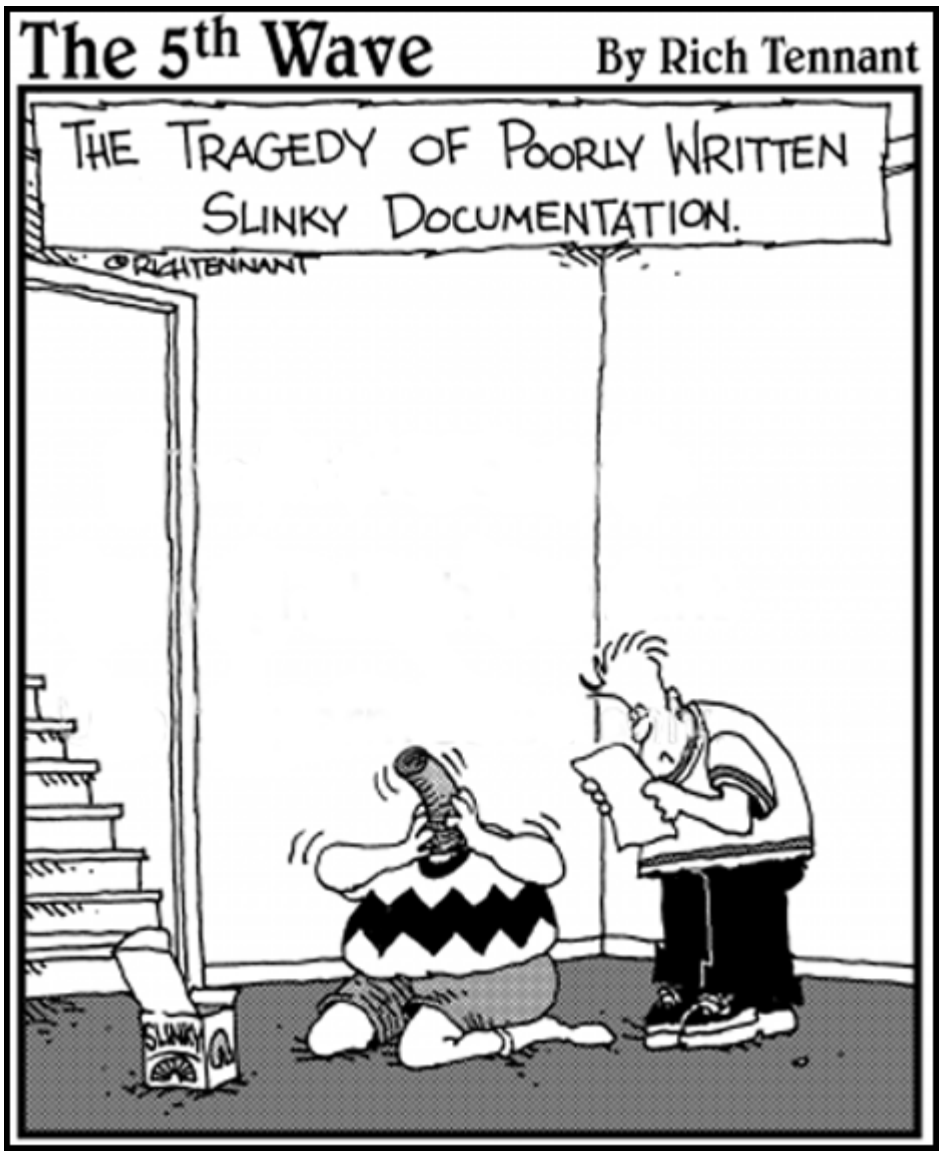
Philip GeLinas

Senior Programmer Writer


The 5th Wave
By Rich Tennant
THE TRAGEDY OF POORLY WRITTEN SLINKY DOCUMENTATION.
©RICHTENNANT
SLINKY

Résumé

Objective

SDK documentation specialist seeking new technical challenges in the Seattle area.

Documentation

SDK doc-sets including developer & API reference guides, code samples, and demo applications.

Tools, Technologies, and Processes

Eclipse, Netbeans, Framemaker, RoboHelp, Javadoc, Agile, TDD, SOAP, REST, Java.

Professional Experience

Senior Programmer Writer, Atigeo, Inc., Bellevue, WA (2010 to Present)
Currently document APIs for a Big Data Analytics Platform.

Senior API Writer, Irdeto, Carlsbad, CA (2008 ? 2010)
Documented APIs for a multimedia provisioning platform with 50 web services and 4000 methods.

Technical Publications Engineer II, Fair Isaac Corp., San Diego, CA (2007 to 2008)
Documented APIs for financial and fraud analytics platforms.

Senior Technical Writer, MediaNet, Inc., Seattle, WA (2004 to 2007)
Documented APIs for a multimedia platform.

Senior Technical Writer / Program Manager, Intel, Dupont, WA (2000 to 2002)
Delivered several large business systems and won Intel's SEC Division Award.

Technical Writer & Web Publisher, IBM (Keane), Rochester, MN (1998 to 2000)
Published online technical training materials for IBM's AS/400 mid-range business servers.

Education

Master's in Science Education (instructional design), EWU, Cheney, WA, 1997.

Bachelor of Arts in Education (natural Science / mathematics), EWU, Cheney, WA, 1995.

Professional training

Gordon's Documenting APIs and SDKs, Irdeto's Developer API Training Course, UML, Use Cases, System Analysis & Design.

Developer's Guides

This section contains several excerpts from developer's guides I have written over the past several years.

- Decision Accelerator Developer's Guide
- Intelligent Data Manager Developer's Guide
- Consortium Data Extract Developer's Guide

Decision Accelerator Developer's Guide

Capstone Decision Accelerator is a rule-based decision management system designed specifically for new account originations. Decision Accelerator provide companies with rules management technology that supports the entire process for designing, building and maintaining rules-based decision applications.

The Capstone Decision Accelerator Developer's Guide presents data input and output specifications for Decision Accelerator's remote API functions. The Preface and Overview sections of the guide are intended for information technology management professionals engaged in business planning. The entire book is intended for technical professionals responsible for creating an application that interfaces with Decision Accelerator and customizing Decision Accelerator for their organization.

In order to successfully implement the Decision Accelerator API functions on the client side, developers should have knowledge of XML parsing, XPath, XSLT, and XML schemas, in addition to the programming language used for implementation (such as Java). Depending on the deployment option chosen during installation, developers may need to have an understanding of Enterprise JavaBeans (EJBs), remote method invocation (RMI), and/or Web services.

- Processing XML Documents

Processing XML Documents

This chapter contains code examples that demonstrate how to process XML documents. It provides guidelines for extracting values from XML elements and attributes contained in XML output. This chapter includes the following sections:

XML Processing

One typical function that your calling application must perform is to search for a particular node (element or attribute) within an XML document and extract its value. Some XML nodes have the same name, but appear in two or more locations in the XML document (under different parents). Because new nodes will be added to the data model schema as Fair Isaac enhances Decision Accelerator functionality, it is important to search by using full path names as opposed to just the node name.

This chapter contains a Java code snippet for a small application that processes an XML document. The sample document (`TestDocument.xml`) is an excerpt of a typical Decision Accelerator output XML document, containing root, `CreditRequest`, `Applicant`, and `DecisionResponse` elements.

Note If you have questions about XML processing, contact Fair Isaac Product Support.

Test XML Document

The following XML document, `TestDocument.xml`, was used to test the example code shown in Figure 2 on page 12. The document contains multiple `ApplicantTelephone`, `UserDefinedField`, and `ReasonText` elements.

```
<Application DeliveryOptionCode="sysid"
ApplicationType="Individual" DecisioningRequestType="Final
Decision" ProcessingRequestType="DA" TransactionId="-
1062731253:86c347:100bfc5cf3b:-8000" Timestamp="Blaze Advisor -10-
17T09:30:47-05:00">
    <MessageList StatusCode="0" StatusDescription="Successful"/>
    <DecisionResponse>
        <Product>
            <Decision DecisionResult="Decline">
                <Reason ReasonCode="AB">
                    <ReasonText>gross total debt service ratio -
insufficient information available to calculate</ReasonText>
                </Reason>
                <Reason ReasonCode="CD">
                    <ReasonText>time at present address too short
or unknown</ReasonText>
                </Reason>
                <Reason ReasonCode="EF">
                    <ReasonText>present employment status</
ReasonText>
                </Reason>
            </Decision>
        </Product>
    </DecisionResponse>
    <CreditRequest ProductCategory="HomeEquity"
LoanPurpose="Renovation" ProductCode="HELOCRevolving"/>
    <Applicant ApplicantCrossReferenceId="222444888"
```

```
ApplicantType="Primary">
        <Personal PersonalIdNumber="444558888" Birthdate="1957-
11-15"/>
        <ApplicantTelephone PhoneNumber="4138887777"
PhoneType="Home"/>
        <ApplicantTelephone PhoneNumber="4138187171"
PhoneType="Mobile"/>
        <UserDefinedField>
            <Name>Customer Value Indicator</Name>
            <DataType>string</DataType>
            <Value>Gold</Value>
        </UserDefinedField>
        <UserDefinedField>
            <Name>In-house Deposit Balance</Name>
            <DataType>real</DataType>
            <Value>785.50</Value>
        </UserDefinedField>
    </Applicant>
</Application>
```

Figure 1: The TestDocument.xml file

Processing Code

The example code shown in Figure 2 demonstrates how to process an XML document using the *Xalan-Java* XML processor. Xalan-java builds on the *DOM* (Document Object Model) level 2 specification. The code evaluates XPath expressions to extract values. To build an XPath expression for a particular XML node, you must know the location of the XML node in the Decision Accelerator data model.

This code uses Xalan-Java version 2.6.0. The file `xalan-j-current-bin-2jars.zip` can be downloaded from http://xml.apache.org/xalan-j/downloads.html. Out of the JARs included in this download, there are three which you must include in the *classpath*:

- `xalan.jar`
- `xsltc.jar`
- `xercesImpl.jar`

See Also

- For more information about Xalan-Java, see http://xml.apache.org/xalan-j/index.html.
- For more information about the DOM, see http://www.w3.org/DOM/.
- For more information about XPath, see http://www.w3.org/TR/xpath.
- For more information about the Decision Accelerator data model, see Chapter 3, "The Capstone Origination Suite Data Model."

In some XML documents, an element may be repeated; each instance of the element may have different attribute values. The test document contains multiple `ApplicantTelephone` elements. The code sample demonstrates how to filter on the value

of the PhoneType attribute. The test document also contains some UserDefinedField elements. The processing code sample demonstrates how to filter these elements on known values of their Name attributes.

```
package Example;

import java.io.*;

import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.*;
import org.xml.sax.*;
import org.apache.xml.serialize.XMLSerializer;
import org.apache.xpath.*;

public class ParseXml
{
   /* main method receives an array of String */
   public static void main(String[] args)
   {
      if (args.length == 1)
         (new ParseXml()).parseFile(args[0]);
      else
         System.out.println("Pass the name of the file to parse.");
   }

   Document _doc;

   /* creates a new instance of ParseXml */
   public ParseXml()
   {
   }

   private
   void
   /* the name fo the file must be an absolute path */
   parseFile(String fileName)
   {
      try
      {
         /* parse xml to dom */
         DOMParser
         parser = new DOMParser();

         parser.parse(new InputSource(new FileReader(fileName)));

         _doc = parser.getDocument();

         showOutput();
      }
      catch(Exception Ex)
      {
         System.out.println(Ex.getMessage());
      }
   }

   /* all XPath expressions are in reference to the XML document (_doc) */
   private void showOutput() throws Exception
   {
      System.out.println("Credit Request Information:");
      System.out.println("    Decisioning Request Type: " + applyXPath("/Application/
      @DecisioningRequestType"));
      System.out.println("    Product Category: " + applyXPath("/Application/
```

```
    CreditRequest/@ProductCategory"));
    System.out.println("    Loan Purpose: " + applyXPath("/Application/
    CreditRequest/@LoanPurpose"));
    System.out.println("    Transaction ID: " + applyXPath("/Application/
    @TransactionId"));
    System.out.println("    Timestamp: " + applyXPath("/Application/@Timestamp"));

    System.out.println();
    System.out.println("Applicant Information:");
    System.out.println("    Applicant ID: " + applyXPath("/Application/Applicant/
    @ApplicantCrossReferenceId"));
    System.out.println("    Personal ID Number: " + applyXPath("/Application/
    Applicant/Personal/@PersonalIdNumber"));
    System.out.println("    Birthdate: " + applyXPath("/Application/Applicant/
    Personal/@Birthdate"));

    System.out.println();
    System.out.println("Telephone Information:");
    System.out.println("    Home: " + applyXPath("/Application/Applicant/
    ApplicantTelephone[@PhoneType='Home']/@PhoneNumber"));
    System.out.println("    Mobile: " + applyXPath("/Application/Applicant/
    ApplicantTelephone[@PhoneType='Mobile']/@PhoneNumber"));

    System.out.println();
    System.out.println("Customer Information:");
    System.out.println("    Customer Value Indicator: " + applyXPath("/Application/
    Applicant/UserDefinedField[Name = 'Customer Value Indicator']/Value/text()"));
    System.out.println("    In-house Deposit Balance: " + applyXPath("/Application/
    Applicant/UserDefinedField[Name = 'In-house Deposit Balance']/Value/text()"));

    System.out.println();
    System.out.println("Decision Result: " + applyXPath("/Application/
    DecisionResponse/Product/Decision/@DecisionResult"));
    System.out.println();
    System.out.println("Decision Reasons: ");
    String [] reasons = applyXPathMulti("/Application/DecisionResponse/Product/
    Decision/Reason/ReasonText/text()");
    for(int i = 0, iEnd = reasons.length; i < iEnd; ++i)
    System.out.println("    " + reasons[i]);
}

/* returns a single node */
private String applyXPath(String xpath) throws Exception
{
    Node
    node = XPathAPI.selectSingleNode (_doc, xpath);

    return  (node != null)
              ?   node.getNodeValue()
              :   "";
}
/* returns a node list, which is copied into an array of String */
private String[] applyXPathMulti(String xpath) throws Exception
{
    NodeList
    nodes = XPathAPI.selectNodeList(_doc, xpath);

    String[] results = new String[nodes.getLength()];

    for(int i = 0, iEnd = results.length; i < iEnd; ++i)
       results[i] = nodes.item(i).getNodeValue();

    return results;
```

```
   }

}
```

Processing Output

Figure 3 shows the result of processing the `TestDocument.xml` file using the code shown in Figure 2.

```
Credit Request Information:
    Decisioning Request Type: Final Decision
    Product Category: Home Equity
    Loan Purpose: Renovation
    Transaction ID: -1062731253:86c347:100bfc5cf3b:-8000
    Timestamp: Blaze Advisor -10-17T09:30:47-05:00

Applicant Information:
    Applicant ID: 222444888
    Personal ID Number: 444558888
    Birthdate: 1957-11-15

Telephone Information:
    Home: 4138887777
    Mobile: 4138187171

Customer Information:
    Customer Value Indicator: Gold
    In-house Deposit Balance: 785.50

Decision Result: Decline

Decision Reasons:
    gross total debt service ratio - insufficient information available to calculate
    time at present address too short or unknown
    present employment status
```

Figure 2: XML processing results

Intelligent Data Manager Developer's Guide

Intelligent Data Manager enables companies to deliver consistent, well-informed decisions by effectively managing and integrating data from external sources.

The Capstone Intelligent Data Manager Developer's Guide presents data input and output specifications for Intelligent Data Manager's remote API functions. The Preface and Overview sections of the guide are intended for information technology management professionals engaged in business planning. The entire book is intended for technical professionals responsible for creating an application that interfaces with Intelligent Data Manager and customizing Intelligent Data Manager for their organization.

In order to successfully implement the Intelligent Data Manager API functions on the client side, developers should have knowledge of XML parsing, XPath, XSLT, and XML schemas, in addition to the programming language used for implementation (such as Java).

Transaction Data and Decisioning

Before calling systems can submit transactions to Decision Accelerator to be decisioned, SMW users must configure decision flows to account for the types of requests that will be submitted. SMW users must provide calling system developers with information about the input and output data used by the decision flows. This chapter provides information on various Decision Accelerator decision flow components and their relation to the IBS data model and transaction data. It includes the following sections:

- Context Paths
- Data Object References
- Data Methods
- Data Methods and Data Object References Used in Decisioning
- Cross-Reference IDs
- Data Method History Output
- Processing History Output
- Using User-Defined Fields
- Resubmitting Transactions
- Internal Services
- Web Services
- Updating the Data Repository with Final Decision and Offer Data
- Using Decision Accelerator with Intelligent Data Manager

Context Paths

The data model schema represents the elements and attributes that can be used in XML input and output. The same schema, when imported into Blaze Advisor, represents the data objects and fields that can be used for decisioning and the relationships among these items.

In XPath and when referring to XML documents, parent and child elements are separated by slashes, and an attribute is separated from its containing element by "/@". For example:

- `Application/Finance/Income/Owner`
- `Application/Finance/Income/Owner/@OwnershipPercent`

In the SMW, elements are separated from each other and from attributes by dots (periods). The SMW equivalents of the paths above are:

- `Application.Finance.Income.Owner`
- `Application.Finance.Income.Owner.OwnershipPercent`

Decision flow components such as data methods, data object references, rules, and mappings can only access data within the *context* specified for the component. To specify a context in the SMW, you must choose the desired object from the Data Object Browser.

Data Object References

Data submitted for decisioning (instantiated objects) must conform to the data model schema. This actual data can have multiple occurrences of some objects (for example, multiple `ApplicantAddress` objects), while the schema only indicates that multiple `ApplicantAddress` objects are allowed. A data object reference can be used to specify which particular instances of objects should be used for decisioning. A data object reference returns a list of objects that satisfy a set of conditions; it can return multiple objects if multiple instances match the specified conditions.

