Paper 032-2012

SAS® BI Content Syndication with the REST Framework

Mike Vanderlinden, Experis, Portage, MI

Abstract

This paper discusses a method to make the SAS® BI content available to other parts of your organization through the HTTP protocol. Data, Information Maps, Metadata, Reports, Stored Processes can be made available to any system that can talk HTTP, whether it's a browser on your desktop, a SharePoint website, a mobile device or anything else including a competitor's portlet. The REST framework is leveraged to provide the ability to reach SAS® content with a zero-footprint client, as well as a powerful set of APIs to allow non-SAS developers to create rich interfaces.

Introduction

A few months ago, I was tasked to figure out a mechanism to execute SAS models from the Cognos software as a proof-of-concept for a potential customer. The solution needed to include a rich user interface to modify the model parameters and could not use the SAS EBI platform since it was not licensed by the customer. However, they did license SAS Integration Technologies.

Needless to say that the first thought that came through my mind was: “Why not use SAS as the reporting platform?” After all, SAS is at least as good as the other BI products for pure reporting and the complexity of integrating disparate systems is often under-estimated. But because I believe there is an architect in me, I decided to take on the challenge. Years of experience and “added wisdom” made me realize that while I’m a great fan of the SAS platform, others prefer to go with the best-of-breed approach and use SAS for what it does best, namely analytics, and leave other activities to systems they gauge superior in functionalities.

I started researching potential solutions that would include Web Services, SOA, and other established or emerging technologies that would allow me to make the SAS BI content accessible by other systems. I figured if I could find a solution for Cognos, it would most likely generalize easily to other similar platforms.

I had tinkered around with the SAS Java API to access the SAS Metadata, connect to a workspace server and launch a SAS program from a Java interface. I knew I could build an application or in this case a Cognos portlet (or whatever the equivalent of it is), that would do the job. This seemed like a solution that would entail a lot of development on the client side and would not be generic enough for my ultimate goal, which was to design and develop a simple mechanism to surface any SAS BI content from various platforms/systems.

I then stumbled upon Dr. Roy Fielding’s dissertation which, at a high level, describes why and how the Web became so popular. The REST framework and its various implementations followed soon after as a means to simplify and standardize communications with web services.

Objectives

The goal of this endeavor was to build a solution that would create a bridge between clients and the SAS system through the SAS metadata interface. The following constraints were decided arbitrarily and included in the scope:

- Client-server communication must be through the HTTP protocol
- Server may be producing XML or HTML (HTML allows zero-footprint clients in many cases)
- Requires SAS Integration Technologies
- Client may be any electronic device that can implement the HTTP protocol

