ORACLE
INSURANCE

Oracle Insurance

Data Capture Activity Software Services Integration

Release 5.2

October 2014

Oracle Insurance Data Capture Activity Software Service Integration

Release 5.2

Part # E56147-01

Library# E56152-01

October 2014

Author: Mike Schwitzgebel

Contributing Authors: Mary Elizabeth Wiger

CONTENTS

STATEMENT OF PURPOSE

This document provides a functional description of the integration features provided by the Oracle Insurance Data Capture (OIDC) product. We will cover import, export, message content, and activity behaviors made available for implementing an activity callout adaptor.

INTEGRATION PRINCIPLES

A few basic principles should be kept in mind when considering integration with Data Capture:

- All input data fields are defined by the end user
- All output data fields are defined by the end user
- All input and output data fields defined by the user must be uniquely named by the user
- Data Capture treats all input and output fields as part of one document
- Data Capture can only import and export data in its integration XML format
- Data Capture provides an “adaptor” mechanism to convert its integration XML format to custom formats
- Data Capture understands several integration “activities” and exhibits “value-add” behavior for these

In simple terms, Data Capture manages a document. The contents that this document can hold are defined by the end user. Data Capture can only receive and export data defined in this document.

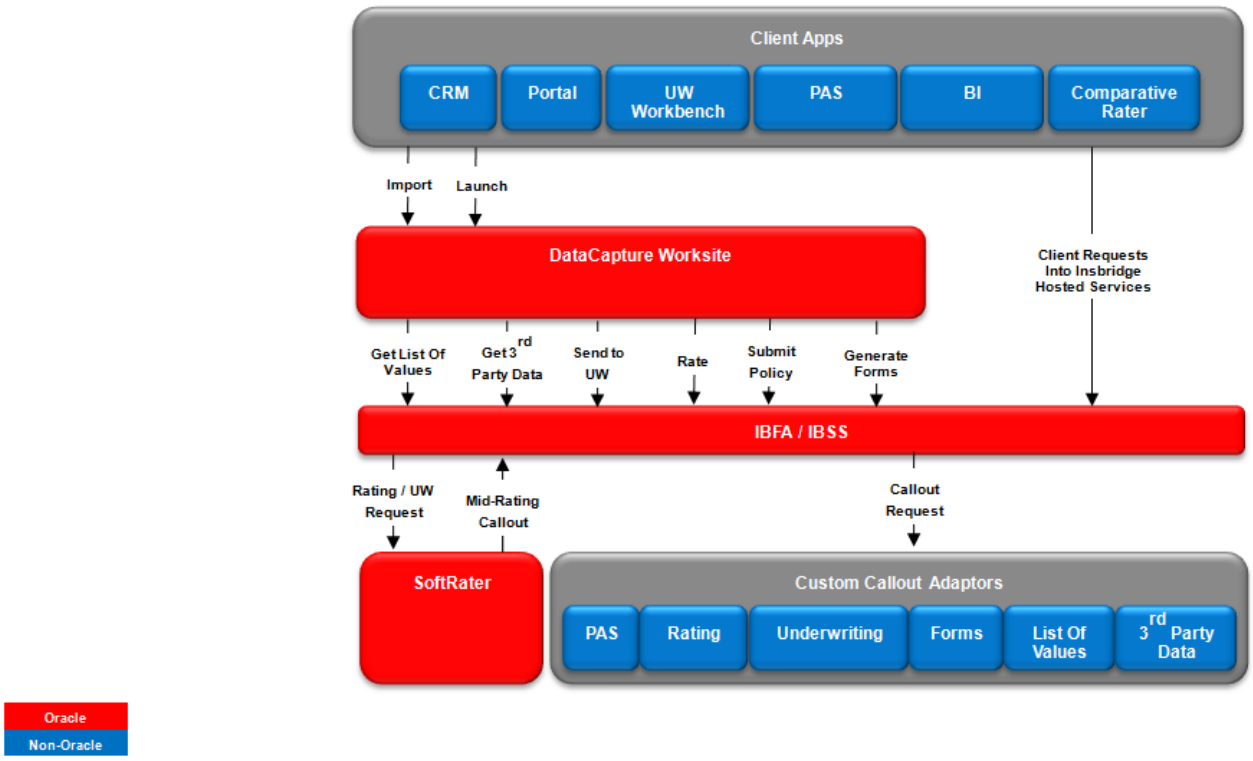

Figure 1 Integration Principals

Data Capture provides for various types of integrations including software services, security, portal embedding, and so on. Here we are concerned with software service integration. This type of integration allows Data Capture to bring data in from external sources and push data out to external targets. Data Capture can pull in data at any point during the questionnaire flow by making a callout to an external system. This same callout mechanism is used to export data to external systems.

OIDC has specific APIs for specific types of integrations or “activities”. Each type requires specific adaptor logic. Though OIDC is dictating what the API looks like, these APIs could be exposed as client APIs to other adaptor clients (not just OIDC) through IBSS.