For example, if you create a data object reference to look at previous addresses for the applicant, and the application input contains multiple previous addresses, the data object reference returns multiple previous addresses. If the data object reference is used as the context of a rule, when the rule executes, it operates on every object instance referred to by the data object reference.

The Data Object Reference Editor in the SMW features a Data Object Browser, which displays the imported data model in a tree structure. The root element (`Application`) is the root node of the tree, and each child element is shown indented one level from its parent. When a data object reference is added, it is shown as a leaf of its associated object.

Data Methods

Data methods are functions that can be authored in the SMW. When executed, they perform computational tasks, such as arithmetic operations, string manipulations, date/time field manipulations, and object creation and handling. They can be used to generate the data needed in business policies (rules). For example, data methods can be used to compute and return values in transaction output, to aggregate income for an individual, to compute the age of a piece of data given a timestamp, or to find the maximum value of a set of values. They can be used to attach data objects to each other, detach data objects, and call internal services or Web services.

The *Data Method Editor* in the SMW also uses the *Data Object Browser*. When used for authoring data methods, the Data Object Browser displays data object references and data methods that have already been created.

For testing purposes, Decision Accelerator can return information on the data methods executed during the execution of a decision flow. For more information on data method output, see "Data Method History Output" on page 19.

Note Decision Accelerator automatically creates only the objects listed in section 8 of "Decision Accelerator Data Model Constraints" on page 314. If you want other objects returned in transaction output (for example `Offer` and `Booking` objects under `Product`), you must create and attach those objects using data methods.

Data Methods and Data Object References Used in Decisioning

The following example shows how data object references and data methods can be used together to aggregate data submitted in multiple objects into a single value for decisioning.

In the StandardCard product in the pre-defined product strategies delivered with Decision Accelerator, some of the rules are based on applicant income. The LowMonthlyIncome rule compares the primary applicant's monthly income to a constant. The rule uses the data object reference `dor_PrimaryApplicant` and the data method `dm_App_WeightedMonthlyIncome`.

The data method `dm_App_WeightedMonthlyIncome` calculates the weighted sum of monthly income for an applicant, given stated ownership percentage. Because the data model allows income items to be entered with various frequencies (such as weekly, monthly, quarterly, and yearly), the data method calls a second data method (`dm_Util_MonthlyAmount`) that converts the amount from the given frequency to the amount for monthly frequency. The `for` loop of the data method uses the context path (`Application.Finance.Income.Owner`) defined by the `dor_AllIncomeOwner` data object reference.

Figure 4 on page 17 shows an example of some input `Income` nodes for an application with two applicants. Each income item is cross-referenced to either the primary or co-applicant. To use database terminology, the `Owner/@ApplicantCrossReferenceId` attribute acts as a "foreign key", referring back to the "primary key" (the `ApplicantCrossReferenceId` attribute of the `Applicant` object).

```
<Application>
…
    <Applicant ApplicantCrossReferenceId="1"
ApplicantType="Primary"/>
    <Applicant ApplicantCrossReferenceId="2"
ApplicantType="CoApplicant"/>
    <Finance>
        <Income IncomeType="Primary" IncomeAmount="7200"
Frequency="Annually" SourceType="Self-employed"
VerifiedFlag="false" StartDate="1985-05-07"
IgnoreIncomeItemFlag="false">
            <Owner ApplicantCrossReferenceId="1"/>
        </Income>
        <Income IncomeType="Primary" IncomeAmount="330"
Frequency="Bi-weekly" SourceType="Primary job"
VerifiedFlag="true" VerificationDate="2004-08-15"
StartDate="2001-03-26" IgnoreIncomeItemFlag="false">
            <Owner ApplicantCrossReferenceId="2"/>
        </Income>
        <Income IncomeType="Ancillary" IncomeAmount="300"
Frequency="Quarterly" SourceType="Investment"
VerifiedFlag="false" IgnoreIncomeItemFlag="false">
            <Owner ApplicantCrossReferenceId="1"/>
        </Income>
        <Income IncomeType="Ancillary" IncomeAmount="40"
Frequency="Weekly" SourceType="Non job-related"
VerifiedFlag="false" StartDate="2002-11-02"
AnticipatedEndDate="2006-01-01" IgnoreIncomeItemFlag="true">
            <Owner ApplicantCrossReferenceId="1"
OwnershipPercent="75"/>
            <Owner ApplicantCrossReferenceId="2"
OwnershipPercent="25"/>
        </Income>
    </Finance>

…</Application>I
```

The `dm_Util_MonthlyAmount` data method converts the income amounts for each applicant as follows:

Applicant	Income amount	Frequency	Conversion factor	Monthly amount
1	$7200.00	Yearly	amount / 12	$600.00
1	$300.00	Quarterly	amount / 3	$100.00
1	$40.00	Weekly	amount * 4.33	$173.20
2	$330.00	Bi-weekly	amount * 2.17	$716.10
2	$40.00	Weekly	amount * 4.33	$173.20

The `dm_App_WeightedMonthlyIncome` data method calculates the weighted monthly income for each applicant as follows:

Applicant	Monthly amount	Ownership %	Weighted amount
1	$600.00	n/a	$600.00
1	$100.00	n/a	$100.00
1	$173.20	75	$129.90
Total			**$829.90**
2	$716.10	n/a	$716.10
2	$173.20	25	$43.30
Total			**$759.40**

The context of the LowMonthlyIncome rule uses the `dor_PrimaryApplicant` data object reference. The rule checks to see whether the weighted monthly income of the primary applicant ($829.90) is less than a constant value ($500). In this case, the *condition* of the rule is not met and the rule does not fire.

Cross-Reference IDs

This example in the previous section demonstrates the importance of using cross-reference IDs consistently throughout transaction input; without them, the pre-defined product strategies delivered with Decision Accelerator cannot identify the correct owners of each income item, and they cannot accurately perform aggregate calculations.

The value of `Application/Applicant/@ApplicantCrossReferenceId` should be an exact string match to the following attributes:

- `Application/CreditRequest/@ProceedsDeliveryApplicantCrossReferenceId`
- The `ApplicantCrossReferenceId` attribute of:
  - `Application/Business/Owner`
  - `Application/ExternalDataSource/BbDataSource/BbDataServiceRequest/BbRequest/BbApplicant`
  - `Application/ExternalDataSource/CraDataSource/CraDataServiceRequest/CraRequest/RequestedCra/CraInputArf/ApplicantCrossReference`
  - `Application/ExternalDataSource/CraDataSource/CraDataServiceRequest/CraRequest/CraApplicant`
  - `Application/ExternalDataSource/CraDataSource/CraRequestDefaults/DefaultRequestedCra/CraInputArf/ApplicantCrossReference`
  - `Application/ExternalDataSource/FesDataSource/FesDataServiceRequest/FesRequest/FesApplicant`
  - `Application/ExternalDataSource/FesDataSource/FesRequestDefaults/DefaultFesApplicant`
  - `Application/ExternalDataSource/SbfeDataSource/SbfeDataServiceRequest/SbfeRequest/SbfeApplicant`
  - `Application/Finance/Asset/Owner`

- `Application/Finance/Income/Owner`
- `Application/Finance/Liability/Owner`

Similarly, the value of `Application/Business/@BusinessCrossReferenceId` should be an exact string match to the `BusinessCrossReferenceId` attribute of the following elements:

- `Application/ExternalDataSource/BbDataSource/BbDataServiceRequest/LosRequest/LosBusiness`
- `Application/ExternalDataSource/BbDataSource/BbDataServiceRequest/BbRequest/RequestedBb/BbInputArf`
- `Application/ExternalDataSource/BbDataSource/BbDataServiceRequest/BbRequest/BbBusiness`
- `Application/ExternalDataSource/BbDataSource/BbRequestDefaults/DefaultRequestedBb/BbInputArf`
- `Application/ExternalDataSource/SbfeDataSource/SbfeDataServiceRequest/SbfeRequest/SbfeBusiness`
- `Application/ExternalDataSource/SbfeDataSource/SbfeDataServiceRequest/SbfeRequest/SbfeApplicant/SbfeBusinessApplicant`

Important Values must match exactly, and are case-sensitive. The introduction of as little as an extra leading or trailing space can cause problems when an application is decisioned.

Data Method History Output

For testing and debugging purposes, Decision Accelerator can return detailed information on data methods executed during decisioning. This information appears in `DataMethodHistory` output elements. All objects in the IBS data model used for Decision Accelerator contain `DataMethodHistory` children, with the following exceptions:

- `DataMethodHistory` objects and their descendants
- `UserDefinedField` objects and their descendants

As described in "Complex and Simple Elements" on page 64, simple elements (which may contain content such as report images, ARF text, or reason text) do not correspond to objects in the data model. Therefore, they do not contain `DataMethodHistory` children.

`DataMethodHistory` elements are not displayed in the Outline or DataModel tabs of the data model spreadsheet. Instead, they appear in the DataMethodHistory tab. The paths for `DataMethodHistory` and its descendants are shown below.

Paths:

- `<ParentElement>/DataMethodHistory`
- `<ParentElement>/DataMethodHistory/@Timestamp`
- `<ParentElement>/DataMethodHistory/Name`
- `<ParentElement>/DataMethodHistory/DataType`
- `<ParentElement>/DataMethodHistory/Value`

- <ParentElement>/DataMethodHistory/DataMethodInput
- <ParentElement>/DataMethodHistory/DataMethodInput/Name
- <ParentElement>/DataMethodHistory/DataMethodInput/DataType
- <ParentElement>/DataMethodHistory/DataMethodInput/Value
- <ParentElement>/DataMethodHistory/DataMethodOutput
- <ParentElement>/DataMethodHistory/DataMethodOutput/Name
- <ParentElement>/DataMethodHistory/DataMethodOutput/DataType
- <ParentElement>/DataMethodHistory/DataMethodOutput/Value

<ParentElement> is a placeholder for any element that can contain a DataMethodHistory object. An example of a DataMethodHistory element is shown in Figure 5.

```
<Application>
    <Applicant>
        <DataMethodHistory Timestamp="Blaze Advisor -11-
03T08:22:06-08:00">
            <Name>dmo_App_GetPreviousApplicantAddress_ByIndex</
Name>
            <DataType>ApplicantAddress</DataType>
            <Value>&lt;ApplicantAddress
AddressStatusIndicator="Previous" AddressType="Home"
UnparsedStreetAddress="1500 Calveros Blvd #13" City="Sheffield"
State="MA" PostalCode="01257" PostalPlusCode="5588" County=""
CountryCode="USA" ResidenceType="Condominium"
ResidentialStatusIndicator="Rents" MonthsAtAddress="50"
StartDate="1999-12-01" EndDate="2004-02-14" VerifiedFlag="true"
VerificationType="Phone bill" VerificationDate="Blaze Advisor -08-
09"&gt;&lt;/ApplicantAddress&gt;</Value>
            <DataMethodInput>
                <Name>xIndex</Name>
                <DataType>numeric</DataType>
                <Value>1</Value>
            </DataMethodInput>
            <DataMethodOutput>
                <Name>xCount</Name>
                <DataType>numeric</DataType>
                <Value>2</Value>
            </DataMethodOutput>
        </DataMethodHistory>
    </Applicant>
</Application>
```

Figure 3: A sample DataMethodHistory output node

Note The value of the `Value` element matches the data type listed in the `DataType` element. However, you should be aware that the actual values in the `Name`, `DataType`, and `Value` elements are strings.

Using Data Method History Output During Testing

When creating a data method using the Data Method Editor of the SMW, a Decision Accelerator user can select the Save History check box. When this option is selected and the data method is executed, information appears in the following output elements:

- `DataMethodHistory`: The output XML document contains one `DataMethodHistory` element for each data method that is executed during a decision flow. The `DataMethodHistory` element appears as a child of the data method's context object. It has the following descendants:
  - `Timestamp` attribute—The execution start date/time of the data method execution.
  - `Name` element—The data method's name.
  - `DataType` element—The data method's *result type*.
  - `Value` element—The value of the data method's "result" when the *data method expression* execution is complete. If the data method's result is an object, the `Value` contains an "escaped XML string" that contains an XML depiction of the result object and all its children. If the `Value` is null, no `Value` attribute is returned. A `Value` string of zero length appears as `<Value/>`.
  - One `DataMethodInput` element for each data method input variable, and one `DataMethodOutput` element for each data method *local variable*.
- `DataMethodHistory/DataMethodInput`: For each `DataMethodHistory` element in the XML output, there are 0-n `DataMethodInput` elements, one for each of the *input* parameters (variables) passed to the data method when it was executed. The `DataMethodInput` element has the following children:
  - `Name` element—The input variable's name.
  - `DataType` element—The input variable's data type.
  - `Value` element—The value of the input variable, as passed to the data method (that is, before the data method's expression was executed). If the input is an object, the `Value` contains an "escaped XML string" that contains an XML depiction of the input object and all its children. If the `Value` is null, no `Value` attribute is returned. A `Value` string of zero length appears as `<Value/>`.
- `DataMethodHistory/DataMethodOutput`: For each `DataMethodHistory` element in the XML output, there are 0-n `DataMethodOutput` elements, one for each of this data method's local variables.
  - `Name` element—The local variable's name.
  - `DataType` element—The local variable's data type.
  - `Value` element—The value of the local after the data method expression has been executed. If the local is an object, the `Value` contains an "escaped XML string" that contains an XML depiction of the local object and all its children. If

the Value is null, no Value attribute is returned. A Value string of zero length appears as <Value/>.

Data Method History Examples

Each of the following four examples illustrates how a data method would appear in the Data Method Editor and how it could be called. Each example contains a snippet of the input XML document sent for decisioning, and a snippet of the output document that shows the resulting *data method history*.

1. Simple Method, No Input Variables/No Local Variables

Table 1: Data Method Editor Input

Item	Value
Data Method Name	dm_Liability_IsBankruptcy
Description	Returns a true if the Liability is a bankruptcy
Context (Path+Element)	Application.Finance.Liability
Result Type	boolean
Save History	checked
Input Variables	none
Local Variables	none

Expression	`result := false;`
	`if (DerogatoryType = "Bankruptcy-Type7") then result := true; endif`
	`if (DerogatoryType = "Bankruptcy-Type11") then result := true; endif`
	`if (DerogatoryType = "Bankruptcy-Type12") then result := true; endif`
	`if (DerogatoryType = "Bankruptcy-Type13") then result := true; endif`

Method Calling

This method can be called from rules, mappings, or other methods and returns a boolean value. Here is an example of calling this method from another method:

Table 2: Data Method Call

Item	Value
Expression	`SomeField := dm_Liability_IsBankruptcy`

Sample XML Input Document

```
<Application>
…
    <Finance>
        <Liability LiabilityTypeCode="MRTG"
LiabilityAmount="364000" MonthlyPaymentAmount="2150"
OurAccountFlag="true">
            <Owner ApplicantCrossReferenceId="1" />
```

```
            <Owner ApplicantCrossReferenceId="2" />
        </Liability>
        <Liability DerogatoryType="Bankruptcy-Type12"
DerogatoryEventDate="1999-08-23" OurAccountFlag="false">
            <Owner ApplicantCrossReferenceId="1" />
        </Liability>
    </Finance>
…
</Application>
```

Saved Data Method History Output

The result is equal to “true” because one of the liabilities is a bankruptcy.

```
<Application>
…
    <Finance>
        <Liability LiabilityTypeCode="MRTG"
LiabilityAmount="364000" MonthlyPaymentAmount="2150"
OurAccountFlag="true">
            <Owner ApplicantCrossReferenceId="1" />
            <Owner ApplicantCrossReferenceId="2" />
            <DataMethodHistory Timestamp="Blaze Advisor -10-
14T17:55:06.253-08:00">
                <Name>dm_Liability_IsBankruptcy</Name>
                <DataType>boolean</DataType>
                <Value>false</Value>
            </DataMethodHistory>
        </Liability>
        <Liability DerogatoryType="Bankruptcy-Type12"
DerogatoryEventDate="1999-08-23" OurAccountFlag="false">
            <Owner ApplicantCrossReferenceId="1" />
            <DataMethodHistory Timestamp="Blaze Advisor -10-
14T17:55:06.253-08:00">
                <Name>dm_Liability_IsBankruptcy</Name>
                <DataType>boolean</DataType>
                <Value>true</Value>
            </DataMethodHistory>
        </Liability>
    </Finance>
…
</Application>
```

2. Simple Method, One Input Variable/No Local Variables

Table 3: Data Method Editor Input

Item	Value		
Data Method Name	dm_Root_ConcatenateVipNumber		
Description	Sets the Application VipNumber		
Context (Path+Element)	Application		
Result Type	string		
Save History	checked		
Input Variables	**Name**	**Data Type**	**Initial Value**
	xString	string	n/a
Local Variables	none		
Expression	`VipNumber := VipNumber + xString;` `result := VipNumber`		

Method Calling

This method can be called from other methods, but not from *data method sequences*, rules, or mappings. It returns a string value. Here is an example of calling this method from another method.