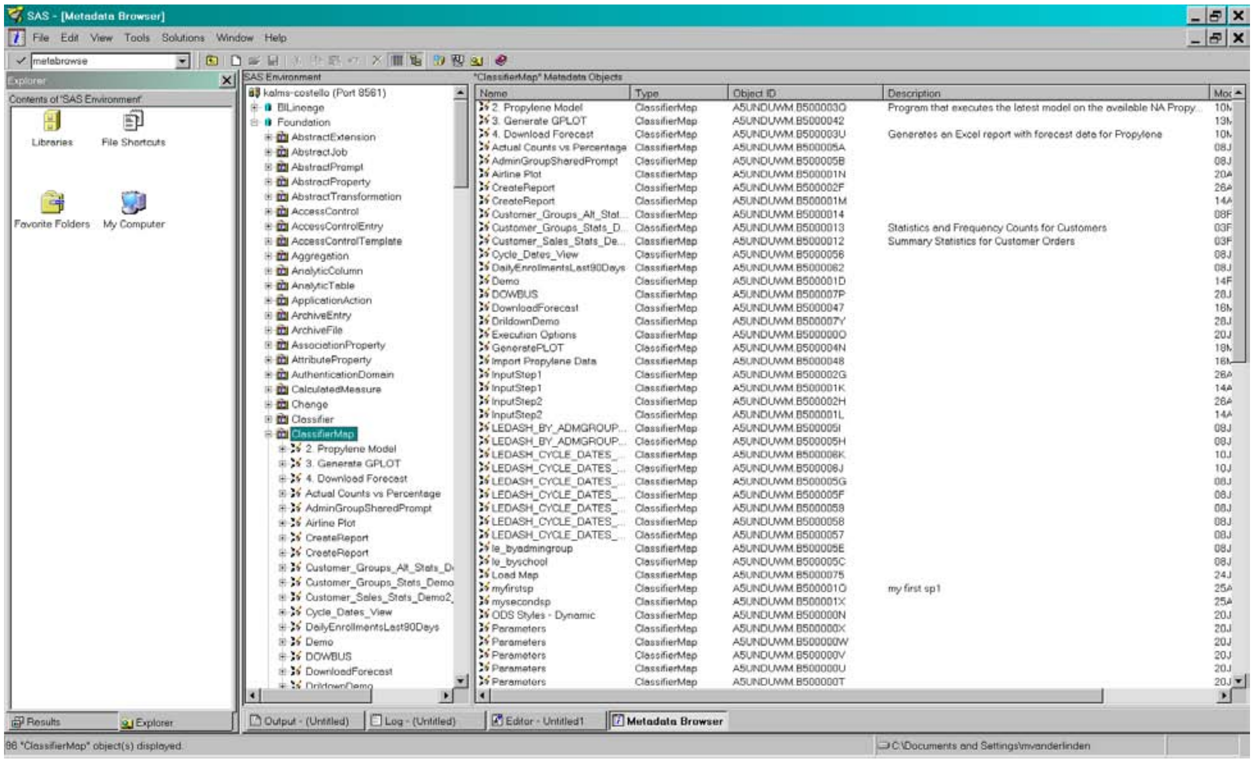
SAS Metadata Model

In a typical environment, SAS Metadata is managed through the SAS Management Console by SAS administrators. Java APIs also allow developers to build custom applications that interface with the SAS Metadata server to query, create, update or delete metadata definitions.

SAS Foundation Services is a set of infrastructure and services to create applications that interface with the different components of a SAS Intelligence platform. It includes services for managing IOM server connections, locating and binding to services, event notification, repository federation and searching, logging, publishing, authenticating and security management, context and resource management, Stored Process retrieval and execution, and user session management.

All these services are extensively used within the SAS Web Applications such as Web Report Studio and the SAS Portal, but are also available to developers to design new applications or new interfaces to the SAS platform.

A simple way to figure out how metadata definitions are organized and hierarchically dependent is to use the SAS editor and type in the 'metabrowse' command. You will be prompted for information to connect to a Metadata Server, and then you will be presented with a tree structure of all metadata definitions in the repositories.



```
SAS Environment
kalms-costello (Port 8561)
  BILineage
  Foundation
    AbstractExtension
    AbstractJob
    AbstractPrompt
    AbstractProperty
    AbstractTransformation
    AccessControl
    AccessControlEntry
    AccessControlTemplate
    Aggregation
    AnalyticColumn
    AnalyticTable
    ApplicationAction
    ArchiveEntry
    ArchiveFile
    AssociationProperty
    AttributeProperty
    AuthenticationDomain
    CalculatedMeasure
    Change
    Classifier
    ClassifierMap
      2. Propylene Model
      3. Generate GPLOT
      4. Download Forecast
      Actual Counts vs Percentage
      AdminGroupSharedPrompt
      Airline Plot
      CreateReport
      CreateReport
      Customer_Groups_Alt_Stats_D
      Customer_Groups_Stats_Demo
      Customer_Sales_Stats_Demo2_
      Cycle_Dates_View
      DailyEnrollmentsLast90Days
      Demo
      DOWBUS
      DownloadForecast
      DrildownDemo
```