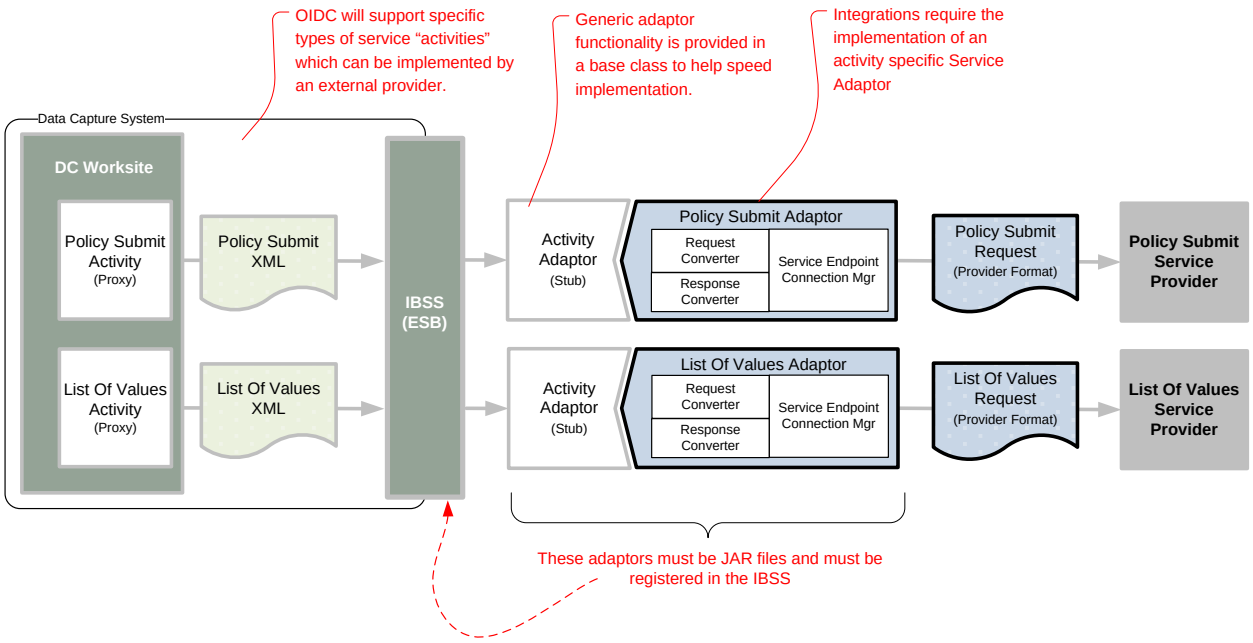

Figure 2 APIs

Setting up a Callout

Setting up a callout involves several different user roles, tools and activities.

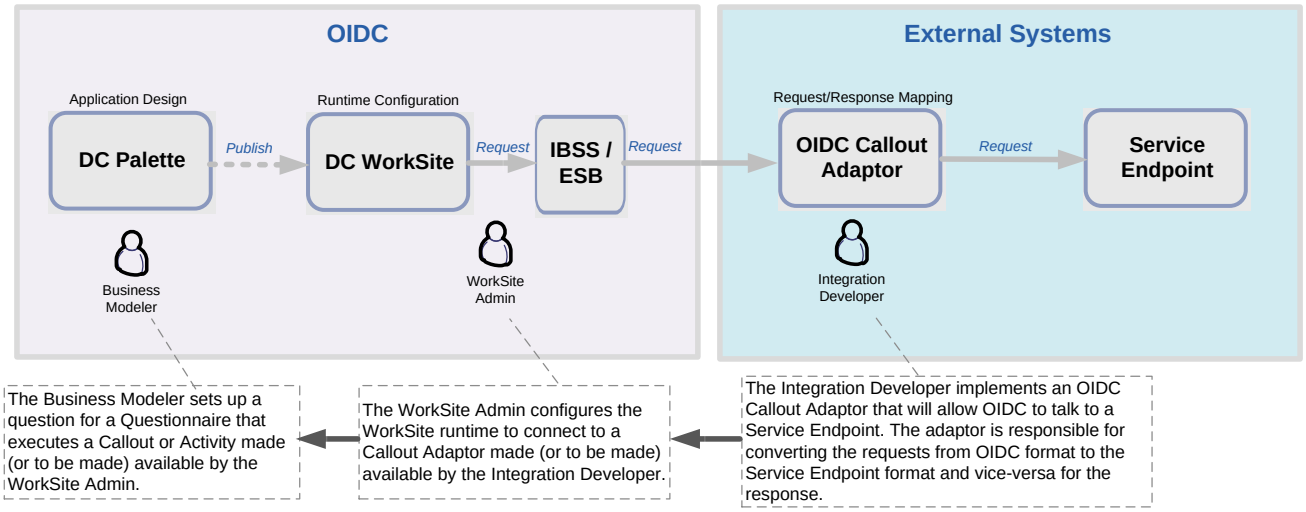

Figure 3 Setting up a Callout

In the diagram above, the runtime callout data flow goes left to right, but, when configuring a callout, it makes more sense to setup the involved components from right to left. The pieces are as follows:

- **Service Endpoint** – When you begin setting up OIDC to call an external system, that external system is the "Service Endpoint" you are trying to communicate with. That service endpoint must have some sort of service API for interaction with other software

clients. Typical service endpoints today will be implemented as web services, though that is not required. It is assumed that this endpoint already exists and its communication API is known. You will not be able to integrate it with OIDC otherwise.

- **OIDC Callout Adaptor** – OIDC has its own API for exporting data, importing data, and executing certain activities. As described above, it is assumed the Service Endpoint has an API for receiving client requests. A service request from OIDC to the Service Endpoint must be "adapted" to adhere to the Service Endpoint's API. An OIDC Callout Adaptor is responsible for converting OIDC requests to that of the target Service Endpoint. It's also responsible for converting the Service Endpoint's response into an OIDC response. This component must be built by a programmer and must adhere to an OIDC Activity API.

- **ESB** – A runtime Enterprise Service Bus is provided by the OIDC system. Your Callout Adaptor must be registered with the Insbridge Software Services (IBSS) bus as a SoftLibrary. The IBSS makes your adaptor component available at runtime through its bus API. You will have to register your adaptor under an "admin name" so IBSS clients can identify it for communication. That name can be anything you want but should reflect the functionality that is being provided by the Service Endpoint. The IBSS "admin name" is the same as the OIDC "service name" identified in some OIDC Activity configurations.

- **Worksite** – The OIDC runtime environment, Worksite, must be configured to map OIDC callout activities to your registered Callout Adaptor. The callout activities are uniquely named in the Worksite configuration. The Worksite callout activity name is used in question definitions to identify the callout to execute. At runtime the callout is triggered by answering a question or clicking on a button. Callout results are stored and/or merged into the project or used to initialize a visual control. Callout responses can include HTML to be displayed as well.

- **Palette** – The OIDC Worksite definition environment, Palette, is used to create questions that trigger callouts. To trigger a callout, a question must be setup to execute a callout activity. This is done by defining a callout activity in the question's AnswerMap identifying the Worksite activity to be used.