Table 4: Data Method Call

Item	Value
Expression	`dm_Root_ConcatenateVipNumber ("34");`

Sample XML Input Document

```
<Application SubmitterId="MySubmitterId" DeliveryOptionCode="sysid"
ApplicationType="Joint" DecisioningRequestType="Final Decision"
ProcessingRequestType="DA" ResubmissionFlag="false" PromotionCode="LFO808"
VipNumber="12"/>
```

Figure 4: The document containing the initial VipNumber

Saved Data Method History Output

The data method concatenates “34” to the initial value of “12” to create a result of “1234”.

```
<Application SubmitterId="MySubmitterId"
DeliveryOptionCode="sysid" ApplicationType="Joint"/>
<DataMethodHistory Timestamp="Blaze Advisor -10-15T08:22:06.253-
08:00">
    <Name>dm_Root_ConcatenateVipNumber</Name>
    <DataType>string</DataType>
    <Value>1234</Value>
    <DataMethodInput>
    <Name>xString</Name>
    <DataType>string</DataType>
    <Value>34</Value>
```

```
    </DataMethodInput>
</DataMethodHistory>

</Application>
```

Figure 5: The output created when the data method is called at runtime

3. Complex Method, No Input Variables/No Local Variables

Table 5: Data Method Editor Input

Item	Value
Data Method Name	dmo_App_GetCurrentAddress
Description	Get current address for this applicant.
Context (Path+Element)	Application.Applicant
Result Type	ApplicantAddress
Save History	checked
Input Variables	none
Local Variables	xAppCurAddr

Expression	`result := null;`
	
	`for every Application.Applicant.ApplicantAddress relative to Applicant do`
	`  if (AddressStatusIndicator = "Current") then`
	`  xAppCurAddr := ApplicantAddress;`
	`  endif`
	`endfor`
	
	`result := xAppCurAddr;`

Method Calling

This method can be called from rules or other methods, but not from mappings. It returns a copy of the current applicant address. Here is an example of calling this method from another method.

Table 6: Data Method Call

Item	Value		
Local Variables	**Name**	**Data Type**	**Initial Value**
	xAppAddress	ApplicantAddress	null
Expression	`xAppAddress := dmo_App_GetCurrentAddress;` `SomeField := xAppAddress.City;`		

Sample XML Input Document

```
<Application>
    <Applicant>
        <ApplicantAddress AddressStatusIndicator="Current"
AddressType="Home" StreetNumber="256" StreetName="Morton"
StreetType="Dr" StreetPostDirection="NW" City="Sandisfield"
```

```
State="MA" PostalCode="01255" PostalPlusCode="0413"
County="Berkshire" CountryCode="USA" ResidenceType="Single
Family Home" ResidentialStatusIndicator="Owns (Financed)"
MonthsAtAddress="26" StartDate="2004-02-15" EndDate=""
VerifiedFlag="false"/>
        <ApplicantAddress AddressStatusIndicator="Previous"
AddressType="Home" UnparsedStreetAddress="1500 Calveros Blvd
#13" City="Sheffield" State="MA" PostalCode="01257"
PostalPlusCode="5588" County="" CountryCode="USA"
ResidenceType="Condominium" ResidentialStatusIndicator="Rents"
MonthsAtAddress="50" StartDate="1999-12-01" EndDate="2004-02-14"
VerifiedFlag="true" VerificationType="Phone bill"
VerificationDate="Blaze Advisor -08-09"/>
        <ApplicantAddress AddressStatusIndicator="Previous"
AddressType="Home" UnparsedStreetAddress="7855 Atwood Ave. #6"
City="Sheffield" State="MA" PostalCode="01256" PostalPlusCode=""
County="" CountryCode="USA" ResidenceType="Apartment"
ResidentialStatusIndicator="Rents" MonthsAtAddress="10"
StartDate="1999-01-15" EndDate="1999-11-30" VerifiedFlag="true"
VerificationType="Phone bill" VerificationDate="Blaze Advisor -08-
09"/>
    </Applicant>
</Application>
```

Figure 6: An input document containing a current address and previous addresses

Saved Data Method History Output

Only the current address is returned as the result of executing the data method.

```
<Application>
    <Applicant>
        <DataMethodHistory Timestamp="Blaze Advisor -11-
02T08:22:06.253-08:00">
            <Name>dmo_App_GetCurrentApplicantAddress</Name>
            <DataType>ApplicantAddress</DataType>
            <Value>&lt;ApplicantAddress
AddressStatusIndicator="Current" AddressType="Home"
StreetNumber="256" StreetName="Morton" StreetType="Dr"
StreetPostDirection="NW" City="Sandisfield" State="MA"
PostalCode="01255" PostalPlusCode="0413" County="Berkshire"
CountryCode="USA" ResidenceType="Single Family Home"
ResidentialStatusIndicator="Owns (Financed)"
MonthsAtAddress="26" StartDate="2004-02-15" EndDate=""
VerifiedFlag="false"&gt; &lt;/ApplicantAddress&gt;</Value>
        </DataMethodHistory>
    </Applicant>
</Application>
```

Figure 7: The output created when the data method is called at runtime

4. Complex Method, One Input Variable/One Local Variable

Table 7: Data Method Editor Input

Item	Value		
Data Method Name	dmo_App_GetPreviousApplicantAddress_ByIndex		
Description	Returns a copy of applicant's Nth previous address		
Context (Path+Element)	Application.Applicant		
Type	ApplicantAddress		
Save History	checked		
Input Variables	**Name**	**DataType**	**Initial Value**
	xIndex	numeric	n/a
Local Variables	**Name**	**DataType**	**Initial Value**
	xCount	numeric	0

Expression	`result := null;`
	`for every`
	`Application.Applicant.ApplicantAddress`
	`relative to Applicant do`
	`if (AddressStatusIndicator = "Previous") then`
	`  xCount := xCount + 1;`
	`  if ( xCount = xIndex ) then`
	`  result := ApplicantAddress;`
	`  endif`
	`endif`
	`endfor`		

Method Calling

This method can be called from other methods, but not from data method sequences, mappings, or rules. It returns a copy of the `ApplicantAddress`. Here is an example of calling this method from another method.

Table 8: Data Method Call

Item	Value		
Local Variables	**Name**	**Data Type**	**Initial Value**
	`xAppAddress`	`ApplicantAddress`	`null`
Expression	`xAppAddress := dmo_App_GetPreviousApplicantAddress_ByIndex(1);` `SomeField := xAppAddress.City;`		

Sample XML Input Document

The sample input for this example is the same as for the previous example (Figure 10 on page 26).

Saved Data Method History Output

The Index value of 1 was set in the method call. On execution, two previous addresses were counted.

```
<Application>
    <Applicant>
        <DataMethodHistory Timestamp="Blaze Advisor -11-
03T08:22:06.253-08:00">
            <Name>dmo_App_GetPreviousApplicantAddress_ByIndex</
Name>
            <DataType>ApplicantAddress</DataType>
            <Value>&lt;ApplicantAddress
AddressStatusIndicator="Previous" AddressType="Home"
UnparsedStreetAddress="1500 Calveros Blvd #13" City="Sheffield"
State="MA" PostalCode="01257" PostalPlusCode="5588" County=""
CountryCode="USA" ResidenceType="Condominium"
ResidentialStatusIndicator="Rents" MonthsAtAddress="50"
StartDate="1999-12-01" EndDate="2004-02-14" VerifiedFlag="true"
VerificationType="Phone bill" VerificationDate="Blaze Advisor -08-
09"&gt;&lt;/ApplicantAddress&gt;</Value>
            <DataMethodInput>
                <Name>xIndex</Name>
                <DataType>numeric</DataType>
                <Value>1</Value>
            </DataMethodInput>
            <DataMethodOutput>
                <Name>xCount</Name>
                <DataType>numeric</DataType>
                <Value>2</Value>
            </DataMethodOutput>
        </DataMethodHistory>
    </Applicant>
</Application>
```

Figure 8: The output created when the data method is called at runtime

Processing History Output

The `ProcessingHistory` output node contains information on the product strategy and decision flow components that were executed during transaction processing. You can use this information to help develop and maintain product strategies in your development and production environments. Table 9 describes the elements that appear in this node.

Table 9: ProcessingHistory Elements

Element	Purpose
DataMethodSequenceHistory	Exists for each data method sequence that was executed during decision flow or decision tree processing. Indicates the data method sequence name.
DecisionFlowHistory	Exists for each decision flow that was executed during transaction processing. Its descendants contain information on the decision flow components (steps) executed during processing of a decision flow.
DecisionFlowStepHistory	Exists for each decision flow component (step) executed during processing of a decision flow. Indicates the step's name, index, implementation type, implementation name, result, start timestamp, and stop timestamp.
DecisionHistory	Exists for each decision registered during transaction processing. Indicates the decision result (level). Decisions can be registered during processing of rulesets, decision scenarios, score models, and decision trees.
DecisionReason	Exists for each decision reason associated with a ruleset decision, decision scenario, score model, or decision tree.
DecisionScenarioHistory	Exists for each decision scenario that was executed during execution of a decision tree. Indicates the decision scenario name.
DecisionTableHistory	Exists for each decision table that was executed during decision flow or decision tree processing. Indicates the table name and result.
DecisionTreeHistory	Exists for each decision tree that was executed during decision flow or decision tree processing. Indicates the decision tree result.
FinalApplicationDecision	(Reserved for future use) Contains information on the final application decision of the calling system (as opposed to the application decision recommended by Decision Accelerator). This object is not generated by Decision Accelerator at runtime.
FinalDecisionHistory	Exists if a a final decision was reached for the application. Indicates the final decision result.
FinalOfferHistory	(Reserved for future use) This object is not generated by Decision Accelerator at runtime.
OfferHistory	Exists for each offer created during execution of a pricing scenario. Indicates the name, rate, credit limit, and term of the offer.
PricingScenarioHistory	Exists for each pricing scenario that was executed during execution of a decision tree. Its descendants contain offer and final offer information.
RuleHistory	Exists for each rule that fired during decision flow processing. Indicates the rule name, reason associated with the rule, and rank order of the reason.
RulesetHistory	Exists for each ruleset executed during decision flow processing. Indicates the ruleset name, ruleset result, and total severity of rules fired in the ruleset.

Table 9: ProcessingHistory Elements (continued)

Element	Purpose
ScoreModelCharHistory	Exists for each scored characteristic evaluated in a score model. Indicates the name, bin, and partial score of the characteristic.
ScoreModelHistory	Exists for each score model executed during decision flow or scoring scenario processing. Indicates the score model name, final score, and odds.
ScoreModelRowHistory	Exists for each score model information row in a scoring scenario. Indicates the row index, its cutoff range, and whether the row was the first one executed in a scenario. Its descendants contains score model, characteristic, and candidate decision information.
ScoringScenarioHistory	Exists for each scoring scenario that was executed during execution of a decision tree. Indicates the scoring scenario name and decision result (level). Its descendants contain information on score model information rows.

See Also Refer to the data model spreadsheet for detailed information on the elements and attributes in the `ProcessingHistory` node.

Using User-Defined Fields

You can use `UserDefinedField` objects for data that doesn't have a home in the data model provided with Decision Accelerator.

User-defined fields can be used for:

- Data submitted by the calling system on input and returned on output without being accessed by Decision Accelerator
- Data submitted by the calling system on input and accessed by Decision Accelerator within a decision flow
- Data created and/or used by Decision Accelerator within a decision flow; this data can be returned to the calling system if desired

All objects in the IBS data model used for Decision Accelerator contain `UserDefinedField` children, with the following exceptions:

- `DataMethodHistory` objects and their descendants
- `UserDefinedField` objects and their descendants

As described in "Complex and Simple Elements" on page 64, simple elements (which contain content such as report images, ARF text, or reason text) do not correspond to objects in the data model. Therefore, they do not contain `UserDefinedField` children. No schema modifications are needed for adding, deleting, or modifying `UserDefinedField` children of objects in the provided data model; however, if you customize the data model with new objects, you must add `UserDefinedField` children to the new objects if you wish to set up user-defined fields on those objects.

UserDefinedField elements are not displayed in the Outline or DataModel tabs of the data model spreadsheet. Instead, they appear in the UserDefinedField tab. The paths for UserDefinedField and its children are shown below.

Paths:

- <ParentElement>/UserDefinedField
- <ParentElement>/UserDefinedField/Name
- <ParentElement>/UserDefinedField/DataType
- <ParentElement>/UserDefinedField/Value

<ParentElement> is a placeholder for any element that can contain a UserDefinedField object. The value of the Value element must match the data type listed in the DataType element. The data types allowed for user-defined fields are:

- boolean
- date
- duration
- integer
- real
- string
- time
- timestamp

Note The actual values in the Name, DataType, and Value elements are strings.

User-Defined Fields Submitted by the Calling System

On input, user-defined fields allow you to associate a name with a value at runtime. The Name, DataType, and Value children of the UserDefinedField element must be populated. Some examples of UserDefinedField elements are shown in Figure 13.

```
<Application>
    <Applicant>
        <UserDefinedField>
            <Name>Customer Value Indicator</Name>
            <DataType>string</DataType>
            <Value>Gold</Value>
        </UserDefinedField>
        <UserDefinedField>
            <Name>In-house Deposit Balance</Name>
            <DataType>real</DataType>
            <Value>785.50</Value>
        </UserDefinedField>
    </Applicant>
    <Business>
        <UserDefinedField>
            <Name>Admin Cost Percentage</Name>
            <DataType>integer</DataType>
            <Value>5</Value>
        </UserDefinedField>
        <UserDefinedField>
            <Name>Non-Profit Indicator</Name>
```

```
            <DataType>boolean</DataType>
            <Value>true</Value>
        </UserDefinedField>
    </Business>
    <UserDefinedField>
        <Name>Promotion Name</Name>
        <DataType>string</DataType>
        <Value>Tent Sale</Value>
    </UserDefinedField>
    <UserDefinedField>
        <Name>Promotion End Date</Name>
        <DataType>date</DataType>
        <Value>Blaze Advisor -02-12</Value>
    </UserDefinedField>
</Application>
```

Figure 9: Sample UserDefinedField nodes at various levels

Documenting Your User-Defined Fields

You are responsible for maintaining documentation of any user-defined XML structures used in your transactions. For example, if your transaction contains some codes in `UserDefinedField` elements, you would need to maintain a list of the code meanings.

```
<UserDefinedField
   <Name>Code1</Name>
   <DataType>string</DataType>
   <Value>A52</Value>
</UserDefinedField>
<UserDefinedField
   <Name>Code2</Name>
   <DataType>string</DataType>
   <Value>B97</Value>
</UserDefinedField>
```

Figure 10: An example of user-defined fields that need to be documented

Table 10 shows an example of how the user-defined field values in Figure 14 could be documented.

Table 10: Sample User-Defined Field Documentation

Code	Meaning
A52	Fraud alert
B97	Security attention

As another example, if your transactions contain user-defined fields with duration values, you might need to document the units of the duration, such as months or years.

```
        <UserDefinedField>
            <Name>TimeSinceDefault</Name>
            <DataType>real</DataType>
            <Value>5.5</Value>
```

```
        </UserDefinedField>
        <UserDefinedField>
            <Name>TimeSinceLastPayment</Name>
            <DataType>integer</DataType>
            <Value>8</Value>
</UserDefinedField>
```

Figure 11: User-defined fields with duration values

Table 11 shows an example of how the user-defined field values in Figure 15 could be documented.

Table 11: Sample User-Defined Duration Field Documentation

Element	Data Type	Units	Range
TimeSinceDefault	real	Years	0 - 99
TimeSinceLastPayment	integer	Months	0 - 120

Alternately, you could imply units in the `Name` attributes that you assign, as shown in Figure 16.

```
        <UserDefinedField>
            <Name>YearsSinceDefault</Name>
            <DataType>real</DataType>
            <Value>5.5</Value>
        </UserDefinedField>
        <UserDefinedField>
            <Name>MonthsSinceLastPayment</Name>
            <DataType>integer</DataType>
            <Value>8</Value>
</UserDefinedField>
```

User-Defined Field Data Accessed by Decision Accelerator

In order for Decision Accelerator to access a user-defined field, there must be a pre-existing agreement between the calling system owners and the Decision Accelerator owners as to the user-defined field location, name, data type, and possible values.

The actual data type of the `Value` element is always a string, even if the value of the `DataType` element is numeric, timestamp, or duration. In this case, you must use a data method to convert the value from string to the desired type.

This does not apply to data object references, since they simply return the object. If the data needs to be type converted, a data method is the best choice for selecting and converting the value.

Using User-Defined Fields in Data Methods

Most of the Decision Accelerator decision flow components where you will use user-defined field values (for example, rules and mappings) cannot call data methods that take parameters, so you will need to create a unique data method for each user-defined field `Name` and `DataType` pair, for each parent data type (object type).