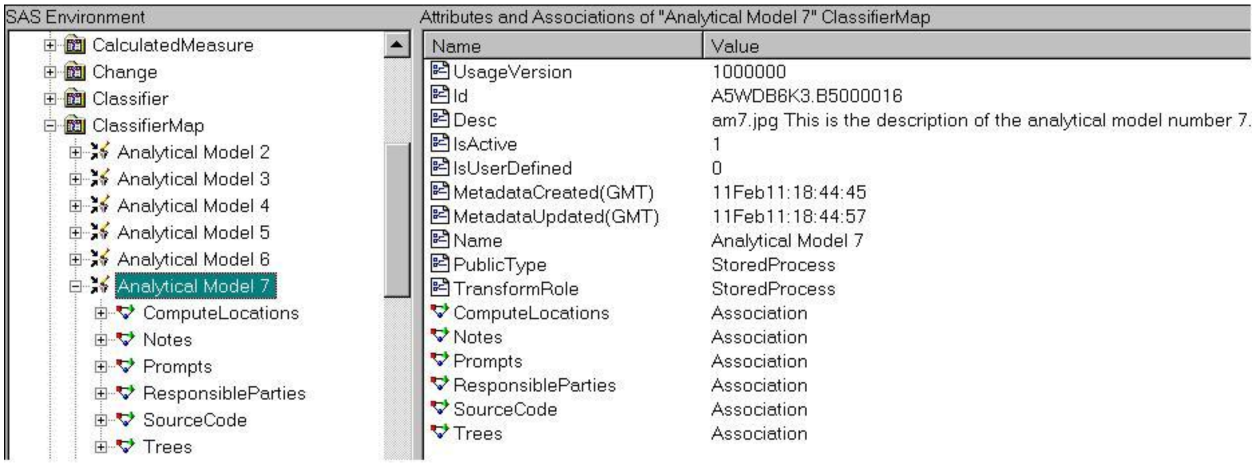
"ClassifierMap" Metadata Objects

Name	Type	Object ID	Description	Moc
2. Propylene Model	ClassifierMap	A5UNDUWM.B500003O	Program that executes the latest model on the available NA Propy...	10M
3. Generate GPLOT	ClassifierMap	A5UNDUWM.B5000042		13M
4. Download Forecast	ClassifierMap	A5UNDUWM.B500003U	Generates an Excel report with forecast data for Propylene	10M
Actual Counts vs Percentage	ClassifierMap	A5UNDUWM.B500005A		08J
AdminGroupSharedPrompt	ClassifierMap	A5UNDUWM.B500005B		08J
Airline Plot	ClassifierMap	A5UNDUWM.B500001N		20A
CreateReport	ClassifierMap	A5UNDUWM.B500002F		26A
CreateReport	ClassifierMap	A5UNDUWM.B500001M		14A
Customer_Groups_Alt_Stat...	ClassifierMap	A5UNDUWM.B5000014		08F
Customer_Groups_Stats_D...	ClassifierMap	A5UNDUWM.B5000013	Statistics and Frequency Counts for Customers	03F
Customer_Sales_Stats_De...	ClassifierMap	A5UNDUWM.B5000012	Summary Statistics for Customer Orders	03F
Cycle_Dates_View	ClassifierMap	A5UNDUWM.B5000056		08J
DailyEnrollmentsLast90Days	ClassifierMap	A5UNDUWM.B5000062		08J
Demo	ClassifierMap	A5UNDUWM.B500001D		14F
DOWBUS	ClassifierMap	A5UNDUWM.B500007P		28J
DownloadForecast	ClassifierMap	A5UNDUWM.B5000047		16M
DrildownDemo	ClassifierMap	A5UNDUWM.B500007Y		28J
Execution Options	ClassifierMap	A5UNDUWM.B500000O		20J
GeneratePLOT	ClassifierMap	A5UNDUWM.B500004N		18M
Import Propylene Data	ClassifierMap	A5UNDUWM.B5000048		16M
InputStep1	ClassifierMap	A5UNDUWM.B500002G		26A
InputStep1	ClassifierMap	A5UNDUWM.B500001K		14A
InputStep2	ClassifierMap	A5UNDUWM.B500002H		26A
InputStep2	ClassifierMap	A5UNDUWM.B500001L		14A
LEDASH_BY_ADMGROUP...	ClassifierMap	A5UNDUWM.B500005I		08J
LEDASH_BY_ADMGROUP...	ClassifierMap	A5UNDUWM.B500005H		08J
LEDASH_CYCLE_DATES_...	ClassifierMap	A5UNDUWM.B500006K		10J
LEDASH_CYCLE_DATES_...	ClassifierMap	A5UNDUWM.B500006J		10J
LEDASH_CYCLE_DATES_...	ClassifierMap	A5UNDUWM.B500005G		08J
LEDASH_CYCLE_DATES_...	ClassifierMap	A5UNDUWM.B500005F		08J
LEDASH_CYCLE_DATES_...	ClassifierMap	A5UNDUWM.B5000059		08J
LEDASH_CYCLE_DATES_...	ClassifierMap	A5UNDUWM.B5000058		08J
LEDASH_CYCLE_DATES_...	ClassifierMap	A5UNDUWM.B5000057		08J
le_byadmingroup	ClassifierMap	A5UNDUWM.B500005E		08J
le_byschool	ClassifierMap	A5UNDUWM.B500005C		08J
Load Map	ClassifierMap	A5UNDUWM.B5000075		24J
myfirstsp	ClassifierMap	A5UNDUWM.B500001O	my first sp1	25A
mysecondsp	ClassifierMap	A5UNDUWM.B500001X		25A
ODS Styles - Dynamic	ClassifierMap	A5UNDUWM.B500000N		20J
Parameters	ClassifierMap	A5UNDUWM.B500000X		20J
Parameters	ClassifierMap	A5UNDUWM.B500000W		20J
Parameters	ClassifierMap	A5UNDUWM.B500000V		20J
Parameters	ClassifierMap	A5UNDUWM.B500000U		20J
Parameters	ClassifierMap	A5UNDUWM.B500000T		20J