THE CALLOUT ADAPTER

The Callout Adaptor is the primary component needed to integrate OIDC with external systems. It serves as a translator between OIDC communication and that of the target system. This component is not provided by OIDC but must adhere to the OIDC Callout API. A programmer familiar with developing web services will be required to implement this component. This includes:

- Building the component as a Java class registered with IBSS as a SoftLibrary
- Converting OIDC SSIX requests to the target endpoint's request format
- Converting the target endpoint's response into an OIDC SSIX response
- Optionally, returning any HTML to be displayed in a report panel

The following diagram illustrates the high level relationships between OIDC and Callout Adaptors, generally.

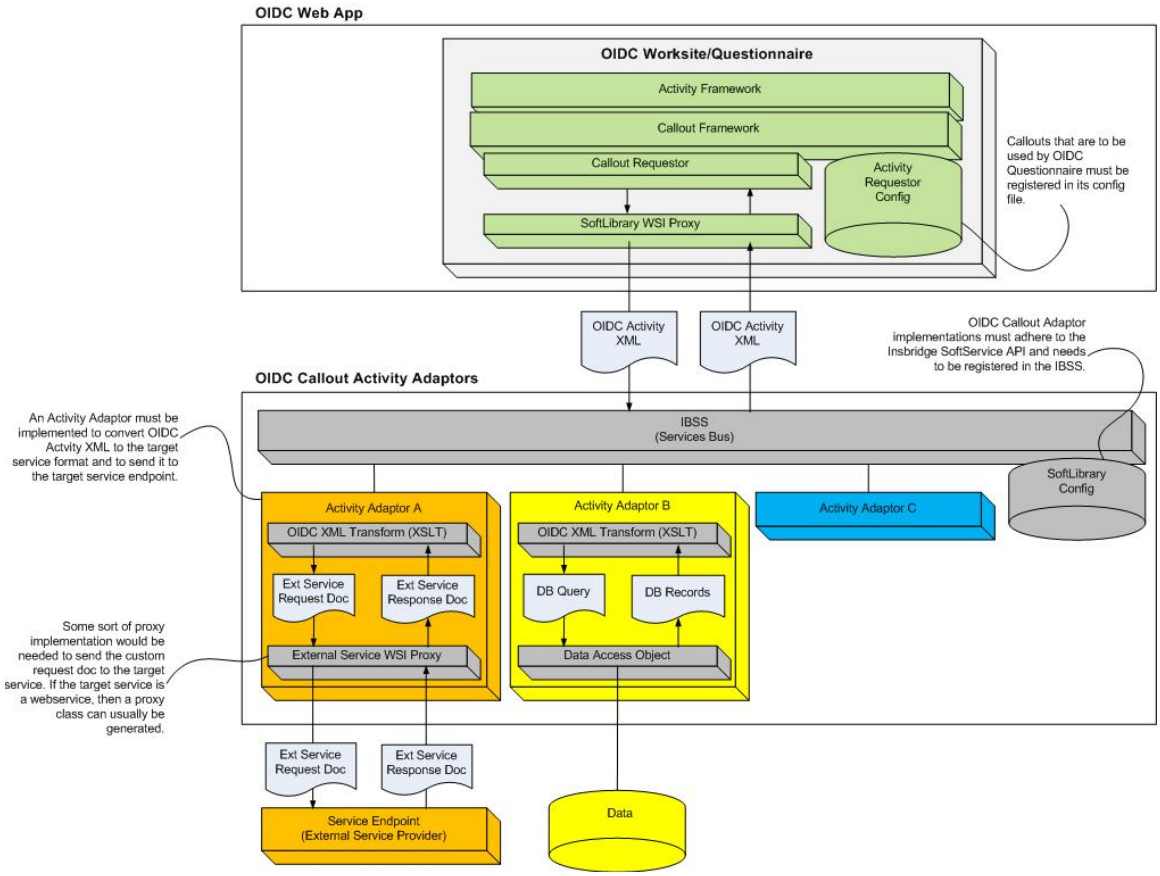

Figure 4 Callout Adapters

Callout adaptors, as shown in the diagram above, can be implemented to connect OIDC to external web-services, databases, or other components. The adaptor serves as a broker between the two apps, hiding the endpoint connection and communication details from OIDC and hiding OIDC API requirements from the endpoint. Adaptors are typically used to connect OIDC to the following types of services:

- Rating
- Underwriting
- Supplemental Data
- Reconciliation
- Valid Values
- Project Administration
- Document Generation
- Billing

NOTE: *At the time of writing, the only defined Activity Callout Adaptors are for the List of Values Activity, Supplemental Data Activity, and Project Submit Activity.*

SSIX – SOFTWARE SERVICES INTEGRATION XML

OIDC only imports and exports data in an XML document format called SSIX. This document has the following properties:

- A single integration XML format for passing data back and forth between OIDC and external apps
- A more streamlined XML format for smaller transmission footprint
- Overall transaction statuses
- Format for messages
- Format for arguments
- Location for debug info
- Location for response XHTML

Export

OIDC exports its collected data in the form of a SSIX request:

```
<se:ssix xmlns:se="http://oracle.com/ins/intg/ssix">
  <se:entities>
    <se:request>
      <se:arguments>
        <!-- Pass through or tracking arguments used by the client
application or customer -->
        <!-- Arguments used by the service processing this request -->
      </se:arguments>
      <se:data>
        <!-- The data to be processed by the service processing this
request -->
      </se:data>
    </se:request>
  </se:entities>
</se:ssix>
```

Import

OIDC imports data in the form of a SSIX response:

```
<se:ssix xmlns:se="http://oracle.com/ins/intg/ssix">
  <se:entities>
    <se:arguments>
      <!-- Echo of arguments passed in by the client application or
customer -->
    </se:arguments>
```

```
    <se:response>
      <se:data>
        <!-- The data processed and returned by the requested service -
->
      </se:data>
      <se:messages>
        <!-- Messages returned by the requested service -->
      </se:messages>
      <se:statuses>
        <!-- Overall status of the transaction -->
      </se:statuses>
      <se:debug>
        <!-- Any debug data that may help diagnose activity involved in
this request -->
      </se:debug>
      <se:reportHTML>
        <!-- Restricted XHTML compliant content that can be returned by
the target
             service to be displayed in a report panel -->
      </se:reportHTML>
    </se:response>
  </se:entities>
</se:ssix>
```

<data> Content Format

The <data> element contains the actual request data being evaluated or used for calculation. The format of its child XML fragment will vary with the type of Activity request being processed.