Figures 17 hrough 20 show examples of various types of user-defined fields; tables 12 through 15 show examples of data methods that use them.

```
<Application>
    <Applicant>
        <UserDefinedField>
            <Name>ABC</Name>
            <DataType>string</DataType>
            <Value>High priority</Value>
        </UserDefinedField>
    </Applicant>
</Application>
```

Figure 12: A string user-defined field

Table 12: Data Method Example for Figure 17

Item	Value		
Data Method Name	dm_Obj_UDF_GetStringABC		
Context (Path+Element)	Application.Applicant		
Result Type	string		
Input Variables	none		
Local Variables	**Name**	**Data Type**	**Initial Value**
	xString	string	null

Expression	`result := null;`
	`for every Application.Applicant.UserDefinedField relative to Applicant do`
	`  if ( ( Name = "ABC") and ( DataType = "string") ) then`
	`  xString := Value;`
	`  endif`
	`endfor`
	`result := xString`		

```
<Application>
    <Applicant>
        <UserDefinedField>
            <Name>DEF</Name>
            <DataType>numeric</DataType>
            <Value>22.35</Value>
        </UserDefinedField>
    </Applicant>
</Application>
```

Figure 13: A numeric user-defined field

Table 13: Data Method Example for Figure 18

Item	Value		
Data Method Name	dm_Obj_UDF_GetNumericDEF		
Context (Path+Element)	Application.Applicant		
Result Type	numeric		
Input Variables	none		
Local Variables	**Name**	**Data Type**	**Initial Value**
	xString	string	null

Expression	`result := null;`
	`for every`
	`Application.Applicant.UserDefinedField`
	`relative to Applicant do`
	`if ( ( Name = "DEF") and ( DataType =`
	`"numeric") ) then`
	`xString := Value;`
	`endif`
	`endfor`
	`if (xString <> null) then result :=`
	`Val(xString); endif`		

```
<Application>
    <Applicant>
        <UserDefinedField>
            <Name>GHI</Name>
            <DataType>timestamp</DataType>
            <Value>Blaze Advisor -10-17T13:17:54"</Value>
        </UserDefinedField>
    </Applicant>
</Application>
```

Figure 14: A timestamp user-defined field

Table 14: Data Method Example for Figure 19

Item	Value
Data Method Name	dm_Obj_UDF_GetTimestampGHI
Context (Path+Element)	Application.Applicant
Result Type	timestamp
Input Variables	none

Table 14: Data Method Example for Figure 19 (continued)

Item	Value		
Local Variables	**Name**	**Data Type**	**Initial Value**
	xString	string	null

Expression	`result := null;`
	`for every`
	`Application.Applicant.UserDefinedField`
	`relative to Applicant do`
	`if ( ( Name = "GHI") and ( DataType =`
	`"timestamp") ) then`
	`xString := Value;`
	`endif`
	`endfor`
	`if (xString <> null) then result :=`
	`TimestampFromStr(xString); endif`		

```
<Application>
    <Applicant>
        <UserDefinedField>
            <Name>JKL</Name>
            <DataType>date</DataType>
            <Value>Blaze Advisor -10-27</Value>
        </UserDefinedField>
    </Applicant>
</Application>
```

Figure 15: A date user-defined field

Table 15: Data Method Example for Figure 20

Item	Value		
Data Method Name	dm_Obj_UDF_GetDateJKL		
Context (Path+Element)	Application.Applicant		
Result Type	date		
Input Variables	none		
Local Variables	**Name**	**Data Type**	**Initial Value**
	xString	string	null

Expression	`result := null;`
	`for every Application.Applicant.UserDefinedField`
	`relative to Applicant do`
	`if ( ( Name = "JKL") and ( DataType = "date") ) then`
	`xString := Value;`
	`endif`
	`endfor`
	`if (xString <> null) then result :=`
	`TimestampToDate(TimestampFromStr(xString)); endif`		

```
<Application> - Removed for now, as there is no DurationFromStr
```

```
function.
    <Applicant>
        <UserDefinedField>
            <Name>MNO</Name>
            <DataType>duration</DataType>
            <Value>5 months</Value>
        </UserDefinedField>
    </Applicant>
</Application>
```

Figure 16: A duration user-defined field

Table 16: Data Method Example for Figure 21

Item	Value		
Data Method Name	dm_Obj_UDF_GetDurationMNO		
Context (Path+Element)	Application.Applicant		
Result Type	duration		
Input Variables	none		
Local Variables	**Name**	**Data Type**	**Initial Value**
	xString	string	null

Expression	`result := null;`
	`for every`
	`Application.Applicant.UserDefinedField`
	`relative to Applicant do`
	`if ( ( Name = "MNO") and ( DataType = "duration") ) then`
	`xString := Value;`
	`endif`
	`endfor`
	`if (xString <> null) then result := DurationFromStr(xString); endif`		

Using User-Defined Fields in Data Object References

Tables 17 through 19 show examples of data object references that work with user-defined fields. Note that data object references currently cannot use parameters.

Table 17: Data Object Reference Example 1—Application UDF

Item	Value		
DOR Name	dor_Root_UDF_XYZ		
Context (Path+Element)	Application.UserDefinedField		
Conditions	**Field**	**Operator**	**Value | Parameter**
	Name	=	“XYZ”
	DataType	=	“String”
Description	A simple data object reference that returns all Application UserDefinedField objects with a name (and type).		
Issues	This data object reference should not be used if multiple UserDefinedFields exist with the same key data.		

Table 18: Data Object Reference Example 2—Applicant UDF

Item	Value		
DOR Name	dor_Applicant_UDF_XYZ		
Context (Path+Element)	Application.Applicant.UserDefinedField		
Conditions	**Field**	**Operator**	**Value | Parameter**
	Name	=	“XYZ”
	DataType	=	“String”
Description	A simple data object reference that returns all Application.Applicant. UserDefinedField objects with a name (and type).		
Issues	This data object reference should not be used if multiple UserDefinedFields exist with the same key data, or if there are multiple applicant objects, each of which could have a UserDefinedField with the same key.		

Table 19: Data Object Reference Example 2—Primary Applicant UDF

Item	Value		
DOR Name	dor_Applicant_PrimaryUDF_XYZ		
Context (Path+Element)	Application.Applicant.UserDefinedField		
Conditions	**Field**	**Operator**	**Value | Parameter**
	Name	=	“XYZ”
	DataType	=	“String”
	Applicant.ApplicantType=		“Primary”
Description	A simple data object reference that returns all Application.Applicant UserDefinedField objects with a name (and type).		
Issues	This data object reference should not be used if multiple UserDefinedFields exist with the same key data. It includes hardcoded conditions to check for the Primary applicant. This data object reference would not be reusable for other applicant types, but could be copied and edited for each unique type.		

Using User-Defined Fields in Rules

When trying to use user-defined field data in *rule conditions*, keep in mind that it will *always* be a good choice to write a data method to access the user-defined field value. You can ask the following questions to identify whether the user-defined field can be used directly or if a data method or data object reference will be needed:

- Will more than one user-defined field be needed? If so, then data methods are the only choice to access the user-defined field values.
- Does `DataType` equal “`string`” for the user-defined field? If so, then the value can be used directly by the rule condition. Otherwise, you will need to use a data method to convert the `Value` field (which is always a string) into the desired data type.
- Is a data object reference being used? If so, you can create a new data object reference (based on the old one) that goes down one more level to the user-defined field, and include in that data object reference all the conditions in the original data object reference plus conditions on the user-defined field key fields.

- Is this a complex rule? If the conditions of the rule are complex, it is easiest to use a data method to access the user-defined field value.
- Does this rule fire on *any* condition? If so, then a data method is the only choice.

Using User-Defined Fields in Mappings

Given the fact that a mapping needs to be very specific as to which instance of an object is being referenced, if you cannot write the data object reference for the user-defined field, then a data method must be used.

User-Defined Fields Created or Populated by Decision Accelerator

In order for Decision Accelerator to create user-defined fields, there must be a pre-existing agreement between the calling system owners and the Decision Accelerator owners as to the user-defined field location, name, data type, and possible values.

Table 20 illustrates a data method that sets the value of a `UserDefinedField` child of the `Applicant` object to the value of the `iValue` input variable, if the `UserDefinedField` has a `Name` value of "`Code1`".

Table 20: Data Method Example 1—Set String Code1

Item	Value		
Data Method Name	dm_App_UDF_SetCode1		
Context (Path+Element)	Application.Applicant		
Result Type	string		
Input Variables	**Name**	**Data Type**	**Initial Value**
	iValue	string	n/a
Local Variables	**Name**	**Data Type**	**Initial Value**
	xUDF	UserDefinedField	null

Expression	`result := null;`
	`if (iValue <> null) then`
	`// check for existing UDF with the key data.`
	`for every`
	`Application.Applicant.UserDefinedField`
	`relative to Applicant do`
	`  if ( ( Name = "Code1") and ( DataType = "string") )`
	`  then`
	`  xUDF := UserDefinedField;`
	`  endif`
	`endfor`
	`if (xUDF = null) then`
	`  xUDF := New UserDefinedField;`
	`  xUDF.Value := iValue;`
	`  Attach (Applicant, xUDF)`
	`else`
	`  xUDF.Value := iValue;`
	`endif`
	`endif`		

Table 21 illustrates a data method that creates a new user-defined field as a child of the `Application.Applicant` object. The data method illustrated in Table 22 calls the data method in Table 21 to set the value of the user-defined field with the value generated by a third data method. The `Name` (`ScoringTimestamp`) and `DataType` (`timestamp`) of the `UserDefinedField` are passed as parameters.

Table 21: Data Method Example 2—Set String XYZ by Name and Type

Item	Value		
Data Method Name	dm_App_UDF_SetXYZ_ByNameType		
Context (Path+Element)	Application.Applicant		
Result Type	string		
Input Variables	**Name**	**Data Type**	**Initial Value**
	iName	string	n/a
	iDataType	string	n/a
	iValue	string	n/a
Local Variables	**Name**	**Data Type**	**Initial Value**
	xUDF	UserDefinedField	null

Expression	`result := null;`
	`if (iValue <> null) then`
	`// check for existing UDF with the key data.`
	`for every`
	`Application.Applicant.UserDefinedField`
	`relative to Applicant do`
	`  if ( ( Name = iName) and ( DataType = iDataType)  ) then`
	`  xUDF := UserDefinedField;`
	`  endif`
	`endfor`
	`if (xUDF = null) then`
	`  xUDF := New UserDefinedField;`
	`  xUDF.Name := iName;`
	`  xUDF.DataType := iDataType;`
	`  xUDF.Value := iValue;`
	`  Attach (Applicant, xUDF)`
	`else`
	`  xUDF.Value := iValue;`
	`endif`
	`endif`		

Table 22: Data Method Example 3—Set a Specific UDF

Item	Value
Data Method Name	dm_App_SetSpecificUDF
Context (Path+Element)	Application.Applicant
Result Type	string
Input Variables	none

Table 22: Data Method Example 3—Set a Specific UDF (continued)

Item	Value		
Local Variables	**Name**	**Data Type**	**Initial Value**
	xValue	string	null

Expression	`result := null;`
	`/* Assume we want to set a UDF with a Name = "ScoringTimestamp", a DataType of timestamp, and a value from some other method dm_App_ScoringTime */`
	
	`xValue := TimestampFormat(dm_AppScoringTime,"yyyy-MM-dd'T'HH:mm:ss");`
	`if (xValue <> null) then dm_App_UDF_SetXYZ_ByNameType( "ScoringTimestamp", "timestamp", xValue);`
	`endif`		

Using Data Object References to set User-Defined Fields

If the user-defined field object needs to be created, there is no value in using a data object reference to set its value. The data object reference could only be used to reset the value. As long as a data method was used to create the object, it would make more sense to set the value using the data method.

Using Mappings to set User-Defined Fields

A user-defined field value cannot be set directly from a table mapping in the SMW.

Using Data Method Sequences to set User-Defined Fields

Sequences cannot pass a value parameter to a data method, so then they cannot set a user-defined field without calling a method that creates and sets the user-defined field.

Resubmitting Transactions

To indicate that a transaction is a resubmission, you must enter a value of "true" for the `Application/@ResubmissionFlag`. To determine which XML nodes can be passed as resubmission input, refer to the **DA Resub In** and **DA+IDM Resub In** columns of the data model spreadsheet. For example, for a regular Decision Accelerator transaction, the `DecisionResponse` node is output only. For a Decision Accelerator resubmission transaction, the DecisionResponse node can be submitted on input.

Resubmitted applications can contain new data, such as consumer credit report data retrieved during the first pass. Decision Accelerator does not make any assumptions about resubmitted data; risk management analysts must configure the decision flow(s) and the data methods used in the decision flow(s) to handle the resubmitted data according to your own requirements. The decision flows delivered with Decision Accelerator contain *steps* that handle requests for external data (from CRAs or the FICO® Expansion score service, for example). Because a resubmission transaction could

already include such data, the decision flows contain steps that check to see if the data already exists; if it does, the decision flow does not re-request it.

When using a *champion decision flow*/*challenger decision flow*, you should submit the `StrategySelectionRandomNumber` value from the first pass, to route the application down the same decision flow. If the calling system submits any products in the `DecisionResponse/Product` node of a resubmission, Decision Accelerator sets their `ProductStatusIndicator` to "Old" to indicate that, upon return, they were not created in the latest call to Decision Accelerator. Decision Accelerator will ignore any products that are marked as "Old".

Resubmissions can facilitate requests for multiple products. If you want to process applications against all products configured in Decision Accelerator, you cannot use the cross-sell feature, because the product decision must always be "Approve" for a cross-sell to be invoked. Instead, you can send in an application multiple times, to run against each product. You can send in additional data gathered during one of the original submissions, such as consumer credit report data, to avoid pulling it again when running against the rest of the products.

Internal Services

Internal services let you enhance Decision Accelerator functionality with Fair Isaac models and other business logic. Before calling systems can submit transactions requiring internal service data to Decision Accelerator, SMW users must configure decision flows to access the desired internal service. SMW users must provide calling system developers with information about input data required and returned by the internal service.

To use an internal service in a decision flow, SMW users must follow these steps

1. Implement the Blaze Advisor internal service project so that it is accessible to Decision Accelerator, along with any data transformations that may be required.
2. Create a definition for this internal service in the **Internal Service Properties Page**.
3. Create data methods that prepare an internal service request object, invoke the internal service, and then handle the response.
4. Create a data method sequence to execute these data methods.
5. In a decision flow, create a *decision flow step* whose *step implementation* is the data method sequence.
6. Use this decision flow in a product.

Pre-defined Product Strategy Internal Service Requests

Some pre-defined product strategies delivered with Decision Accelerator contain a pre-configured internal service request for Blaze Advisor data. The strategies use pre-configured data methods to construct the request to the Blaze Advisor project; they also use pre-configured data transformations. If you want to invoke other types of projects, you will need to create new data methods and internal service calls based on the delivered data methods and internal service call.

Important The Blaze Advisor project called from the UsedCarLoanStrategy does not contain an actual Blaze Advisor model. It *must not* be used to produce <REMOVED>s on applicants in a development or production environment. It is a placeholder for the actual Blaze Advisor project, which must be customized for each client. Once an actual Blaze Advisor project is installed, a decision flow author will need to modify the internal service that calls it and modify the data method that invokes the internal service. For more information, contact your Fair Isaac account representative.

Adding New Internal Service Projects

At this time, Decision Accelerator can only support *Blaze Advisor projects* that conform to the following:

- The project must have an entry point that accepts a string and returns a string.
- The string received by the entry point must be one of the following:
  - An XML document whose schema matches the Decision Accelerator XML schema or a subset of the Decision Accelerator XML schema.
  - An XML document or other string derivable from executing an XSL transformation on the above type of document.

  Similar constraints apply to the output argument. The output transformation does not require the XML document sent to the entry point, just the response.
- The project must be deployed using the Blaze Advisor QuickDeployer, using the Java deployment option.
- The deployment's `.server` file must already be installed in the Services installation folder.

If transformations are applied to the input that Decision Accelerator submits, the resulting XML must conform to the data format expected by the internal service; likewise, transformations applied to the output of the internal service must accommodate the data format produced by the internal service. The transformation files must be placed in the Services installation folder, as noted in the "Web and Internal Services" chapter of the *Capstone Decision Accelerator User's Guide*.

Important When you deploy an internal service project, Blaze Advisor writes the absolute or relative locations of the associated files (such as .adb, class, .jar, or repository files) to the `.server` file. When it calls the internal service, Decision Accelerator attempts to access the files based on the locations specified in the `.server` file. You must ensure that each associated file has been installed in the correct location—this may be directly in the `Services` folder, in a subfolder, or in any other location specified when using the QuickDeployer.

Web Services

Web services allow Decision Accelerator to access various sources of external data. Before calling systems can submit transactions requiring Web service data to Decision Accelerator, SMW users must configure decision flows to access the desired Web service. SMW users must provide calling system developers with information about input data required and returned by the Web service.

To use a Web service in a decision flow, SMW users must follow these steps

1. Obtain connection information for the Web service, along with any data transformations that may be required.
2. Create a definition for this Web service in the **Web Service Properties Page**.
3. Create data methods that prepare a Web service request object, invoke the Web service, and then handle the response.
4. Create a data method sequence to execute these data methods.
5. In a decision flow, create a decision flow step whose implementation is the data method sequence.
6. Use this decision flow in a product.

Pre-defined Product Strategy Web Service Calls

Decision Accelerator can submit multiple Intelligent Data Manager data service requests during a single transaction. Some pre-defined product strategies delivered with Decision Accelerator use pre-configured Web service requests to Intelligent Data Manager to obtain consumer credit report (CRA) data and FICO® Expansion score data. The strategies use pre-configured data methods to construct data service request objects and submit the data service request to Intelligent Data Manager; they also use pre-configured data transformations. If you want to submit requests for other types of data from Intelligent Data Manager, you will need to create new data methods and Web service calls based on the delivered data methods and Web service calls.

Adding New External Web Services

Decision Accelerator can support both synchronous and asynchronous Web service calls. The Web service must accept a single string argument which must be one of the following:

- An XML document whose schema matches the Decision Accelerator XML schema or a subset of the Decision Accelerator XML schema.
- An XML document or other string derivable from executing an XSL transformation on the above type of document.

Similar constraints apply to the output argument. The output transformation may involve just the Web service's response, or a "merge" of the Web service's response and the original request. For asynchronous calls, Decision Accelerator does not wait for or process any response that is returned from the Web service.

If transformations are applied to the input that Decision Accelerator submits, the resulting XML must conform to the data format expected by the Web service; likewise, transformations applied to the output of the Web service must accommodate the data format produced by the Web service and must conform to the data format expected by Decision Accelerator. The transformation files must be placed in the services installation folder, as noted in the "Web and Internal Services" chapter of the *Capstone Decision Accelerator User's Guide*.

Updating the Data Repository with Final Decision and Offer Data

At runtime, the Process Server sends XML transaction data to the DSS. The FinalApplicationDecision is the final application decision of the calling system (as opposed to the application decision recommended by Decision Accelerator). This data is not provided by Decision Accelerator and must be provided by the calling system.

At runtime, the Process Server sends XML transaction data to the DSS. The following types of data are not provided by Decision Accelerator and must be provided by the calling system:

- **Final Application Decision**—The final application decision of the calling system (as opposed to the application decision recommended by Decision Accelerator).
- **Final Offer History**—The actual offer made to the credit applicant (as opposed to the offer recommended by Decision Accelerator).

You may wish to add this data to the data repository used by the DSS for use in reports according to your own requirements. This section contains some sample SQL queries that you can modify and use to update the data repository reporting tables.