86 "ClassifierMap" object(s) displayed.

C:\Documents and Settings\mvanderlinden

For instance, retrieving a list of Stored Processes entails retrieving ClassifierMap objects which have a public type of “StoredProcess”. The different Associations objects allow us to retrieve additional information about a specific Stored Process.



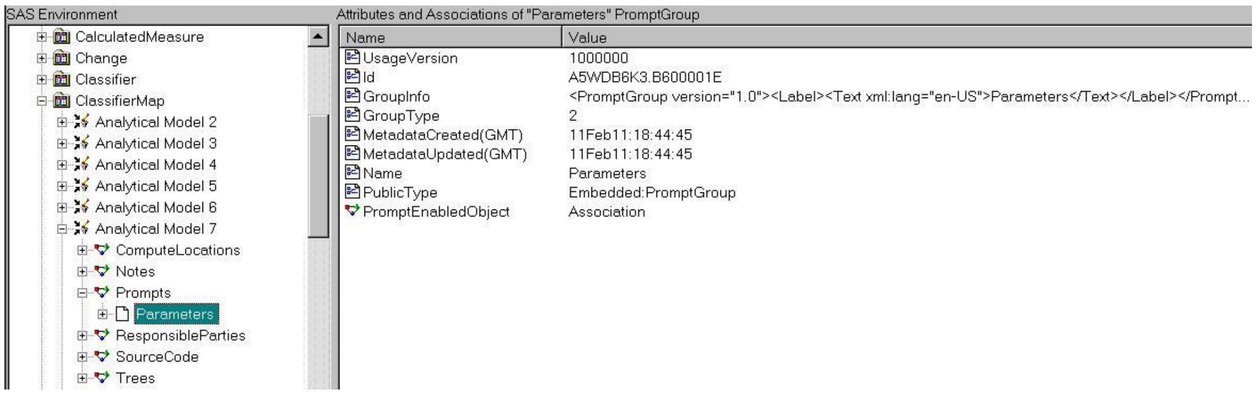
SAS Environment

```
⊞ CalculatedMeasure
⊞ Change
⊞ Classifier
⊟ ClassifierMap
   ⊞ Analytical Model 2
   ⊞ Analytical Model 3
   ⊞ Analytical Model 4
   ⊞ Analytical Model 5
   ⊞ Analytical Model 6
   ⊟ Analytical Model 7
      ⊞ ComputeLocations
      ⊞ Notes
      ⊞ Prompts
      ⊞ ResponsibleParties
      ⊞ SourceCode
      ⊞ Trees
```

Attributes and Associations of "Analytical Model 7" ClassifierMap

Name	Value
UsageVersion	1000000
Id	A5WDB6K3.B5000016
Desc	am7.jpg This is the description of the analytical model number 7.
IsActive	1
IsUserDefined	0
MetadataCreated(GMT)	11Feb11:18:44:45
MetadataUpdated(GMT)	11Feb11:18:44:57
Name	Analytical Model 7
PublicType	StoredProcess
TransformRole	StoredProcess
ComputeLocations	Association
Notes	Association
Prompts	Association
ResponsibleParties	Association
SourceCode	Association
Trees	Association

If we wanted to retrieve the prompts defined for this Stored Process, we could query the Prompts Association object which conveniently returns XML with all prompt definitions.