Sample data content XML

```
<sx:data>
  <act:ProjectSubmit xmlns:act="http://oracle.com/ins/intg/activity">
    <act:Activity serviceAdaptorName="ProjectSubmitMockImpl">
      <!--
      OIDC will always send the Activity element and the
      serviceAdaptorName may have no child elements, depending on the
      Activity type and the requirements of the adaptor logic.
      -->
    </act:Activity>
    <pc:Product customId="" productConfigId=""
      xmlns:pc="http://oracle.com/insurance/ProductConfiguration">
      <!--
      OIDC may or may not send a Product element in the request. If the
      AnswerMap that triggers the Activity callout sets the
      includeProjectData to "true", OIDC includes Product in the
      request; otherwise, Product is omitted.
      -->
    </pc:Product>
  </act:ProjectSubmit>
</sx:data>
```

<message> Content Format

Messages can be returned in callout responses. General messages can appear under the <response><messages> node, but they can also appear under any named object element that is a child of <Categories>, such as <Insured>, or <Vehicle>, if they are specific to that object. Three message types are supported:

- RemarkText
- Warning
- Failure

If a failure message is returned then the overall TransactionStatus should be set to FAIL (see <statuses>)

```
<se:messages>
  <se:msg t="Failure">
    <se:p n="MsgCd" v="09991823" />
    <se:p n="MsgSourceCd" v="Somewhere" />
    <se:p n="MsgDesc" v="Something didn't work!!" />
  </se:msg>
  <se:msg t="Warning">
    <se:p n="MsgSourceCd" v="CarrierRating" t="MsgSourceCd"/>
    <se:p n="MsgDesc" v="Mapping Program:  COLL deductible changed on
Vehicle #1" />
  </se:msg>
  <se:msg t="RemarkText">
    <se:p n="MsgSourceCd" v="CarrierRating" />
    <se:p n="MsgDesc" v="This policy is going into the SIS database" />
  </se:msg>
  <se:msg t="RemarkText">
    <se:p n="MsgSourceCd" v="CarrierRating" t="MsgSourceCd"/>
    <se:p n="MsgDesc" v="We are not able offer coverage with Acme
Insurance"/>
  </se:msg>
</se:messages>
```

<statuses> Content Format

The overall transaction status is contained by the <statuses> element. The <status> of type *TransactionStatus* is required on every response and must have a *TransactionStatusCd* of either *PASS* or *FAIL*. It is highly recommend that the *StatusDesc* be returned on failures so OIDC users can have an idea of what may have contributed to the failure. One or more <messages><msg> entries of type *Failure* can be returned for more detail regarding a failure.

```
<se:statuses>
  <se:status t="TransactionStatus">
    <se:p n="TransactionStatusCd" v="FAIL" />
    <se:p n="CustomTransactionStatusCd" v="Service Failure"/>
    <se:p n="Provider" v="ChoicePoint"/>
    <se:p n="Service" v="MVR Callout"/>
    <se:p n="StatusDesc" v="Unable to acquire database connection"/>
  </se:status>
</se:statuses>
```

ACTIVITY REQUEST/RESPONSE EXAMPLES

As noted previously, the XML content contained within the <data> element of a SSIX request or response payload differs from one Activity type to another. Following are examples payloads for defined Activity types.

List of Values Activity

Sample request payload

```
<data>
  <act:ListOfValues xmlns:act="http://oracle.com/ins/intg/activity">
    <act:Activity serviceAdaptorName="LOVTestSoftData">
      <act:Arg name="TargetID" value="2"/>
      <act:Arg name="DisplayTextResultID" value="3"/>
      <act:Arg name="ValueResultID" value="2"/>
      <act:Filters>
        <act:Filter name="StateProvCd" index="2" value="OH"/>
        <act:Filter name="CoverageCd" index="1" value="PD"/>
      </act:Filters>
    </act:Activity>
  </act:ListOfValues>
</data>
```

<ListOfValues>

- The wrapper element is named for the activity type.

<Activity> attributes:

- *serviceAdaptorName* – the name under which the activity adaptor is registered with IBSS as a SoftLibrary.

<Arg> attributes:

- *name* – the argument name
- *value* – the argument value

The meanings of these arguments are agreed upon by the business analyst and the implementer of the Activity adaptor. It is valid for the *value* attribute to be empty or missing, and it is the adaptor's responsibility to handle these cases.

<Filter> attributes:

- *name* – the filter name. In many cases, this name will be the same as the column name, element name, etc. against which the value will be compared. One exception occurs

when mapping to an Insbridge SoftData request, which uses the index attribute (see below) to locate the target field.
- *value* – contains a value that will be used to match the target data.
- *index* – (optional) for repositories that require an integer locator, like SoftData, the optional *index* attribute may be used.

If these descriptions seem vague, it is partly because their precise meanings may vary with the adaptor implementation.

Sample response payload

```
<se:data>
  <lov:ListOfValues xmlns:lov="http://oracle.com/ins/intg/activity">
    <lov:Map desc="" id="" name="">
      <lov:A name="-- Select One --" value=""/>
      <lov:A name="$15,000/$30,000" value="15/30"/>
      <lov:A name="$25,000/$50,000" value="25/50"/>
      <lov:A name="$50,000/$100,000" value="50/100"/>
      <lov:A name="$100,000/$300,000" value="100/300"/>
      <lov:A name="$250,000/$500,000" value="250/500"/>
    </lov:Map>
  </lov:ListOfValues>
</se:data>
```

Project Submit Activity

<ProjectSubmit>

- The wrapper element is named for the activity type. Its contents will vary with the definition of the Product Object Model (POM).

<Activity> attributes:

- *serviceAdaptorName* – the name under which the activity adaptor is registered with IBSS as a SoftLibrary.