Final Application Decision

Each application is recorded by the DSS each time it is submitted for processing. The data repository will contain a timestamped `DATransaction` record and a number of `DecisionFlowHistory` records for each time an application is submitted.

One option is to update the `DecisionFlowHistory` for the decision associated with the *latest* transaction processed for the application:

```
UPDATE decisionflowhistory1 AS target
INNER JOIN (
SELECT tran. hncid, dfh. hncid AS dfhid, superid
FROM datransaction1 AS tran
inner join decisionflowhistory1 AS dfh ON tran.hncid =
dfh.superid
WHERE tran.SystemId = "sysid"
AND tran.ApplicationCrossReferenceId = "APPREF"
AND dfh.ProductCode = "PRODUCT"
AND dfh.DecisionFlowName = "DECISONFLOW"
AND dfh.ProductCategory = "CATEGORY"
ORDER BY ProcessingTimestamp DESC
LIMIT 1
) AS source ON source.dfhid = target.hncid AND source.superid =
target.superid
```

```
SET FinalDecisionResult = "XXXXXX";
```
You may prefer to update the final decision for all occurrences of the `DecisionFlowHistory`:

```
UPDATE decisionflowhistory1 INNER JOIN datransaction1 ON
datransaction1.hncid = decisionflowhistory1.superid
SET FinalDecisionResult = "XXXXX"
```

```
WHERE datransaction1.SystemId = "sysid"
AND datransaction1.ApplicationCrossReferenceId = "REF"
AND decisionflowhistory1.ProductCode = "PROD"
AND decisionflowhistory1.DecisionFlowName = "DECISIONFLOW"
     AND decisionflowhistory1.ProductCategory = "CATEGORY";
```

Note Substitute values for `SystemId`, `ApplicationCrossReferenceId`, `ProductCode`, `ProductCategory`, `FinalDecisionResult`, and `DecisionFlowName` as appropriate.

Using Decision Accelerator with Intelligent Data Manager

Some nodes in the IBS data model apply only to Decision Accelerator. For example, the `DecisionResponse` output node is not applicable to Intelligent Data Manager-only transactions, because Intelligent Data Manager does not perform decisioning. Other nodes apply only to Intelligent Data Manager. For example, for non-resubmission input, the `CraDataServiceRequest`, `BbDataServiceRequest`, `FesDataServiceRequest`, and `SbfeDataServiceRequest` children of `ExternalDataSource` are only used when the calling system submits transactions directly to Intelligent Data Manager.

Some nodes apply only to situations where Decision Accelerator and Intelligent Data Manager are used in conjunction with each other. For example, for transactions in which Decision Accelerator makes a request to Intelligent Data Manager for consumer credit report data, calling systems can submit data to Decision Accelerator in the `CraRequestDefaults` node. Decision Accelerator must be configured to accept this type of data and use it to construct the `DataServiceRequest` node required by the Intelligent Data Manager Web service.

Request Object Construction

When Decision Accelerator makes are request to Intelligent Data Manager, it must construct and submit the same type of `DataServiceRequest` object that a calling system would submit if it were calling Intelligent Data Manager directly. For example, for a consumer credit report data request, the `DataServiceRequest` object contains the following items:

- `DataflowName (mandatory)`
- `RequestedProductType (optional)`
- `ReportInstructions`
- `RequestedResponse`
- `RequestedCra`
- `InquiryParameters`
- `AnalysisParameters`
- CraHousehold
- CraApplicant

Sources of DataServiceRequest Data

When building decision flows, you can use data methods to populate the `DataServiceRequest` node. The data model spreadsheet contains information on which `DataServiceRequest` nodes must be populated for requests to Intelligent Data Manager.

The data used to populate the `DataServiceRequest` node can come from several sources:

1. Areas of input XML documents other than `ExternalDataSource`. The pre-defined product strategies supplied with Decision Accelerator use data methods that perform this type of task for consumer credit report and FICO® Expansion score requests.
2. The `RequestDefault` areas of input XML documents (as mentioned above). The pre-defined product strategies supplied with Decision Accelerator *do not* contain examples of data methods that perform this type of task. You will need to add them if you want calling systems to submit data in `RequestDefault` nodes.
3. Values hard-coded into data methods. If calling systems will not pass data in the request defaults sections of input XML documents, you can use this option. The pre-defined product strategies supplied with Decision Accelerator contain examples of data methods that perform this type of task for consumer credit report and FICO® Expansion score requests. If you want Decision Accelerator to request other types of data from Intelligent Data Manager, you can add data methods to construct request objects using hard-coded values.
4. Configured values in the Intelligent Data Manager Dataflow Configuration Workstation.

See Also For information on using the Intelligent Data Manager Dataflow Configuration Workstation, see the <REMOVED>.

More detailed information on Decision Accelerator requests to Intelligent Data Manager is contained in the individual data source chapters.

See Also

- For more information on consumer credit report requests and consumer ARF input, see Chapter 4, “Consumer Credit Report Data.”
- For more information on list of similars requests, business credit report requests, and business ARF input, see Chapter 5, “Business Credit Report and LOS Data.”
- For more information on FICO® Expansion score service requests, see Chapter 6, “FICO® Expansion Score Report Data.”
- For more information on SBFE data requests, see Chapter 7, “SBFE Data.”

Conditional Branching

Decision Accelerator can perform conditional branching based on data from Intelligent Data Manager. For example, some of the pre-defined product strategies delivered with Decision Accelerator check to see whether credit report data exists before requesting it. After credit report data has been requested, a decision flow determines whether full consumer credit report analysis data is available, or whether there is a thin file or no report. If there is a full report, it may be analyzed to see whether it indicates fraud. If the file is thin or non-existent, Decision Accelerator may request FICO® Expansion score data from Intelligent Data Manager.

Consortium Data Extract Developer's Guide

The Consortium Data Extraction (CDE) process is comprised of a collection of scripts and executable files that simplify the task of extracting relevant data from the daily transaction data that is captured by Falcon. The CDE process extracts, encrypts, packages, and compresses both the transaction data and the associated case data. Live transaction data, combined with the case data produced by case analysts, contains an enormous amount of valuable information, which helps Fair Isaac determine how well Falcon is performing in production.

Data Security

Because the transaction data that is being collected and sent to Fair Isaac contains sensitive customer information, precautions have been taken to ensure that the entire end-to-end process of collecting, managing, and transferring this data is secure.

The actual values contained in the more sensitive fields such as customer account numbers are not needed. In order to associate multiple transactions from a customer and to connect fraud cases created by analysts, the CDE process protects sensitive information by encrypting all incoming account numbers using an algorithm. This algorithm generates a unique number that is the same length as the original account number but conceals its content while providing Fair Isaac the information needed to improve future model performance.

Encryption

Sensitive data is automatically encrypted on the Falcon Server by the CDE process as it enters the system. Sensitive data is never stored to disk in its unencrypted form. The encryption is done using a private key that is generated by the CDE process.

Once encrypted, the data stays encrypted throughout the analysis process and is destroyed immediately afterward. Fair Isaac can extrapolates the majority of information required for analysis by using the field sizes and their encrypted contents.

Encrypted Fields

As Falcon captures live transactions, the CDE process encrypts the following data fields prior to saving any of the data to temporary storage.

Fully encrypted data fields

- AGENCY ACCT WITH MEMB
- BANK CUST NUM
- BENEF CUST ACCT ADDR
- BENEF CUST ACCT NAME
- BENEF CUST ACCT NUM
- CUST ACCT NUM
- CUST ID NUM
- FIRST NAME
- LAST NAME

- REF INFO
- REGULATORY REPORTING
- REMITTANCE INFO
- ORIG CUST ACCT ADDR
- ORIG CUST ACCT NAME
- STREET LINE 1
- STREET LINE 2

Partially encrypted data fields

- DOB — Encrypt columns 5-8 (MMDD), leave columns 1-4 (CCYY) in clear text.
- TELEPHONE — Encrypt columns 6-16, leaves columns 1-5 in clear text.

Sending Extracted Data to Fair Isaac

Your Professional Services representative will coordinate a schedule and secure method of transmitting your compressed archives to Fair Isaac that conforms with your organizations security polices.

The Consortium Data Extract Process

Both live transaction data and case creation data contribute to analyzing the performance of models. The following sections describe how the CDE process captures information from live transaction data and case data.

Capturing Live Transaction Data

The CDE process uses a bulk loading file to write live transaction data into the database in large batches rather than as individual transactions. The information is stored in a bulk-load file until a pre-determined limit (size or schedule) is reached. Once this limit is reached, the CDE process writes the data to the database. The bulk-loader is much more efficient than writing individual transactions and has a lessor impact on system performance.

Note If CDE process is disabled, the bulkload file is deleted after the data is loaded into the database.

Whichever the trigger, size or schedule, once the write process is underway, profile data that we use for analysis purposes is appended to the bulkload data and then the entire contents is written to the database in one large batch, thereby lightening the burden on your system that would be caused by saving individual transactions to the database separately.

Live transaction data alone, however, is only part of the picture. To analyze how IBS is performing, we must examine your live transactions in light of the fraud cases that your analysts investigate. It's necessary to analyze both data sets together in order to see the complete picture of how well IBS is performing.

Case Data

The CDE process also collects the case data that is produced by your analysts who are investigating potential cases of fraud. In order to gather this information, the CDE process runs a nightly query against the database and creates an archive file for the previous day.

Once the nightly query is complete, the CDE process combines the archived live data with the results of the query of the case data. It combines this information with several other files that define the structure and the data and writes all of it to an archive file. Finally, the CDE process combines the current daily archive with subsequent archives. The full archive can then be compressed and prepared for transfer.

API Reference Guides

Here are a few excepts from several of the API reference guides I have written over the past few years.

- xPatterns C.A.C. API Reference Guide
- xPatterns Platform API Reference Guide
- xPatterns E/M Audit API Reference

xPatterns C.A.C. API Reference Guide

Welcome to the xPatterns C.A.C. API Reference. This guide contains technical reference information for developers who are using xPatterns C.A.C. platform, which provides a restful interface for programmatically accessing xPatterns C.A.C. technology.

Before you start accessing xPatterns C.A.C., we recommend that you read the API Overview, which provides detailed information about how to use the xPatterns C.A.C. API.

The xPatterns C.A.C. interfaces contain a large number of methods that are organized into the following sections:

- EMR Analysis — Processes unstructured provider notes and returns assertions, which indicate what is known about the medical problems contained in the note text.
- EMR Coding Inference — Identifies medical codes that may apply to the list of assertions from Analysis, which are provided as input.
- Encoder— Provides functionality to help coders more efficiently encode provider notes.
- Affiliation— Provides the functionality necessary to associate users with a particular organization.
- Notes— Provides the functionality necessary to create, assign, and modify patient notes.
- Encounters— Provides the functionality necessary to work with patient records at the encounter level, which groups all of the medical notes into one billable unit.
- Product— Provides information to support xPatterns C.A.C. products and workflow.
- Reporting— Exposes usage data that can be used to generate productivity reports.
- User— Provides the functionality necessary to create and manage user accounts and settings.

Getting Started

The IBS API provides a RESTful interface for accessing the xPatterns C.A.C. platform. You can use the xPatterns C.A.C. API to develop applications using simple HTTP methods to efficiently and securely encode provider notes and to store, share, and manage them from anywhere on the Internet.

These sections from the developer's guide provide detailed information about how to get started using the xPatterns C.A.C. API.

- Setting Up a Java Development Environment
  Describes how to set up your development environment to begin working with the REST APIs immediately.
- Getting Acquainted with REST
  The REST API is the underlying interface for all APIs in the xPatterns C.A.C. platform. This topic discusses some of the details of the xPatterns REST implementation.
- Getting Acquainted with JSON
  xPatterns uses JavaScript Object Notation for data transfer. This section provides a more detail description of how xPatterns uses JSON to represent data.

EMR Analysis

The EMR Analysis service identifies medical conditions contained in unstructured EMR text and generates an assertion classifier for each one. Assertion classifiers are labels, which indicate how and to what degree of certainty medical conditions pertain to a patient.

The assertions generated by this service are often used as input to the EMR Coding Inference service.

The EMR Analysis Service is comprised of a single method that processes unstructured provider notes and returns assertions, which indicate what is known about the medical problems contained in the note text.

- analyzeEmr

analyzeEmr

Analyzes unstructured EMR text and generates a list of assertions, which connect the medical issues from the EMR to a specific patient.

Analysis Service WADL

https://emr-analysis-beta.atigeo.com/emr-analysis/services?_wadl

Base URL

https://emr-analysis-beta.atigeo.com/

Resource URI

emr-analysis/services/note/assertion

Request Body	Description
AnalysisInput input	Provides properties that enable users to submit emrText and addtionalFields, which can contain known facts about the patient.

Returns

Returns a set of assertions which label medical problems with a degree of certainty as they relate to the current patient.

HTTP POST Request

This request is truncated to save space.

```
POST https://cac.atigeo.com/emr-analysis/services/note/assertion HTTP/1.1
Accept: application/json
apikey: <GUID>
Content-Type: application/xml
User-Agent: Java/1.6.0_33
Host: https://cac.atigeo.com
Connection: keep-alive
Content-Length: 4616

<?xml version="1.0" ?>
<analysisInput>
    <additionalFields>
        <entry>
            <key>
        ALLERGY
            </key>
            <value>
        Pollen
            </value>
        </entry>
        <entry>
            <key>
```

```
        AGE
            </key>
            <value>
        15
            </value>
        </entry>
        <entry>
            <key>
        GENDER
            </key>
            <value>
        Male
            </value>
        </entry>
    </additionalFields>
    <source>
    Cardiac Consultation - Description: The patient has a
previous history of aortic valve disease, status post aortic
valve replacement, a previous history of paroxysmal atrial
fibrillation, congestive heart failure, a previous history of
transient ischemic attack with no residual neurologic deficits.
HISTORY OF PRESENT ILLNESS: The patient is a 41-year-old African-
American male previously well known to me. He has a previous
history of aortic valve disease, status post aortic valve
replacement on 10/15/2007, for which he has been on chronic
anticoagulation. There is a previous history of paroxysmal atrial
fibrillation and congestive heart failure, both of which have
been stable prior to this admission. Keywords: cardiovasculr /
pulmonary, aortic valve disease, aortic valve replacement,
anticoagulation, paroxysmal atrial fibrillation, congestive heart
failure, neurologic deficits, atrial fibrillation, aortic valve,
heart, ischemic, atrial, neurologic, aortic, valve, disease,
    </source>

</analysisInput>
```

JSON Response


```
[
   {
      "id":1,
      "originalSection":null,
      "normalizedSection":null,
      "subject":null,
      "conceptType":"ALLERGY",
      "conceptValue":"Pollen",
      "normalizedConceptValue":"Pollen",
```

```
        "dictionary":null,
        "idInDictionary":null,
        "polarity":"POSITIVE",
        "justificationAssertionIds":[

        ],
        "confidence":1.0,
        "sequenceNumber":null,
        "textPositions":[

        ]
    },
    {
        "id":2,
        "originalSection":null,
        "normalizedSection":null,
        "subject":null,
        "conceptType":"AGE",
        "conceptValue":"15",
        "normalizedConceptValue":"15",
        "dictionary":null,
        "idInDictionary":null,
        "polarity":"POSITIVE",
        "justificationAssertionIds":[

        ],
        "confidence":1.0,
        "sequenceNumber":null,
        "textPositions":[

        ]
    },
    {
        "id":3,
        "originalSection":null,
        "normalizedSection":null,
        "subject":null,
        "conceptType":"GENDER",
        "conceptValue":"Male",
        "normalizedConceptValue":"Male",
        "dictionary":null,
        "idInDictionary":null,
        "polarity":"POSITIVE",
        "justificationAssertionIds":[

        ],
        "confidence":1.0,
```

```
      "sequenceNumber":null,
      "textPositions":[

      ]
   },
   }
      "confidence":0.0,
      "sequenceNumber":null,
      "textPositions":[
         {
            "start":1889,
            "end":1890
         }
      ]
   }
```


Error Codes and Messages

The following table lists error codes and HTTP response codes returned by the system.

Code	Name
0	**UnhandledException** This exception likely caught us off guard.
1	**ParameterTypeMismatch** Parameters in the request do not match the data type that's expected. For example, this error happens if requests contain integers when Strings are expected.
2	**InvalidBodyRequestParameter** Parameters are likely missing, spelled incorrectly, or appear out of order in the request.
4	**InvalidCredentials** We are unable to find a match in our system for the username and password combination you provided.
6	**NotAuthorizedException** Be sure you're using the correct API key for the method you're calling. Admin features require a different access level than coder features.
7	**NullOrEmptyUrl** The url in the request seems to be missing.
8	**InvalidTag** Please check the entity names (e.g., encounters, notes) included in your request.
400	**Bad Request** The request could not be understood by the server due to malformed syntax (such as a bad query string).
401	**Unauthorized** A user token has expired, is invalid, or attempted to access an endpoint to which it does not have access.

Code	Name
404	**NotFound** The resource requested was not found or the URL used to make the call was invalid.
500	**Internal Server Error** This means something went wrong on our end. Please try again or contact your Atigeo Customer Service Representative.
503	**Service Unavailable** We’re experiencing temporary down-time.

EMR Coding Inference

This service automatically assigns medical codes to an EMR. These codes are inferred from a given set of assertions, which are derived from the EMR text. during.

The EMR Inference Service is comprised of a single method that generates medical codes from the assertions it receives.

- generateCodes

generateCodes

Retrieves codes inferred from a given set of assertions.

Base URL

https://cac.atigeo.com/

Resource URI

coding-inference-ws/services/coding-service/generate-codes

Path Variable	Description
AnalysisInput input	Contains properties for setting and getting emrText and additionalFields, which can hold assertions.

Returns

Returns a JSON object containing EMR codes and the assertions sent in the request.