SAS Environment

```
⊞ CalculatedMeasure
⊞ Change
⊞ Classifier
⊟ ClassifierMap
   ⊞ Analytical Model 2
   ⊞ Analytical Model 3
   ⊞ Analytical Model 4
   ⊞ Analytical Model 5
   ⊞ Analytical Model 6
   ⊟ Analytical Model 7
      ⊞ ComputeLocations
      ⊞ Notes
      ⊟ Prompts
         ⊞ Parameters
      ⊞ ResponsibleParties
      ⊞ SourceCode
      ⊞ Trees
```

Attributes and Associations of "Parameters" PromptGroup

Name	Value
UsageVersion	1000000
Id	A5WDB6K3.B600001E
GroupInfo	<PromptGroup version="1.0"><Label><Text xml:lang="en-US">Parameters</Text></Label></Prompt...
GroupType	2
MetadataCreated(GMT)	11Feb11:18:44:45
MetadataUpdated(GMT)	11Feb11:18:44:45
Name	Parameters
PublicType	Embedded:PromptGroup
PromptEnabledObject	Association

These APIs thus allow us to create a custom application to query the SAS Metadata to retrieve definitions. Other APIs and services are designed to use these definitions in the context of a user session and so we could execute a Stored Process, prompt the users for parameters… all while leveraging the security framework from SAS Integration Technologies.

However, in my quest for absolute simplicity (meaning I don’t expect anyone to become a Java developer and start building complex applications overnight) I needed to find a way to make the same capabilities and features available through a zero-footprint client.

By using Web Services, I could easily provide a set of services on top of the SAS APIs to provide such functionalities as retrieving metadata objects and performing actions on them based on their type. In order to implement clients that interface with these Web Services, I would have to develop code for each of those services.

A simple mechanism for exchanging information such as the HTTP protocol is the most intuitive choice. REST-based Web Services provide just that; they function just like ordinary Web Services and while they do not need to use the HTTP protocol, they are mainly designed on the very same premises that made HTTP and the Web so popular.

RESTful Framework

The REST (Representational State Transfer) architectural style is a software architecture for distributed hypermedia. It was introduced by Dr. Roy Fielding in his doctoral dissertation which proposes a simpler-than-before client-server architecture based on the success of the World Wide Web.

I prefer the term “framework” because in order to fully take advantage of the benefits of developing REST-based Web Services, best practices/constraints must be respected by the developer.

REST Constraints

In order to be RESTful, an architecture must abide several constraints. There must be a clear separation between the client and the server as they communicate through a uniform interface. This allows clients and servers to be developed independently and generally promotes higher portability.

Sessions between a client and a server are stateless. This means the client context is never stored on the server, unless the state is addressable as a resource on the server. This provides better reliability and scalability on the server side.

Server responses must clearly define themselves as cacheable (or not) to ensure clients are always aware whether the information they receive is up-to-date. This also improves scalability and performance since it can reduce unnecessary interactions between the client and the server.

Intermediaries between the client and the server provide scalability, performance and sometimes enforce security constraints. Clients must not be able to tell whether they are connected to an end server or an intermediary, this is referred to as a layered system.

Following these constraints promotes simplicity, portability and performance which is just what we need to build a SAS BI Content Syndication solution.

REST Characteristics

A REST-based solution looks a lot like SOA (Service Oriented Architecture), however at the center of the REST architecture is the resource, and so it is sometimes referred to as a Resource Oriented Architecture.

A resource is any source of information that can be referenced with a global identifier. For our implementation, which is based on the HTTP protocol, global identifiers are URIs and resources are the SAS BI content such as tables, users, information maps, etc.

These URIs are highly visible and offer a great sense of addressability because each is human readable and provides simple options for manipulating representations. For instance, the following URI requests the list of information maps from the “Internal” folder in XML:
http://services.experis.com/SASREST/InformationMaps;folder=Internal/XML.