<Arg> attributes:

- *name* – the argument name
- *value* – the argument value

The meanings of these arguments are agreed upon by the business analyst and the implementer of the Activity adaptor. It is valid for the *value* attribute to be empty or missing, and it is the adaptor's responsibility to handle these cases.

<Filter> attributes:

- *name* – the filter name. In many cases, this name will be the same as the column name, element name, etc. against which the value will be compared. One exception occurs when mapping to an Insbridge SoftData request, which uses the index attribute (see below) to locate the target field.
- *value* – contains a value that will be used to match the target data
- *index* – (optional) for repositories that require an integer locator, like SoftData, the optional *index* attribute may be used

Sample request payload

```
<sx:data>
  <act:ProjectSubmit xmlns:act="http://oracle.com/ins/intg/activity">
    <act:Activity serviceAdaptorName="ProjectSubmitMockImpl" />
    <pc:Product
        xmlns:pc="http://oracle.com/insurance/ProductConfiguration"
        customId="" productConfigId="">
      <pc:Categories>
        <pc:Insured id="0_Insured_1" d="Insured">
          <pc:Fields>
            <pc:FirstName>
              <pc:Value>Kermit</pc:Value>
            </pc:FirstName>
            <pc:LastName>
              <pc:Value>Klipsch</pc:Value>
            </pc:LastName>
            <pc:GenderCd>
              <pc:Value>M</pc:Value>
            </pc:GenderCd>
            <pc:DoNotSolicitInd>
              <pc:Value>1</pc:Value>
              <pc:DispValue>Yes</pc:DispValue>
            </pc:DoNotSolicitInd>
            <pc:BirthDt>
              <pc:Value>1987-1-23</pc:Value>
              <pc:DispValue>1/23/1987</pc:DispValue>
            </pc:BirthDt>
            <pc:MaritalStatusCd>
              <pc:Value>S</pc:Value>
            </pc:MaritalStatusCd>
            <pc:OccupationDesc>
              <pc:Value>Mattress tester</pc:Value>
            </pc:OccupationDesc>
          </pc:Fields>
        </pc:Insured>
        <pc:Policy id="0_Policy_1" d="Policy">
          <pc:Fields>
            <pc:PolicyStatusCd>
              <pc:Value></pc:Value>
            </pc:PolicyStatusCd>
            <pc:ControllingStateProvCd>
              <pc:Value>MI</pc:Value>
            </pc:ControllingStateProvCd>
            <pc:InsurerName>
              <pc:Value>Illustrations</pc:Value>
            </pc:InsurerName>
          </pc:Fields>
        </pc:Policy>
      </pc:Categories>
    </pc:Product>
  </act:PolicySubmit>
</sx:data>
```

You can see that the data payload for a PolicySubmit Activity request is quite different from that for a ListOfValues request. In this case, the payload consists of the collected values from a Data

Capture project exported to the format defined in the Product Object Model (POM) schema. The SSIX data element is the same. Its first child is a wrapper element whose name is the same as the Activity's, and there is an empty Activity element, but the rest of the data is dictated by the POM.

Sample response payload

```
<sx:data>
  <act:ProjectSubmit xmlns:act="http://oracle.com/ins/intg/activity">
    <Product d="" id="0"
        xmlns="http://oracle.com/insurance/ProductConfiguration">
      <Categories>
        <Policy id="0_Policy_1" t="Policy">
          <Fields>
            <OIPAPolicyNumber>
              <Value>STUB339125923</Value>
            </OIPAPolicyNumber>
            <PolicyNumber>
              <Value>STUB339125923</Value>
            </PolicyNumber>
            <HoldingTypeCode>
              <Value>Policy</Value>
            </HoldingTypeCode>
            <HoldingStatus>
              <Value>Proposed - Pending - Not yet inforce</Value>
            </HoldingStatus>
          </Fields>
        </Policy>
      </Categories>
    </Product>
  </act:ProjectSubmit>
</sx:data>
```

Like the request payload, the ProjectSubmit response payload follows the POM schema, in terms of the way the Categories and Fields are structured. It is not necessary to return elements for *all* Categories and Fields in the POM schema – just those with values that should be reflected in the Data Capture project. Also note that while it is acceptable to include Categories and Fields that are *not* defined in the POM schema, these will be ignored by Data Capture when the response payload is imported.

Supplemental Data Activity

<SupplementalData>

- The wrapper element is named for the activity type. Its contents will vary with the definition of the Product Object Model (POM).

<Activity> attributes:

- *serviceAdaptorName* – the name under which the activity adaptor is registered with IBSS as a SoftLibrary

Note: The Activity element and its serverAdaptorName attribute are required but may be empty. Arg and Filter elements are required only if the adaptor implementation has need of them.

<Arg> attributes:

- *name* – the argument name
- *value* – the argument value

The meanings of these arguments are agreed upon by the business analyst and the implementer of the Activity adaptor. It is valid for the *value* attribute to be empty or missing, and it is the adaptor's responsibility to handle these cases.

<Filter> attributes:

- *name* – the filter name. In many cases, this name will be the same as the column name, element name, etc. against which the value will be compared. One exception occurs when mapping to an Insbridge SoftData request, which uses the index attribute (see below) to locate the target field.
- *value* – contains a value that will be used to match the target data (ex.,
- *index* – (optional) for repositories that require an integer locator, like SoftData, the optional *index* attribute may be used