HTTP POST Request

```
POST https://cac.atigeo.com/coding-inference-ws/services/coding-service/generate-
codes HTTP/1.1
Accept: application/json
apikey: <GUID>
Content-Type: application/json;charset=UTF-8
User-Agent: Java/1.6.0_33
Host: https://cac.atigeo.com
Connection: keep-alivehttps://cac.atigeo.com
Content-Length: 706

[
    {
       "id":1,
       "originalSection":"worst problem",
       "normalizedSection":"CHIEF_COMPLAINT",
       "subject":"PATIENT",
       "conceptType":"SYMPTOM",
       "conceptValue":"hyperlipidemia",
       "normalizedConceptValue":"hyperlipidemia",
       "dictionary":null,
       "idInDictionary":null,
       "polarity":"POSITIVE",
       "justificationAssertionIds":[

       ],
       "confidence":null,
       "sequenceNumber":null,
       "textPositions":[

       ]
    },
```

```
    {
        "id":2,
        "originalSection":"histori",
        "normalizedSection":"HISTORY_PRESENT_CONDITION",
        "subject":"PATIENT",
        "conceptType":"SYMPTOM",
        "conceptValue":"history of migraine",
        "normalizedConceptValue":"migraine",
        "dictionary":null,
        "idInDictionary":null,
        "polarity":"POSITIVE",
        "justificationAssertionIds":[

        ],
        "confidence":null,
        "sequenceNumber":null,
        "textPositions":[

        ]
    }
```


]JSON Response


```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Date: Wed, 15 Aug 2012 22:39:12 GMT
Content-Type: application/json
Content-Length: 1330

[
    {
        "id":1,
        "originalSection":"worst problem",
        "normalizedSection":"CHIEF_COMPLAINT",
        "subject":"PATIENT",
        "conceptType":"SYMPTOM",
        "conceptValue":"hyperlipidemia",
        "normalizedConceptValue":"hyperlipidemia",
        "dictionary":null,
        "idInDictionary":null,
        "polarity":"POSITIVE",
        "justificationAssertionIds":[

        ],
        "confidence":null,
        "sequenceNumber":null,
        "textPositions":[

        ]
```

```
},
{
   "id":2,
   "originalSection":"histori",
   "normalizedSection":"HISTORY_PRESENT_CONDITION",
   "subject":"PATIENT",
   "conceptType":"SYMPTOM",
   "conceptValue":"history of migraine",
   "normalizedConceptValue":"migraine",
   "dictionary":null,
   "idInDictionary":null,
   "polarity":"POSITIVE",
   "justificationAssertionIds":[

   ],
   "confidence":null,
   "sequenceNumber":null,
   "textPositions":[

   ]
},
{
   "id":3,
   "originalSection":null,
   "normalizedSection":null,
   "subject":null,
   "conceptType":"CODE",
   "conceptValue":"272.4",
   "normalizedConceptValue":"272.4",
   "dictionary":"ICD9_VOL1_VOL2",
   "idInDictionary":null,
   "polarity":"POSITIVE",
   "justificationAssertionIds":[
      1
   ],
   "confidence":null,
   "sequenceNumber":0,
   "textPositions":[

   ]
},
{
   "id":4,
   "originalSection":null,
   "normalizedSection":null,
   "subject":null,
   "conceptType":"CODE",
```

```
        "conceptValue":"346.90",
        "normalizedConceptValue":"346.90",
        "dictionary":"ICD9_VOL1_VOL2",
        "idInDictionary":null,
        "polarity":"POSITIVE",
        "justificationAssertionIds":[
           2
        ],
        "confidence":null,
        "sequenceNumber":1,
        "textPositions":[

        ]
    }
]
```


xPatterns Platform API Reference Guide

This section of the xPatterns Platform API Reference Guide provides an in depth look at the public methods exposed by the xPatterns API. There are two versions of the platform currently in production.

xPatterns 3.6.x
xPatterns 3.6.x supports native search indexes, static domain experts, and dynamic domain experts.

xPatterns 3.2.x
xPatterns 3.2.x supports CRNs and Standard DEs, both of which are adaptable (support training).

Note Use submitContentFeedback() with CRNs. Use submitDisambiguationFeedback() with standard domain experts.

xPatterns 3.6.x

xPatterns 3.6.x supports native search indexes, static domain experts, and dynamic domain experts.

- searchContent
- getUrlTerms
- disambiguateByCategories
- getContentList
- getExpertList
- getContentInfo
- getExpertInfo
- searchContent
- getContentMetadata
- getContentTerms
- searchRelatedLinkedDocuments
- autocompleteContent
- spellcheckContent
- disambiguateTerms(GET)
- disambiguateTerms(POST)
- disambiguateByCategories(GET)
- disambiguateByCategories(POST)
- getRelevantConcepts(GET)
- getRelevantConcepts(POST)
- getSimilarContent
- getTextTerms
- getUrlTerms
- addContentDocument
- deleteContentDocument
- updateContentDocument
- generateAPIKey

searchContent

Returns relevant content that can be filtered, sorted, and presented in a multitude of ways based on the values of input parameters.

Note This feature must use a native search index.

Base URL

https://services36.xpatterns.com/xrelevance-api-rest/relevance

Resource URI

/searchContent/content/{contentId}

Boolean Queries

This Boolean logic works for both "edismax" and "standard" queryTypes.

Query	xRelevance Query
`Term1 AND Term2 AND Term3`	query=+Term1 +Term2 +Term3
`Term1 OR Term2 OR Term3`	query=Term1 Term2 Term3
`"Multi word ngram 1" AND "Multi word ngram 2"`	query=+"Multi word ngram 1" +"Multi word ngram 2"
`"Multi word ngram 1" OR "Multi word ngram 2"`	query="Multi word ngram 1" "Multi word ngram 2"
`NOT Term1`	query=-Term1

Facet Retrieval

Use facets to include all documents in the results set. To retrieve all possible facets, use query="*:*", in both "edismax" and "standard" queryType mode.

Boost Queries

Use boost queries for queryType=edismax

Parameter	Description
`query=tamoxifen&bq=publicationDate\[NOW/DAY-1YEAR TO NOW/DAY\]`	Boosts the score of all articles that are at most a year older. This translates into more recent articles coming up higher in the list. This applies even when the relevance score is lower than for older articles
`query=tamoxifen&bf=recip(rord(publicationDate),1,1000,1000)^2.5`	Boosts releance scores.
`query=tamoxifen&bq=author:John Doe^5.0`	Boosts scores of all articles written by author "John Doe"

Path Variable	Description
String contentId	Identifies the target index for this query.

Request Parameter	Required	Default	Description
String query	Yes		Identifies a specific query. Required.
String [] filters	No		Custom fields used in context.
String [] facets	No		Category into which to group relevant content (e.g., Manufacturer). Supports faceted browsing and enables users to refine search results.
String [] facetFields	No		Constraints or facet values that pertain to the specified facet (e.g., Ford, Toyota, BMW).
Integer offset	No	0	Specifies a zero-based offset into the relevant content at which to begin retrieving (e.g., for pagination).
Integer limit	No	10	An optional maximum number of relevant content items to return as a result of the query starting at offset.
String [] sort	No	score descending.	Specifies the order in which to sort results. Provide the sort field followed by a space and the sort method.
String [] fields	No		String array that contains one or more partner-defined fields.
String facetDateStart			Start date of faceted data.
String facetDateEnd			End date of faceted data.
Integer facetDateGap			The size of each date range expressed as an interval to be added to the lower bound of the facet date.
String [] categoryFilter	No		The unique Id of plugin and category ex:1:3 (plugin1 and the 3rd category)
Boolean spellCheckerEnabled			Toggles spell checker.on or off. Default = False.
Boolean showCategories	No	False	
queryType	No	edismax	Possible values: "edismax "standard

Request Parameter	Required	Default	Description
qf	No		Query Fields: Specify the fields to be queried by default with their boost values. E.g. fieldOne^2.3 fieldTwo fieldThree^0.4 Can only be used with queryType=edismax
bq	No		Boost Query: Specify additional score boosting for documents matching this query - see Boost Queries section Can only be used with queryType=edismax
bf	No		Boost Function: Specify additional score boosting using a function - see Boost Queries section. Can only be used with queryType=edismax
boost	No		Same as "bf" but the boost value is multiplicative. Can only be used with queryType=edismax
pf	No		Can only be used with queryType=edismax See http://wiki.apache.org/solr/ExtendedDisMax
ps	No		Can only be used with queryType=edismax See http://wiki.apache.org/solr/ExtendedDisMax
pf2	No		Can only be used with queryType=edismax See http://wiki.apache.org/solr/ExtendedDisMax
ps2	No		Can only be used with queryType=edismax See http://wiki.apache.org/solr/ExtendedDisMax
pf3	No		Can only be used with queryType=edismax See http://wiki.apache.org/solr/ExtendedDisMax

Request Parameter	Required	Default	Description
ps3	No		Can only be used with queryType=edismax See http://wiki.apache.org/solr/ExtendedDisMax
qs	No		Can only be used with queryType=edismax See http://wiki.apache.org/solr/ExtendedDisMax
mm	No	100%	Can only be used with queryType=edismax See http://wiki.apache.org/solr/ExtendedDisMax
tie	No	0.1	Can only be used with queryType=edismax See http://wiki.apache.org/solr/ExtendedDisMax

Returns

Returns relevant content.

HTTP GET Request

```
GET https://services36.xpatterns.com/xrelevance-api-rest/relevance/searchContent/
content/fullWiki?query=mock
query&offset=0&limit=10&facet=age:23&facet=age:30&facet=author:John
Doe&facetField=age&facetField=author&field=title&categoryFilter=1:3&filter=age:[1 TO
*]&showCategories=true&spellCheckerEnabled=true&sort=score%20desc&queryType=edismax&
qf=title^20.0+description+body

Content-Type:application/json; charset=utf-8
X-Request:JSON
dataType:json
apikey:5ba70490-8744-4912-a8bf-5ad9bf79969d
```

JSON Response

```
Content-Type :application/json;charset=UTF-8
Date :Thu, 13 Dec 2012 13:37:50 GMT
Server :Apache-Coyote/1.1
Transfer-Encoding :chunked

{
    "query": "mock query",
    "content": [
        {
            "categories": {
                "1:11": 0.70,
                "1:12": 0.15,
                "2:21": 0.10,
        "None": 0.05
```

```
                },
                "fields": {
                    "title": "1st title",
            "author": [ "John Doe", "Jane Doe" ]
                },
                "documentId": "id1",
                "score": 4.05
            },
            {
                "categories": {
            "None": 0.60,
                    "2:21": 0.20,
                    "2:22": 0.10,
                    "1:14": 0.05,
            "1:92": 0.05
                },
                "fields": {
                    "title": "second title",
            "author": [ "John Doe", "Jane Doe 2" ]
                },
                "documentId": "id2",
                "score": 3.45
            },
            {
                "categories": {
            "None":1.00
},
                "fields": {
                    "title": "3rd title"
            "author": [ "John Doe 2", "Jane Doe 2"]
                },
                "documentId": "id3",
                "score": 3.25
            }
        ],
        "queryFacets": {
            "age:23": 5,
            "age:30": 4,
            "author:John Doe":3
        },
        "fieldFacets": {
            "age": {
                "age:14": 2,
                "age:23": 5,
                "age:30": 4
            },
            "author": {
```

```
        "author:John Doe":23,
        "author:Jane Doe":34,
        "author:John Doe 2":2,
        "author:Jane Doe 2":7
            }
        },
        "spellCheckResponse": {
            "correctlySpelled": false,
            "spellcheckSuggestions": [
                {
                    "collationQueryString": "mocking query",
                    "numberOfHits": 11
                },
                {
                    "collationQueryString": "mocked query",
                    "numberOfHits": 21
                },
                {
                    "collationQueryString": "mock queries",
                    "numberOfHits": 6
                }
            ]
        },
        "queryId": null,
        "totalFound": 3
}
```


Error Codes and Messages

Code	Name	Message
0	UnhandledException	Provides notification of uncaught exceptions.
29	ParameterTypeMismatch	Parameter is not the correct type.
54	InvalidContentId	Content id does not exist.
55	MissingContent	Missing content id parameter.
59	InvalidRight	Null or empty right name.
60	NotAuthorizedException	HTTPS URLs not supported.
81	NullOrEmptyUrl	Null or empty url string detected.
83	NoContentRetrievedFromUrl	No textual content extracted from URL
84	ErrorRetrievingOrProcessingContentFromUrl	Error retrieving or processing content from url.

Code	Name	Message
91	ContentNotOnline	Content corpus not online yet.
404	NotFound	Not Found.

getUrlTerms

Returns relevant terms for web content located at the specified url.

Base URL

https://services36.xpatterns.com/xrelevance-api-rest/relevance

Resource URI

urlTerms/content/{contentId}

Path Variable	Description
String contentId	Identifies the native content for which the desired information is retrieved. Required

Request Parameter	Description
Boolean showOnlyInCorpus	Optional. Default = true
Integer offset	Specifies an offset into the list of results where retrieval will begin or resume.Default = 0.
Integer limit	Specifies the maximum number of results to return. Default = 10.Default = 10.
String url	Url specifies a web page containing text that will be submitted as a query string.

Returns

Returns an array of scored terms.

HTTP GET Request

```
GET https://services36.xpatterns.com/xrelevance-api-rest/xrelevance-api-rest/
relevance/urlTerms/content/fullWiki?offset=0&limit=10&url=http://en.wikipedia.org/
Machine&showOnlyInCorpus=true
```

JSON Response

```
Access-Control-Allow-Credentials :true
Access-Control-Allow-Origin :chrome-extension://fdmmgilgnpjigdojojpjoooidkmcomcm
Cache-Control :private
Content-Type :application/json;charset=UTF-8
Date :Thu, 13 Dec 2012 13:07:53 GMT
Expires :Thu, 13 Dec 2012 14:07:53 GMT
Last-Modified :Thu, 13 Dec 2012 13:07:53 GMT
Server :Apache-Coyote/1.1
Transfer-Encoding :chunked

[
     {
          "term": "a",
          "score": 3.4,
},
     {
          "term": "b",
```

```
            "score": 2.1,
},
    {
            "term": "c",
            "score": 1.1,
},
    {
            "term": "d",
            "score": 1.1,
    }
]
```

Error Codes and Messages

Code	Name	Message
0	UnhandledException	Provides notification of uncaught exceptions.
8	LowLimit	Limit must be greater than 0.
9	LowOffset	Offset must be greater than or equal to 0.
10	MissingLimitParameter	Term limit required.
11	MissingOffsetParameter	Term offset required.
19	OffsetExceededresults	Term offset exceeded the number of results.
28	NotExistingDocumentId	Document not found.
29	ParameterTypeMismatch	Parameter is not the correct type.
39	RightsViolation	You are not authorized
54	InvalidContentId	Null or empty content id.
55	MissingContent	Missing content id parameter.
59	InvalidRight	Null or empty right name.
60	NotAuthorizedException	Not authorized on a resource.
81	NullOrEmptyUrl	Null or empty url.
89	MethodDoesNotExist	Invalid method name
91	ContentNotOnline	Content corpus not online yet.
404	NotFound	No Found.

disambiguateByCategories

Returns related domain expert terms for a list of input terms and filters for specific categories.

Base URL

https://services3.xpatterns.com/xrelevance-api-rest/relevance

Resource URI

/disambiguateByCategories/expert/{expertId}

Path Variable	Description
String expertId	Identifies the domain expert that will be used to disambiguate terms.

Request Parameter	Description
String[] categoryFilter	Identifies a list of categories used to filter the result set.

Request Body	Description
ExpertDisambiguationModel query	Contains terms to disambiguate as well as term limit and term offset parameters.

Returns

Returns a list of relevant terms and categories.

HTTP POST Request

```
GET https://services3.xpatterns.com/xrelevance-api-rest/relevance/
disambiguateByCategories/expert/
testExpert?categoryfFilter=1:12&categoryfFilter=2:23&offset=0&limit=10

"apiKey:b9cfc72c2999-71a1-41e5-a5ee-b9cfc72c2999-a5e5ee"
"Content-Type: application/json"
"Accept: application/json"

'["blood","skin"]'
```

JSON Response

```
Access-Control-Allow-Credentials :true
Access-Control-Allow-Origin :chrome-extension://fdmmgilgnpjigdojojpjoooidkmcomcm
Cache-Control :private
Content-Type :application/json;charset=UTF-8
Date :Thu, 13 Dec 2012 13:08:45 GMT
Expires :Thu, 13 Dec 2012 14:08:45 GMT
Last-Modified :Thu, 13 Dec 2012 13:08:45 GMT
Server :Apache-Coyote/1.1
Transfer-Encoding :chunked

{
    "terms": [
        {
            "term": "contraceptive agents female",
```

```
                "score": 0.2,
                "annotatorCategoryPairs": [
                    {
                        "category": "1:11",
                        "externalId": "externalId1"
                    },
                    {
                        "category": "1:13",
                        "externalId": "externalId2"
                    }
                ]
            },
            {
                "term": "vascular diseases",
                "score": 0.5,
                "annotatorCategoryPairs": [
                    {
                        "category": "2:23",
                        "externalId": "externalId3"
                    },
                    {
                        "category": "2:21",
                        "externalId": "externalId4"
                    }
                ]
            }
        ],
        "queryId": null
}
```


Error Codes and Messages

Code	Name	Message
0	UnhandledException	Provides notification of uncaught exceptions.
2	IllegalCharacters	One of the arguments contains illegal characters.
4	EmptyArgument	Argument cannot be empty
5	NullOrEmptyDisambiguateTerms	No terms were received for disambiguation.
8	LowLimit	Limit must be greater than 0.
9	LowOffset	Offset must be greater than or equal to 0.
10	MissingLimitParameter	Term limit required.
11	MissingOffsetParameter	Term offset required.

