: 720
Date: Thu, 26 Feb 2009 10:28:27 GMT
Server: WebSphere Application Server/6.1

<soapenv:Envelope
```

```
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
 <soapenv:Header />
 <soapenv:Body>
  <se:operation1Response
xmlns:se="http://CuramCreateEvidence/
SimpleWIDCreateEvidenceService"
   xmlns:ws="http://ws.curam/EvidenceCreateWS"
  >
    <output1>
      <ws:response>
        <ws:evidenceCreate success="true">
          <ws:evidenceID>-3637782599008518144
          </ws:evidenceID>
          <ws:evidenceType>ET500</ws:evidenceType>
          <ws:evidenceDescriptorID>-5439222449956716544
          </ws:evidenceDescriptorID>
        </ws:evidenceCreate>
      </ws:response>
    </output1>
   </se:operation1Response>
  </soapenv:Body>
</soapenv:Envelope>
```

WID Export responds to Cúram Outbound Client request (F)

This is an example of the message that gets sent when the WID Export responds to the Cúram Outbound Client request.

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Language: en-IE
Transfer-Encoding: chunked
Date: Thu, 26 Feb 2009 10:28:28 GMT
Server: WebSphere Application Server/6.1

26b
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
  <soapenv:Body>
    <ws:CreateEvidenceRootType_element
      xmlns:ws="CreateEvidenceWS"
      xmlns:ws_1="http://ws.curam/EvidenceCreateWS"
    >
      <response>
        <evidenceCreate success="true">
          <evidenceID>-36377825990085181144</evidenceID>
          <evidenceType>ET500</evidenceType>
          <evidenceDescriptorID>-5439222449956716544
          </evidenceDescriptorID>
        </evidenceCreate>
```

```
      </response>
    </ws:CreateEvidenceRootType_element>
  </soapenv:Body>
</soapenv:Envelope>
0
```

Notices

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the Merative website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of Merative

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of Merative.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

Merative reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by Merative, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

MERATIVE MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Merative or its licensors may have patents or pending patent applications covering subject matter described in this document. The furnishing of this documentation does not grant you any license to these patents.

Information concerning non-Merative products was obtained from the suppliers of those products, their published announcements or other publicly available sources. Merative has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-Merative products. Questions on the capabilities of non-Merative products should be addressed to the suppliers of those products.

Any references in this information to non-Merative websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this Merative product and use of those websites is at your own risk.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

The licensed program described in this document and all licensed material available for it are provided by Merative under terms of the Merative Client Agreement.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to Merative, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. Merative, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. Merative shall not be liable for any damages arising out of your use of the sample programs.

Privacy policy

The Merative privacy policy is available at https://www.merative.com/privacy.

Trademarks

Merative ™ and the Merative ™ logo are trademarks of Merative US L.P. in the United States and other countries.

IBM®, the IBM® logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

Adobe™, the Adobe™ logo, PostScript™, and the PostScript™ logo are either registered trademarks or trademarks of Adobe™ Systems Incorporated in the United States, and/or other countries.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft™, Windows™, and the Windows™ logo are trademarks of Microsoft™ Corporation in the United States, other countries, or both.

UNIX™ is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.