Sample request payload

```
<sx:data>
  <act:SupplementalData
      xmlns:act="http://oracle.com/ins/intg/activity">
    <act:Activity serviceAdaptorName="SupplementalDataMockService">
      <act:Arg name="Service"
          value="SupplementalDataCalloutRecordsXML"/>
      <act:Arg name="ObjectType" value="Insured"/>
      <act:Arg name="Provider" value="SampleContent"/>
      <act:Filters>
        <act:Filter name="BirthDt" value="1960-12-23"/>
        <act:Filter name="FirstName" value="Andrew"/>
        <act:Filter name="LastName" value="Anderson"/>
      </act:Filters>
    </act:Activity>
    <pc:Product
      xmlns:pc="http://oracle.com/insurance/ProductConfiguration"
      customId="" productConfigId="">
      <pc:Categories>
        <pc:Insured id="0_Insured_1" d="Andrew Anderson">
          <pc:Fields>
            <pc:OccupationDesc>
              <pc:Value>Analyst</pc:Value>
            </pc:OccupationDesc>
            <pc:MiddleName>
              <pc:Value>A</pc:Value>
            </pc:MiddleName>
            <pc:FirstName>
              <pc:Value>Andrew</pc:Value>
              <pc:DispValue>Andrew</pc:DispValue>
            </pc:FirstName>
            <pc:BirthDt>
              <pc:Value>1960-12-23</pc:Value>
              <pc:DispValue>12/23/1960</pc:DispValue>
            </pc:BirthDt>
```

```
            <pc:LastName>
              <pc:Value>Anderson</pc:Value>
              <pc:DispValue>Anderson</pc:DispValue>
            </pc:LastName>
            <pc:GenderCd>
              <pc:Value>M</pc:Value>
            </pc:GenderCd>
            <pc:MaritalStatusCd>
              <pc:Value>M</pc:Value>
            </pc:MaritalStatusCd>
            <pc:Age>
              <pc:Value>52</pc:Value>
              <pc:DispValue>52</pc:DispValue>
            </pc:Age>
            <pc:SupplementalData>
              <pc:Value>FAIL</pc:Value>
              <pc:DispValue>Fail (or Re-order)</pc:DispValue>
            </pc:SupplementalData>
            <pc:DoNotSolicitInd>
              <pc:Value>1</pc:Value>
              <pc:DispValue>1</pc:DispValue>
            </pc:DoNotSolicitInd>
          </pc:Fields>
        </pc:Insured>
```

[some data omitted for brevity]

```
      </pc:Categories>
    </pc:Product>
  </act:SupplementalData>
</sx:data>
```

You can see that the data payload for a SupplementalData Activity request is similar to that for a PolicySubmit request. In this case, the payload consists of the collected values from a Data Capture quote exported to the format defined in the Product Object Model schema. The SSIX data element is the same and its first child is a wrapper element whose name is the same as the Activity's.

Note that the Activity element in this sample SupplementalData request includes Arg and Filter elements, like those we've seen in the ListOfValues request. This example also includes a Product payload – but it is not *required*. One implementation might require only a payload and an empty Activity element; another might need only the information conveyed in the Arg and Filter elements of the Activity element. It all depends on the needs of the adaptor implementation.

Sample response payload

```
<sx:data>
  <act:SupplementalData
      xmlns:act="http://oracle.com/ins/intg/activity">
    <Product xmlns="http://oracle.com/insurance/ProductConfiguration">
      <Categories>
        <Policy d="" id="0_Policy_1">
          <Fields>
            <EffectiveDate>
              <Value>2013-09-17</Value>
            </EffectiveDate>
```

```
            <ContractTerm>
              <Value>12</Value>
            </ContractTerm>
            <ControllingStateProvCd>
              <Value>AL</Value>
            </ControllingStateProvCd>
            <LineOfBusinessCd>
              <Value>HO</Value>
            </LineOfBusinessCd>
          </Fields>
        </Policy>
        <Insured d="" id="0_Insured_1">
          <Fields>
            <FirstName>
              <Value>Andrew</Value>
            </FirstName>
            <MiddleName>
              <Value>A</Value>
            </MiddleName>
            <LastName>
              <Value>Anderson</Value>
            </LastName>
            <GenderCd>
              <Value>M</Value>
            </GenderCd>
            <BirthDt>
              <Value>1960-12-23</Value>
            </BirthDt>
            <MaritalStatusCd>
              <Value>M</Value>
            </MaritalStatusCd>
            <OccupationDesc>
              <Value>Analyst</Value>
            </OccupationDesc>
          </Fields>
        </Insured>
```

[data omitted for brevity]

```
      </Categories>
    </Product>
  </act:SupplementalData>
</sx:data>
```

Like the request payload, the SupplementalData response payload follows the POM schema, in terms of the way the Categories and Fields are structured. It is not necessary to return elements for *all* Categories and Fields in the POM schema – just those with values that should be reflected in the Data Capture project. Also note that while it is acceptable to include Categories and Fields that are *not* defined in the POM schema, these will be ignored by Data Capture when the response payload is imported.

IMPLEMENTATION EXAMPLE

As previously described, OIDC integration will typically occur in the following order

1. Service Endpoint setup
2. Creating a callout adaptor
3. Deploying callout adaptor to the ESB
4. Worksite configuration
5. Palette questionnaire implementation

Creating an Activity Callout Adapter

An OIDC Activity callout adaptor must be implemented as an Insbridge SoftLibrary. The Insbridge SoftLibrary framework makes SoftLibrary components available via a runtime service bus, which allows OIDC to communicate with multiple SoftLibrary instances from a single endpoint. The SoftLibrary framework provides some testing and management utilities for the SoftLibrary implementations as well.

Each instance is required to implement Insbridge's ISoftService web service interface. In addition to this, OIDC requires each of these instances to transfer data using the OIDC Soft Services Integration XML (SSIX). These callout adaptors are required to convert SSIX to whatever format the target service expects, and to convert the target service response to SSIX. OIDC provides some base classes that can be used as a starting point.

Mapping Data to/from Data Capture

When implementing the Activity callout adaptor, the developer will need to know the format of the XML fragment contained by the SSIX <data> element. It is this content that needs to be mapped to the format understood by the Service Endpoint. Mapping is performed in the Activity callout adaptor code using XSLT transformation, construction of the Document directly in Java code, etc.

Deploying Callout Adapter (Insbridge IBSS SoftLibrary)

As mentioned, the callout adaptor is required to be a SoftLibrary implementation and must be registered with the Insbridge ESB service provided by the IBSS.

Prerequisites:

- Activity adaptor class deployed to a .jar
- Insbridge Engine for WebLogic or WebSphere installed and configured.

SoftLibrary

Register a SoftLibrary with IBSS by doing the following:

1. On the server where IBSS is running, find the SoftLibrary deployment directory. Depending on what your Oracle and Middleware home directories are called and what you have named the Insbridge domain, it should be something like the following:

   `C:\Oracle\Middleware\user_projects\domains\insbridge\lib`

2. Copy your adaptor class .jar to the lib folder.

3. Open IBSS (http://[server name:port]/IBSS) \

4. Expand the Nodes folder and select SoftLibraries.

5. On the SoftLibraries page, click the **Add SoftLibrary link** at the top of the window. A separate window is displayed.