Code	Name	Message
	OffsetExceededresults	Term offset exceeded the number of results.
29	ParameterTypeMismatch	Parameter is not the correct type.
39	RightsViolation	You are not authorized
52	InvalidExpertId	Null or empty expert id.
60	NotAuthorizedException	Not authorized on a resource.
81	NullOrEmptyUrl	Null or empty url.
89	MethodDoesNotExist	Invalid method name
91	CopntentNotOnline	Content corpus not online yet.
110	MaxLimitExceeded	Limit must be lower than: x
404	NotFound	No Found.

xPatterns E/M Audit API Reference

This section of the xPatterns E/M Audit Developer's Guide provides an in depth look at the public methods exposed through the E/M Audit API.

The E/M Audit API provides the functionality necessary to audit coding for specific EMRs. This can be done as part of a self-audit process or prior to submitting encounters for billing.

The auditCodedEmr method allows the calling system to submit EMR text, a CPT code, patient information, and a service type. The system analyzes this information and returns a confidence score, which indicates how appropriate the assigned code is for that EMR.

Before analysis begins, the service verifies the following information:

- The code is a valid CPT code;
- The patient type and service type are correct for the given code; and
- The code is one that the system can audit.

Using a machine-learning model, the service parses the EMR text and predicts a score that captures how appropriate the code is for the EMR. It should be understood that any such algorithm will not be correct 100% of the time.

The EMR E/M Audit Service is comprised of a single method that validates historical audits.

- auditCodedEmr

auditCodedEmr

Validates input parameters, analyzes EMR text against its assigned CPT code, and returns a confidence score indicating how well the code matches information contained in the EMR.

Base URL

https://services.xpatterns.com/

Resource URI

coding-inference-ws/services/coding-service/audit-em-codes

Request Parameter	Required	Description
String code	Yes	Specifies the CPT code that the audit system will validate against the EMR text.
String patientType	Yes	Specifies the patient type as ESTABLISHED or NEW.
String serviceType1	Yes	Specifies a service type. There are 58 possible service types, which are defined in the sections/subsections/ subsubsections of the 2012 CPT code book.
String serviceType2	No	Specifies a service type. There are 58 possible service types, which are defined in the sections/subsections/ subsubsections of the 2012 CPT code book.
String serviceType3	No	Specifies a service type. There are 58 possible service types, which are defined in the sections/subsections/ subsubsections of the 2012 CPT code book.

Request Body	Description
String emrText	EMR text to be coded for this audit.

Returns

Returns audit metadata and confidence scores that indicate the accuracy of the assigned CPT code for the given EMR Text.

HTTP POST Request

```
POST https://services.xpatterns.com/coding-inference-ws/services/coding-service/
audit-em-
codes?code=99214&patientType=ESTABLISHED&serviceType1=OUTPATIENT_SERVICES&serviceTyp
e2=""&serviceType3=""
Accept: application/json
apikey: 7a52c9f8-43e3-5d9e-7f32-185d4b93a0f5
Content-Type: application/json;charset=UTF-8
User-Agent: Java/1.6.0_33
Host: services.xpatterns.com
Connection: keep-alive

[Sample Name: Atrial Fibrillation Management
Description: The patient is a very pleasant 62-year-old African
```

American female with a history of hypertension, hypercholesterolemia, and CVA, referred for evaluation and management of atrial fibrillation.
(Medical Transcription Sample Report)
REASON FOR CONSULTATION: Atrial fibrillation management.

PAST MEDICAL HISTORY:
1. Hypertension.
2. Myocardial infarction in 2003.
3. Left heart catheterization at University Hospital.
4. Hypercholesterolemia.
5. Arthritis.
6. CVA in 2002 and in 2003 with right eye blindness.

PAST SURGICAL HISTORY:
1. Left total knee replacement in 2002.
2. Left lower quadrant abscess drainage in 12/07

FAMILY MEDICAL HISTORY: Significant for lung and brain cancer. There is no history that she is aware of cardiovascular disease in her family nor has any family member had sudden cardiac death.

SOCIAL HISTORY: She is retired as a cook in a school cafeteria, where she worked for 34 years. She retired 7 years ago because of low back pain. She used to smoke 2-1/2 packs per day for 32 years, but quit in 1995. Denies alcohol, and denies IV or illicit drug use.

ALLERGIES: No known drug allergies.

MEDICATIONS:
1. Coumadin 5 mg a day.
2. Toprol-XL 50 mg a day.
3. Aspirin 81 mg a day.
4. Hydrochlorothiazide 25 mg a day.
5. Plendil 10 mg daily.
6. Lipitor 40 mg daily.

REVIEW OF SYSTEMS: As above stating that following her stroke, she has right eye blindness, but she does have some minimal vision in her periphery.

PHYSICAL EXAMINATION:
VITAL SIGNS: Blood pressure 138/66, pulse 96, and weight 229 pounds or 104 kg. GENERAL: A well-developed, well-nourished, middle-aged African American female in no acute distress. NECK: Supple. No JVD. No carotid bruits. CARDIOVASCULAR: Irregularly

irregular rate and rhythm. Normal S1 and S2. No murmurs, gallops or rubs. LUNGS: Clear to auscultation bilaterally. ABDOMEN: Bowel sounds positive, soft, nontender, and nondistended. No masses. EXTREMITIES: No clubbing, cyanosis or edema. Pulses 2+ bilaterally.

LABORATORY DATA: EKG today revealed atrial fibrillation with nonspecific lateral T-wave abnormalities and a rate of 94.

IMPRESSION: The patient is a very pleasant 62-year-old African American female with atrial fibrillation of unknown duration with symptoms of paroxysmal episodes of palpitations, doing well today.

RECOMMENDATIONS:
1. Her rate is suboptimally controlled, we will increase her Toprol-XL to 75 mg per day.
2. We will obtain a transthoracic echocardiogram to evaluate her LV function as well as her valvular function.
3. We will check a thyroid function panel.
4. We will continue Coumadin as directed and to follow up with Dr. X for INR management.
5. Given the patient's history of a stroke in her age and recurrent atrial fibrillation, the patient should be continued on Coumadin indefinitely.
6. Depending upon the results of her transthoracic echocardiogram, the patient may benefit from repeat heart catheterization. We will await results of transthoracic echocardiogram.
7. We will arrange for the patient to wear a Holter monitor to monitor the rate controlled on a 24-hour period. She will then return to the electrophysiology clinic in 1 month for followup visit with Dr. Y.

The patient was seen, discussed, and examined with Dr. Y in electrophysiology.

]

JSON Response

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/json
Transfer-Encoding: chunked
Date: Wed, 15 May 2013 22:58:07 GMT
```


```
{
    "code": "99214",
    "status": "PREDICTION",
```

```
    "details": "",
    "requiredPatientExam": "DETAILED",
    "requiredPatientHistory": "DETAILED",
    "requiredMedicalDecisionMaking": "MODERATE",
    "requiredNumComponents": 2,
    "patientHistoryScore": 0.5854258406146795,
    "patientExamScore": 0.5854258406146795,
    "medicalDecisionMakingScore": 0.06352944649838221,
    "confidence": 0.37356091640349887
}
```

Return Values

The following table describes each field in the audit results.

Name	Type	Description
Status	Enumerated type, with the following values: ERROR("Server error"), NOT_EM_CODE("Code is not an E/M code."), UNSUPPORTED_EM_CODE("Auditing for this code is not supported."), BAD_PATIENT_TYPE("Patient type is not valid for this code."), BAD_SERVICE_TYPE1("Service type 1 is not valid for this code."), BAD_SERVICE_TYPE2("Service type 2 is not valid for this code."), BAD_SERVICE_TYPE3("Service type 3 is not valid for this code."), PREDICTION("made a prediction");	"PREDICTION". indicates a successful audit.
Code	String	Specifies the CPT code that was audited.
requiredPatientExam	String: one of "Problem focused", "Extended problem focused", "detailed", "comprehensive	Indicates the required level of patient history to support this code.
requiredPatientHistory	String: one of "Straightforward", "Low", "Medium", "High"	Indicates the required level of medical decision making to support this code.
requiredNumComponents	int	Indicates the number of key components necessary to be supported for this code.
patientHistoryScore	double	Measure of confidence that the specified level of patient history is supported in the EMR.
physicalExamScore	double	Measure of confidence that the specified level of physical examination is supported in the EMR.
medicalDecisionMakingScore	double	Measure of confidence that the specified level of medical decision making is supported in the EMR
Confidence	double	Measure of confidence that the specified EMR code is supported by the EMR.

Error Codes and Messages

The following table lists error codes and HTTP response codes returned by the audit system.

Code	Name
0	**UnhandledException** This exception likely caught us off guard.
1	**ParameterTypeMismatch** Parameters in the request do not match the data type that's expected. For example, this error happens if requests contain integers when Strings are expected.
2	**InvalidBodyRequestParameter** Parameters are likely missing, spelled incorrectly, or appear out of order in the request.
4	**InvalidCredentials** We are unable to find a match in our system for the username and password combination you provided.
6	**NotAuthorizedException** Be sure you're using the correct API key for the method you're calling. Admin features require a different access level than coder features.
7	**NullOrEmptyUrl** The url in the request seems to be missing.
8	**InvalidTag** Please check the entity names (e.g., encounters, notes) included in your request.
400	**Bad Request** The request could not be understood by the server due to malformed syntax (such as a bad query string).
401	**Unauthorized** A user token has expired, is invalid, or attempted to access an endpoint to which it does not have access.
404	**NotFound** The resource requested was not found or the URL used to make the call was invalid.
500	**Internal Server Error** This means something went wrong on our end. Please try again or contact your Atigeo Customer Service Representative.
503	**Service Unavailable** We're experiencing temporary down-time.

Java Sample Code

```
package com.atigeo.cac.service;

import java.util.Arrays;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.ResponseEntity;
import org.springframework.web.client.RestTemplate;

public class SampleCode
{
  public static class EMCodeAuditInput
  {
    public String emrText;
    public String code;
    public String patientType;
    public String serviceType1;
    public String serviceType2;
    public String serviceType3;
  }

  public static class EMCodeAuditResult
  {
    public String code;
    public String status;
    public String details;
    public String requiredPatientExam;
    public String requiredPatientHistory;
    public String requiredMedicalDecisionMaking;
    public int requiredNumComponents;
    public double patientHistoryScore;
    public double patientExamScore;
    public double medicalDecisionMakingScore;
    public double confidence;

    @Override
    public String toString()
    {
      StringBuilder sb = new StringBuilder();
      sb.append("code: " + code + "\n");
      sb.append("status: " + status.toString() + "\n");
      sb.append("details: " + details + "\n");
      if (status.equals("PREDICTION"))
      {
        sb.append("patientHistoryScore: " + patientHistoryScore + "\n");
        sb.append("patientExamScore: " + patientExamScore + "\n");
        sb.append("medicalDecisionMakingScore: " + medicalDecisionMakingScore + "\n");
        sb.append("requiredPatientExam: " + requiredPatientExam + "\n");
        sb.append("requiredPatientHistory: " + requiredPatientHistory + "\n");
        sb.append("requiredMedicalDecisionMaking: " + requiredMedicalDecisionMaking + "\n");
        sb.append("requiredNumComponents: " + requiredNumComponents + "\n");
        sb.append("confidence: " + confidence + "\n");
      }
      return sb.toString();
    }
  }

  public static void main(String[] args)
  {
    EMCodeAuditInput input = new EMCodeAuditInput();
    input.emrText = "Description: The patient is a 62-year-old female ...";
```

```
        input.code = "99214";
        input.patientType = "ESTABLISHED";
        input.serviceType1 = "OUTPATIENT_SERVICES";
        input.serviceType2 = "";
        input.serviceType3 = "";

        HttpHeaders requestHeaders = new HttpHeaders();
        requestHeaders.put("apikey", Arrays.asList("d8da0d64-7905-4185-9305-4531185befbe"));
        @SuppressWarnings({ "unchecked", "rawtypes" })
        HttpEntity<?> requestEntity = new HttpEntity(input, requestHeaders);

        EMCodeAuditResult output;
        try
        {
          RestTemplate client = new RestTemplate();
          ResponseEntity<EMCodeAuditResult> response = client.exchange(
              "https://services.xpatterns.com/coding-inference-ws/services/coding-service/audit-em-codes",
              HttpMethod.POST,
              requestEntity,
              EMCodeAuditResult.class);
          output = response.getBody();

          System.out.println("result is: " + output);
        }
        catch (Exception e)
        {
          e.printStackTrace();
        }
    }
}
```

Java Code Samples

I typically write my own code samples in Java (and previously in C#). I use Eclipse or Netbeans, depending on the project. In my current position, I document a Java REST APIs. However, I'm also quite familiar with Visual Studio and SOAP-based web services.

This section includes several different examples, which I have organized into three groups.

- REST API Calls
- Selenium UI Automation Tests
- JUnit Tests

REST API Calls

These samples represent a small sampling from hundreds of pages of code I have added to our API documentation.

- getEmrCode
- getEmrWrapper
- writeConfigValuesToFile

getEmrCode

This sample uses the return value from a REST call as input for the code block that loops through a data structure to retrieve medical codes.

```
public Set<EmrCode> getEmrCode(EmrWrapper emrWrapper) {

        EmrCode emrCode = null;
        Map<String, CodePillGroup> codePillGroups =
emrWrapper.getCodePillGroups();
        Set<CodePill> codePillSet;
        Set<EmrCode> emrCodeSet = new HashSet<>();
        if (codePillGroups != null) {
            for (Map.Entry<String, CodePillGroup> entry :
codePillGroups.entrySet()) {
                codePillSet = entry.getValue().getCodePills();

                //Let's make sure there's something in the codePillSet.
                if (codePillSet != null) {

                    //Iterate over codePillSet
                    for (CodePill codePill : codePillSet) {
                        //Add the code pills to our set to eliminate
duplicates.
                        emrCode = codePill.getCode();
                        emrCodeSet.add(emrCode);
                    }
                }
            }
        } else {
            System.out.println("Codes:  There are no codes associated with
this note.");
        }
        return emrCodeSet;
    }
```

getEmrWrapper

This is a sample REST call that returns a medical note.

```
public final EmrWrapper getEmrWrapper(int emrId, String visitType) throws
Exception {

        endpointApiPath = ConfigValues.getEndpointApiPath();
        baseAddress = ConfigValues.getStagingEndpointBaseAddress();
        apiKey = ConfigValues.getStoredJobApiKey();

        Map<String, String> vars = new HashMap<>();
        vars.put("emrId", new Integer(emrId).toString());
        String url = baseAddress + endpointApiPath + "note/" + emrId + "/data"
                + "?visitType=" + visitType;


        HttpHeaders requestHeaders = new HttpHeaders();
        requestHeaders.put(AuthConstants.API_KEY, Arrays.asList(apiKey));
        //Specify "json" or "xml".
        requestHeaders.put("Accept", Arrays.asList("application/xml"));

        CacRestTemplate client = new CacRestTemplate();
        HttpEntity<?> requestEntity = new HttpEntity(requestHeaders);

        EmrWrapper result = client.exchange(url, HttpMethod.GET,
                requestEntity, EmrWrapper.class, vars).getBody();


        return result;
    }
```

TestGetEmrDetails

Tests the getEMRDetails method, which returns patient and encounter metadata.

```
package com.atigeo.testing.api;

import static org.junit.Assert.assertTrue;

import java.util.Arrays;
import java.util.UUID;

import org.junit.Before;
import org.junit.Ignore;
import org.junit.Test;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.ResponseEntity;
import org.springframework.util.Assert;
import org.springframework.web.client.xPatternsRestTemplate;

import com.atigeo.cac.api.rest.model.*;
import com.atigeo.testing.api.util.ConfigValues;
import com.xpatterns.xPatternsRestClientException;
import com.xpatterns.security.api.authorization.AuthConstants;

public class TestGetEmrDetails {

	private static String baseAddress;
	private static String endpointApiPath;
	private String testName;
	private static String emrId;
	private String status;

	@Before
	public void setUp() throws Exception {

	System.setProperty("http.proxyHost", "127.0.0.1");
	System.setProperty("http.proxyPort", "8888");

	baseAddress = ConfigValues.getEndpointBaseAddressCac();
	endpointApiPath = ConfigValues.getEndpointApiPath();
	if (baseAddress.contains("ubuntu")){
	createAndAutoCodeEmrInpatient();
	}

	}

	@Test
	public void createNote() {

	// adminUser
```

```
	String apiKey = getApiKey();

	EmrLabelAndText handle = new EmrLabelAndText();
	setTestName();
	handle.setLabel(getTestName());
	handle.setEmrText("There is 1 to 2+ aortic regurgitation easily seen,
but no aortic stenosis.2. Mild tricuspid regurgitation with only mild increase
in right heart"
	+ "pressures, 30-35 mmHg maximum.");

	HttpHeaders requestHeaders = new HttpHeaders();
	requestHeaders.put(AuthConstants.API_KEY, Arrays.asList(apiKey));

	xPatternsRestTemplate client = new xPatternsRestTemplate();
	@SuppressWarnings({ "unchecked", "rawtypes" })
	HttpEntity<?> requestEntity = new HttpEntity(handle, requestHeaders);

	String url = baseAddress + endpointApiPath + "note/import";

	try {
	ResponseEntity<Object> wrapper = client.exchange(url,
	HttpMethod.POST, requestEntity, Object.class);
	System.out.println("wrapper.getBody() " + wrapper.getBody());
	setEmrId(wrapper.getBody().toString());

	System.out.println("emrId: " + getEmrId());
	System.out.println(wrapper.getStatusCode().toString());
	setStatus(wrapper.getBody().toString());
	assertTrue(wrapper.getStatusCode().toString().equals("200"));
	System.out.println("emrId: " + getEmrId());

	} catch (xPatternsRestClientException ex) {
	System.out.println(ex.getMessage());
	ex.printStackTrace();
	// Assert.isTrue(false);

	}

	}

	public void createAndAutoCodeEmrInpatient() {

		//testUser
	String apiKey = "3a52b9e7-42f3-4d9d-8f55-185c4b92a0f5";


	EmrLabelAndText handle = new EmrLabelAndText();
		setTestName();
		handle.setLabel(getTestName());
		handle.setEmrText("There is 1 to 2+ aortic regurgitation easily
seen, but no aortic stenosis.2. Mild tricuspid regurgitation with only mild
increase in right heart"
```

```
      + "pressures, 30-35 mmHg maximum.");

            HttpHeaders requestHeaders = new HttpHeaders();
            requestHeaders.put(AuthConstants.API_KEY, Arrays.asList(apiKey));

            xPatternsRestTemplate client = new xPatternsRestTemplate();
            @SuppressWarnings({ "unchecked", "rawtypes" })
            HttpEntity<?> requestEntity = new
HttpEntity(handle,requestHeaders);

            String url = baseAddress + endpointApiPath +
"note?produtType=inpatient";

            try {
            ResponseEntity<Object> wrapper = client.exchange(url,
HttpMethod.POST, requestEntity, Object.class);
            System.out.println(wrapper.getStatusCode().toString());
            assertTrue(wrapper.getStatusCode().toString().equals("200"));

            } catch (xPatternsRestClientException ex) {
               System.out.println(ex.getMessage());
               ex.printStackTrace();
               assertTrue(false);

            }

      }

      public void setTestName() {

      // Random string for EMR Label
      UUID guid = UUID.randomUUID();
      testName = "CreateEMRToTestListEMRAPI" + guid.toString().substring(0,
7);

      }

      public String getTestName() {
      String tName = this.testName;
      return tName;
      }

      public void setEmrId(String emrId) {

      this.emrId = emrId;

      }

      public String getEmrId() {
      return emrId;
      }
```

```
	public void setStatus(String status) {
	this.status = status;
	}

	public String getStatus() {

	return status;
	}
}
```

Selenium UI Automation Tests

These code samples represent a small sampling from hundreds of Selenium UI automation tests I have created in the past year.

- TestEncounterDataView
- TestLoginPage
- TestGetEmrDetails

TestEncounterDataView

Verifies data is displayed on the portal encounter data view page in Chrome.