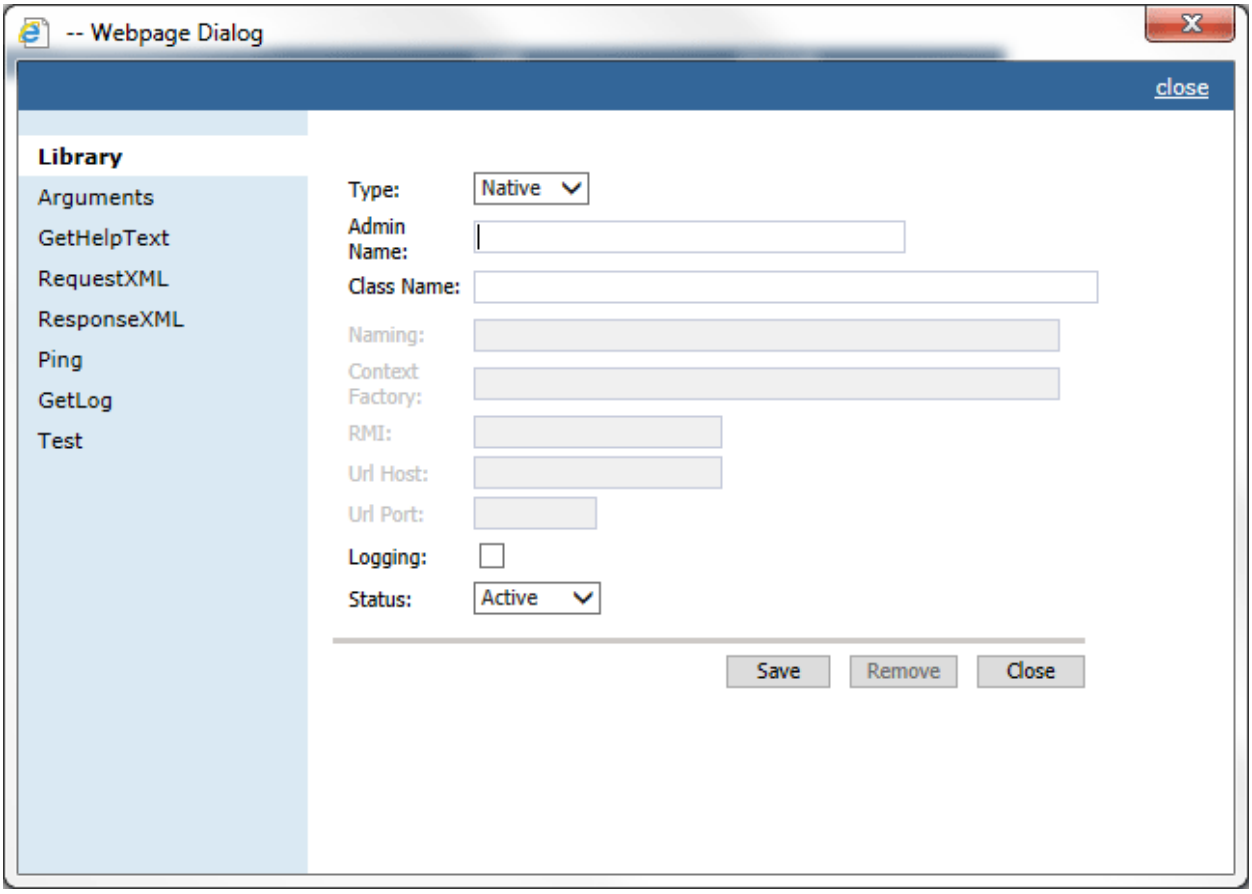

Figure 5 SoftLibraries Main Screen

The SoftLibrary window allows you to enter in basic information and to view critical information about the SoftLibrary. There are eight options on the side menu:

- **Library:** The basic SoftLibrary information is entered here.

- **Arguments:** Enter or edit the arguments for the SoftLibrary here.
- **GetHelpText:** This is an information window. If you want a quick view of the SoftLibrary, this window contains the basic information.
- **Request XML:** The request XML will be displayed here. This will tell you what requests you are sending out. This is an information only window.
- **Response XML:** The response XML will be displayed here. This will tell you exactly what you are getting back. This is an information only window.
- **Ping:** Will ping the server to verify server response.
- **GetLog:** If logging is turned on, this window will display logs.
- **Test:** Allows you to test the SoftLibrary.

There are three buttons located in the Library Add/Edit section.

- **Save** – Saves your work.
- **Remove** – Removes the entire library.
- **Close** – Closes the window without saving.

6. The first window is the Library information window. Information must be entered here before you can continue.

Select the Native Type. This type of library is installed as an application**.**

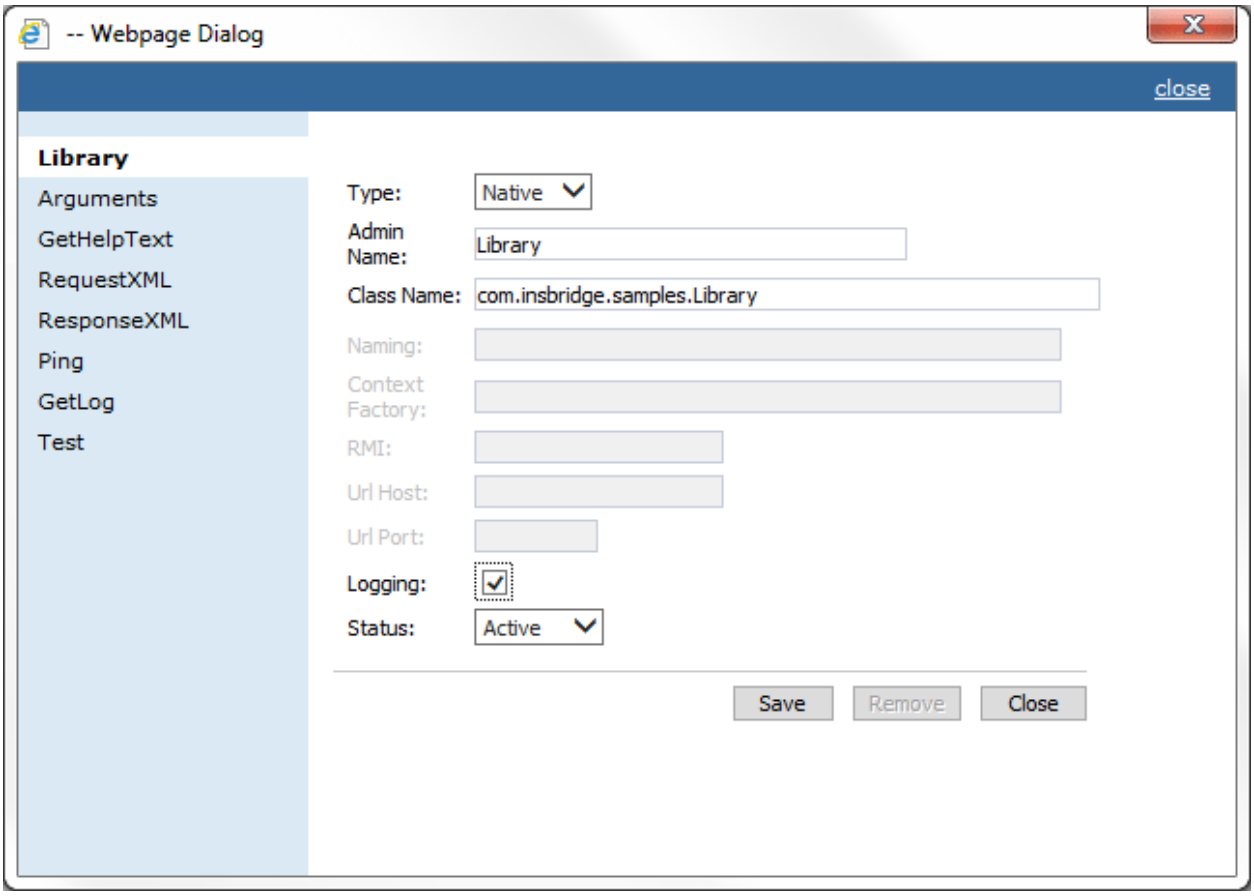

Figure 6 SoftLibrary Native Type

- **Admin Name:** This is the name of the SoftLibrary as it is displayed in RateManager. It is also the name that will be displayed on the lower section of the SoftLibraries page. Admin names must be unique.
- **Class Name:** This is the name of the actual SoftLibrary created by the developer from the program where it was created. This information can be obtained from the developer. Class names must be unique.
- **Logging:** Check this box if you want logging for this program. Leave it blank for no logging. The default is for no logging take place.
- **Status:** The status of the SoftLibrary. If active is selected, the SoftLibrary will be displayed to RateManager users. If disable is selected, the SoftLibrary will not will be displayed to users. The default status is disabled.

7. Click **Save** to add your SoftLibrary. It will be displayed on the lower portion of the screen.

If there are any errors with the new SoftLibrary, a message will be displayed at the top of the screen explaining the conflict.

DEVELOPING AN ACTIVITY CALLOUT ADAPTER AS A NATIVE SOFTLIBRARY

An Activity Callout Adaptor written as a native SoftLibrary is really nothing more than a plain old Java class that implements the ISoftService interface. A developer could build her adaptor from the ground up using nothing but her favorite Java development tool and the ISoftService.java interface definition.

That said, the following JARs are provided to provide some structure and make the job a bit easier:

- IBSoftService.jar – Provides stub implementations of all the methods defined in the ISoftService interface, as well as some helper methods for tasks common to most SoftServices.
- ActivityBaseSDK.jar – Provides an implementation of the Execute() method that takes care of initialization and some request payload validation. It also provides some useful methods to help with common XML manipulation tasks.
- libIntgUtil.jar – Provides XML utilities, a cached namespace resolver, and the SsixManager class.

Getting Started

At a high level, without getting into details of a specific IDE…

- Copy IBSoftService.jar, ActivityBaseSDK.jar, and libIntgUtil.jar to some directory where you can find them again.
- Create a new Java project in your IDE and add the above JAR files to its classpath.
- Create a new Java class that extends the type of Activity you want to implement. For example:

  ```
  public class ListOfValuesSampleXmlImpl extends
  ListOfValuesActivity
  ```

- Add an implementation for the required method processActivityRequest(). It is this method that will serve as the point of entry and exit to your custom implementation logic to and from the base Execute() method.

  ```
  public Document processActivityRequest() throws Exception
  ```

Here – or in additional methods called from here – you will retrieve the SsixManager instance that holds the already-parsed request Document, transform the request, transmit it to the business service, transform the response from the business service, and set the response into the SsixManager.

CALLOUT ADAPTER IMPLEMENTATION API

See Javadocs for…

- ActivityServiceSDK
- IBSoftServiceSDK
- IntegrationUtilities

OIDC now supports the following integration APIs:

- **Project Submit Activity API** - allows you to integrate any external system with Data Capture to perform a Project Submit activity. Data Capture will send your system all of its collected data, merge the returned results back into the Data Capture project, and update the project's status. You can also return a custom HTML report to display your policy submission results.
- **Supplemental Data Activity API** - allows you to integrate any external system with Data Capture to perform a Supplemental Data activity. Data Capture will send your system all of its collected data and will merge the returned results back into the Data Capture project.
- **List Of Values Activity API** - allows you to integrate any external system with Data Capture to perform a List Of Values lookup activity. You can use this feature to populate any selection list (dropdown, radiobox, checkbox, etc) in Data Capture with values supplied from an external system. You can use any collected or static data as request criteria.

CONTACTING SUPPORT

If you need assistance with an Oracle Insurance Insbridge Enterprise Rating System product, please log a Service Request using My Oracle Support at https://support.oracle.com/.

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Address any additional inquiries to:

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com