```
package com.atigeo.selenium.encounters;

import static org.junit.Assert.*;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.openqa.selenium.*;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

import com.atigeo.selenium.LoginPage;
import com.atigeo.selenium.helper.EncounterImporter;
import com.atigeo.selenium.helper.LocalSystemProperties;

public class TestEncounterDataView {

	//private StringBuffer verificationErrors = new StringBuffer();
	private WebDriver selenium;
	private String encounterId;
	private int noteCount;
	private String userType;
	private String productType;

	@Before
	public void setUp() throws Exception {
	LocalSystemProperties.setLocalSystemProperties();
	userType = "coder"; //admin, superUser
	productType = "inpatient"; // inpatient, outpatient, physician
	noteCount = 9; // Must be 1 - 9;
	LoginPage loginPage = new LoginPage();
	WebDriver selenium = loginPage.userLogin(productType, userType);
	this.selenium = selenium;
	importEncounter();
	selenium.get("http://cac-test1.life.atigeo.com:9090/portal/" +
productType + "/encounters/" + getEncounterId() + "/view");
	selenium.findElement(By.cssSelector("#tab-data"));
	}

	@Test
	public void shouldDisplayDRGData() throws Exception {

	WebDriverWait wait = new WebDriverWait(selenium, 30L);

wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector("#inpu
t-primary-diagnosis")));
```

```
        selenium.findElement(By.cssSelector("#input-primary-
diagnosis")).sendKeys("002.0");
        selenium.findElement(By.cssSelector("#btn-status-recode")).click();


        wait = new WebDriverWait(selenium, 30L);

wait.until(ExpectedConditions.textToBePresentInElement(By.cssSelector("#reimbu
rsement-value"), "$5034.12"));



        assertEquals("$5034.12",
selenium.findElement(By.cssSelector("#reimbursement-value")).getText());
        assertEquals("-\n315", selenium.findElement(By.cssSelector("#input-
reimbursement-code")).getText());
        assertEquals("Other circulatory system diagnoses w CC",
selenium.findElement(By.cssSelector("#input-reimbursement-
description")).getText());
        }

        @After
        public void tearDown() {

        logOut();
        selenium.quit();
        }

        public void logOut() {

        selenium.findElement(By.id("webuser")).click();
        selenium.findElement(By.id("btn-logout")).click();

        }


        public void shouldImportEncounter() {

        EncounterImporter importer = new EncounterImporter();
        try {
        String encounterId = importer.importEncounter(productType,
        userType, noteCount);
        setEncounterId(encounterId);
        System.out.println("encounterId = " + encounterId);

        } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        }

        }
```

```
      public String getEncounterId() {
      return encounterId;
      }

      public void setEncounterId(String encounterId) {
      this.encounterId = encounterId;
      }

    public void sleep(int length) {
        try {
            Thread.sleep(length);
        } catch (InterruptedException ex) {
            Thread.currentThread().interrupt();
        }
    }


}
```

TestLoginPage

Tests user login with Chrome.

```
package com.atigeo.selenium.pageobjects;

import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import com.atigeo.testing.api.util.ConfigValues;

public class LoginPage {

	WebDriver selenium;
	private String loginUrl;
	private String username;
	private String password;

	public shouldLoginUser() {
	setLocalSystemProperties();

	}

	public WebDriver userLogin(String productType, String userType)
	throws Exception {

	// Sets wait time for entire driver session
	selenium.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

	setLoginNameAndPassword(userType);
	setProductType(productType);

	// Login to system
	String baseUrl = ConfigValues.getEndpointBaseAddressCac();

	selenium.get(baseUrl + getLoginUrl());
	selenium.findElement(By.id("j_username")).clear();
	selenium.findElement(By.id("j_username")).sendKeys(getUserName());
	selenium.findElement(By.id("j_password")).clear();
	selenium.findElement(By.id("j_password")).sendKeys(getPassword());
	selenium.findElement(By.id("btn-login")).click();

	return selenium;

	}

	public void setLocalSystemProperties() {

	System.setProperty("http.proxyHost", "127.0.0.1");
	System.setProperty("http.proxyPort", "8888");
```

```
String name = System.getProperties().getProperty("user.name");

if (name.toLowerCase().equals("philg")) {
System.setProperty("webdriver.chrome.driver",
"C:\\Users\\philg\\workspace\\chromedriver.exe");
}

selenium = new ChromeDriver();

}

public void setLoginNameAndPassword(String userType) throws Exception {

if (userType == "super") {

username = ConfigValues.getSuperUserName();
password = ConfigValues.getSuperUserPassword();

} else if (userType == "admin") {
username = ConfigValues.getAdminUserName();
password = ConfigValues.getAdminUserPassword();

} else {
username = ConfigValues.getDefaultUserName();
password = ConfigValues.getDefaultUserPassword();

}

}

public void setProductType(String productType) throws Exception {

if (productType == "inpatient") {

loginUrl = ConfigValues.getEndpointInpatientPortalPath();

} else if (productType == "physician") {
loginUrl = ConfigValues.getEndpointPhysicianPortalPath();

} else {
loginUrl = ConfigValues.getEndpointOutpatientPortalPath();

}

}
public String getLoginUrl() {
String url = loginUrl;
return url;

}

public String getUserName() {
```

```
        String name = username;

        return name;
        }

        public String getPassword() {
        String pw = password;
        return pw;

        }
}
```

JUnit Tests

These code samples represent a small sampling from several dozen JUnit integration tests I have created in the past year.

- TestAnalysisAndInference
- TestGetPublishedEmrCount
- TestCreateAssignPublishOpenRejectApis

TestAnalysisAndInference

End-to-end test of code retrieval. Output of analysis is sent as an argument to the inference service.

```
package com.atigeo.testing.api;

import static org.junit.Assert.*;

import java.util.Arrays;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;

import org.junit.BeforeClass;
import org.junit.Test;
import org.springframework.http.HttpHeaders;

import com.atigeo.cac.common.assertions.Assertion;
import com.atigeo.cac.common.assertions.ConceptType;
import com.atigeo.cac.common.assertions.MultiNoteAnalysisInput;
import com.atigeo.cac.common.assertions.VisitType;
import com.atigeo.testing.api.cac.AnalysisAndInferenceHelper;
import com.atigeo.testing.api.util.ConfigValues;
import com.xpatterns.security.api.authorization.AuthConstants;

public class TestAnalysisAndInference {


	private static String baseAddress;
	private static String analysisEndpoint;
	private static String inferenceEndpoint;


	@BeforeClass
	public static void setUp() throws Exception {
	System.setProperty("http.proxyHost", "127.0.0.1");
	System.setProperty("http.proxyPort", "8888");
	baseAddress = ConfigValues.getEndpointBaseAddressCac();
	analysisEndpoint = ConfigValues.getAnalysisEndpointApiPath();
	inferenceEndpoint = ConfigValues.getInferenceEndpointApiPath();
	}


	@Test
	public void shouldReturnMedicalCodes() throws Exception {

	String apiKey = ConfigValues.getHimssUserApiKey();
	String affiliation = ConfigValues.getHimssUserAffiliation();

	HttpHeaders requestHeaders = new HttpHeaders();
```

```
		requestHeaders.put(AuthConstants.API_KEY, Arrays.asList(apiKey));
		Map<String, String> addtionalFields = new HashMap<String, String>();
		addtionalFields.put(ConceptType.AGE.toString(), "15");
		addtionalFields.put(ConceptType.GENDER.toString(), "Male");
		addtionalFields.put(ConceptType.ALLERGY.toString(), "Pollen");
		MultiNoteAnalysisInput input = new MultiNoteAnalysisInput();


		String emrText = "hyperlipidemia";
		Map<String,String> noteMap = new HashMap<String,String>();
		noteMap.put("123", emrText);
		input.setNotes(noteMap);
		input.setAdditionalFields(addtionalFields);


		String analysisUrl = baseAddress + analysisEndpoint;
		String inferenceUrl = baseAddress + inferenceEndpoint;
		try{
		Assertion[] analysisAssertions =
AnalysisAndInferenceHelper.analyze(input, analysisUrl, apiKey);
		assertNotNull(analysisAssertions);
		analysisAssertions =
AnalysisAndInferenceHelper.addRequiredAssertions(analysisAssertions, new
Date(), VisitType.OUTPATIENT, "02", "5200");
		Assertion[] inferenceAssertions =
AnalysisAndInferenceHelper.getCodingAssertions(apiKey, affiliation,
analysisAssertions, inferenceUrl);
		assertNotNull(inferenceAssertions);
		}
		catch(Exception e){
		assertTrue(false);
		}

		}

}
```

TestGetPublishedEmrCount

Confirms the system returns the number of published medical records.

```
package com.atigeo.testing.api;

import static org.junit.Assert.assertTrue;

import java.util.Arrays;
import java.util.UUID;

import org.junit.Before;
import org.junit.Test;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.ResponseEntity;
import org.springframework.util.Assert;
import org.springframework.web.client.xPatternsRestTemplate;

import com.atigeo.cac.api.rest.model.*;
import com.atigeo.testing.api.util.ConfigValues;
import com.xpatterns.xPatternsRestClientException;
import com.xpatterns.security.api.authorization.AuthConstants;

public class TestGetPublishedEmrCount {

	private static String baseAddress;
	private static String endpointApiPath;
	private String testName;

	@Before
	public void setUp() throws Exception {

	System.setProperty("http.proxyHost", "127.0.0.1");
	System.setProperty("http.proxyPort", "8888");
	baseAddress = ConfigValues.getEndpointBaseAddressCac();
	endpointApiPath = ConfigValues.getEndpointApiPath();

	if (baseAddress.contains("ubuntu")){
	createAndAutoCodeEmrInpatient();
	}

	}

	@Test
	public void shouldGetMostUsedCodes() {

	// defaultUser
	String apiKey = getApiKey();

	HttpHeaders requestHeaders = new HttpHeaders();
```

```
	requestHeaders.put(AuthConstants.API_KEY, Arrays.asList(apiKey));

	String url = baseAddress + endpointApiPath + "reports/publishedemr/all";

	xPatternsRestTemplate client = new xPatternsRestTemplate();
	@SuppressWarnings({ "unchecked", "rawtypes" })
	HttpEntity<?> requestEntity = new HttpEntity(requestHeaders);


	try {

	ResponseEntity<Object> codeReport = client.exchange(url, HttpMethod.GET,
requestEntity,
	Object.class);
	System.out.println(codeReport.getBody());
	    assertTrue(codeReport.getStatusCode().toString().equals("200"));

	} catch (xPatternsRestClientException ex) {
	System.out.println(ex.getMessage());
	ex.printStackTrace();
	assertTrue(false);

	}

	}

	public void createAndAutoCodeEmrInpatient() {

		//testUser
	String apiKey = getApiKey();


	EmrLabelAndText handle = new EmrLabelAndText();
		setTestName();
		handle.setLabel(getTestName());
		handle.setEmrText("There is 1 to 2+ aortic regurgitation easily
seen, but no aortic stenosis.2. Mild tricuspid regurgitation with only mild
increase in right heart"
	+ "pressures, 30-35 mmHg maximum.");

		HttpHeaders requestHeaders = new HttpHeaders();
		requestHeaders.put(AuthConstants.API_KEY, Arrays.asList(apiKey));

		xPatternsRestTemplate client = new xPatternsRestTemplate();
		@SuppressWarnings({ "unchecked", "rawtypes" })
		HttpEntity<?> requestEntity = new
HttpEntity(handle,requestHeaders);

		String url = baseAddress + endpointApiPath +
"note?produtType=inpatient";

		try {
```

```
            ResponseEntity<Object> wrapper = client.exchange(url,
HttpMethod.POST, requestEntity, Object.class);
            System.out.println(wrapper.getStatusCode().toString());
            assertTrue(wrapper.getStatusCode().toString().equals("200"));

            } catch (xPatternsRestClientException ex) {
               System.out.println(ex.getMessage());
               ex.printStackTrace();
            //   Assert.isTrue(false);

            }

      }

      public void setTestName() {

      // Random string for EMR Label
      UUID guid = UUID.randomUUID();
      testName = "CreateEMRToTestMostUsedCodesAPI" +
guid.toString().substring(0, 7);

      }

      public String getTestName() {
      String tName = this.testName;
      return tName;
      }

}
```

TestCreateAssignPublishOpenRejectApis

End-to-end tests that exercise each step in an encounter workflow.

```
package com.atigeo.testing.api;

import static org.junit.Assert.assertTrue;

import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;
import java.util.UUID;

import org.junit.Before;
import org.junit.Ignore;
import org.junit.Test;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.ResponseEntity;
import org.springframework.web.client.xPatternsRestTemplate;

import com.atigeo.cac.api.rest.model.*;
import com.atigeo.testing.api.util.ConfigValues;
import com.xpatterns.xPatternsRestClientException;
import com.xpatterns.security.api.authorization.AuthConstants;

public class TestCreateAssignPublishOpenRejectApis {

	private static String baseAddress;
	private static String endpointApiPath;

	private String testName;
	private static String emrId;
	private String status;

	@Before
	public void setUp() throws Exception {

	System.setProperty("http.proxyHost", "127.0.0.1");
	System.setProperty("http.proxyPort", "8888");
	baseAddress = ConfigValues.getEndpointBaseAddressCac();
	endpointApiPath = ConfigValues.getEndpointApiPath();

	}
	@Test
	public void shouldPublishNote() {

	// defaultUser
	String apiKey = getApiKey();

	HttpHeaders requestHeaders = new HttpHeaders();
```

```
requestHeaders.put(AuthConstants.API_KEY, Arrays.asList(apiKey));

xPatternsRestTemplate client = new xPatternsRestTemplate();
@SuppressWarnings({ "unchecked", "rawtypes" })
HttpEntity<?> requestEntity = new HttpEntity(requestHeaders);

String url = baseAddress + endpointApiPath + "note/" + 805
+ "/publish";
System.out.println(url.toString());

try {
ResponseEntity<Object> wrapper = client.exchange(url,
HttpMethod.PUT, requestEntity, Object.class);
assertTrue(wrapper.getStatusCode().toString().equals("200"));

} catch (xPatternsRestClientException ex) {
System.out.println(ex.getMessage());
ex.printStackTrace();
 Assert.isTrue(false);

}

}

@Test
public void shouldOpenNote() {

// defaultUser
String apiKey = getApiKey();

HttpHeaders requestHeaders = new HttpHeaders();
requestHeaders.put(AuthConstants.API_KEY, Arrays.asList(apiKey));

xPatternsRestTemplate client = new xPatternsRestTemplate();
@SuppressWarnings({ "unchecked", "rawtypes" })
HttpEntity<?> requestEntity = new HttpEntity(requestHeaders);

String url = baseAddress + endpointApiPath + "/note/" + 805
+ "/unpublish";
System.out.println(url.toString());

try {
ResponseEntity<Object> wrapper = client.exchange(url,
HttpMethod.PUT, requestEntity, Object.class);
assertTrue(wrapper.getStatusCode().toString().equals("200"));

} catch (xPatternsRestClientException ex) {
System.out.println(ex.getMessage());
ex.printStackTrace();
}

}
```

```
	public void setTestName() {

	// Random string for EMR Label
	UUID guid = UUID.randomUUID();
	testName = "PublishEmrTest_" + guid.toString().substring(0, 7);

	}

	public String getTestName() {
	String tName = this.testName;
	return tName;
	}

	public void setEmrId(String emrId) {

	this.emrId = emrId;

	}

	public String getEmrId() {
	return emrId;
	}

	public void setStatus(String status) {
	this.status = status;
	}

	public String getStatus() {

	return status;
	}

}
```