Resources are manipulated through a uniform interface, in our case HTTP methods such as PUT, POST, GET and DELETE. This paper showcases a prototype that only implements GET services since we will only consume information.

It is important to note that resources are separate from representations. For example, we can request a table which will return the records in any selected representation such as HTML, XML or comma-separated.

Before we delve into our implementation of a prototype, another aspect of the REST architecture is Hypermedia as the engine of application state. This simply means that transitions between resources are done through hypermedia actions on the server, eg. Hyperlinks. This is exactly how the World Wide Web works. When passively surfing the Web, most of us start with a search engine. We enter some keywords and from there we can easily spend hours in front of our computer without typing anything more, just clicking links that we find interesting. This connectedness is omni-present in the design of service responses.

When designing a set of REST-based Web Services, one must first identify the resources and design a resource model for interacting with them. This resource model is typically documented as a table which lists resources, their methods, representations, options and a description.

URI	Method	Representation	Options	Description
http://...	GET	HTML		Retrieves a resource identified by [URI] in HTML format
http://...	PUT	XML		…

Design

As mentioned earlier, our first step to implement REST Web Services that interact with SAS is to define which resources we are going to go after and how. For our prototype, we want to be able to retrieve libraries, tables, and Stored Processes.

We need to be able to retrieve a list of libraries as defined in the metadata repository, either in HTML or XML. Because we are concerned with connectedness, a library will have meta information such as a link to its member tables through another URI.

URI	Method	Representation	Description
…/Libraries/html	GET	HTML	Retrieves a collection of SAS libraries in HTML format
…/Libraries/xml	GET	XML	Retrieves a collection of SAS libraries in XML format
…/Libraries/objected/html	GET	HTML	Retrieves a SAS library identified by object ID in HTML format
…/Libraries/objected/xml	GET	XML	Retrieves a SAS library identified by object ID in XML format

The object ID in the URI above represents the Metadata unique identifier assigned by the SAS system so each resource can be uniquely identified.

For tables, we want to be able to add options on the URI to subset the query to a specific library; this is how a link from a library to a list of tables will be implemented. Other subsetting methods could be created such as by folder or anything else. For instance, if an application retrieved the resource at URI: …/Libraries/XGR5678/xml, it will include an XML element that is a link and has a URI attribute with a value of …/Tables;lib=XGR5678/xml.

URI	Method	Representation	Options	Description
…/Tables/html	GET	HTML	Lib=objectID	Retrieves a collection of SAS tables in HTML format
…/Tables/xml	GET	XML	Lib=objectID	Retrieves a collection of SAS tables in XML format
…/Tables/objectID/html	GET	HTML		Retrieves SAS table metadata identified by object ID in HTML format
…/Tables/objectID/xml	GET	XML		Retrieves SAS table metadata identified by object ID in XML format
…/Tables/objectID/data/html	GET	HTML		Retrieves SAS table data identified by object ID in HTML format
…/Tables/objectID/data/xml	GET	XML		Retrieves SAS table data identified by object ID in XML format

For Stored Processes, we need to be able to retrieve a list (possibly based on a folder), retrieve the input parameters (prompts) and execute the Stored Process.

URI	Method	Representation	Options	Description
…/Models/html	GET	HTML	Folder=Name	Retrieves a collection of models (Stored Processes) in HTML format
…/Models/xml	GET	XML	Folder=Name	Retrieves a collection of models (Stored Processes) in XML format
…/Models/objectID/input/html	GET	HTML		Retrieves the prompts of a Stored Process identified by object ID and display its HTML representation
…/Models/objectID/input/xml	GET	XML		Retrieves the prompts of a Stored Process identified by object ID and display its XML representation
…/Models/objectID/output	GET	HTML		Executes a Stored Process identified by object ID and streams back the results (assumed to be HTML).

The choice of providing HTML and XML representations is obviously arbitrary. We want to provide these services for zero-footprint clients, such as a simple Web browser, which will use the HTML representations. We also want to give developers a simple interface to SAS using a common standard, namely XML.

We are now ready to start developing our set of REST-based Web Services to make all the resources available.

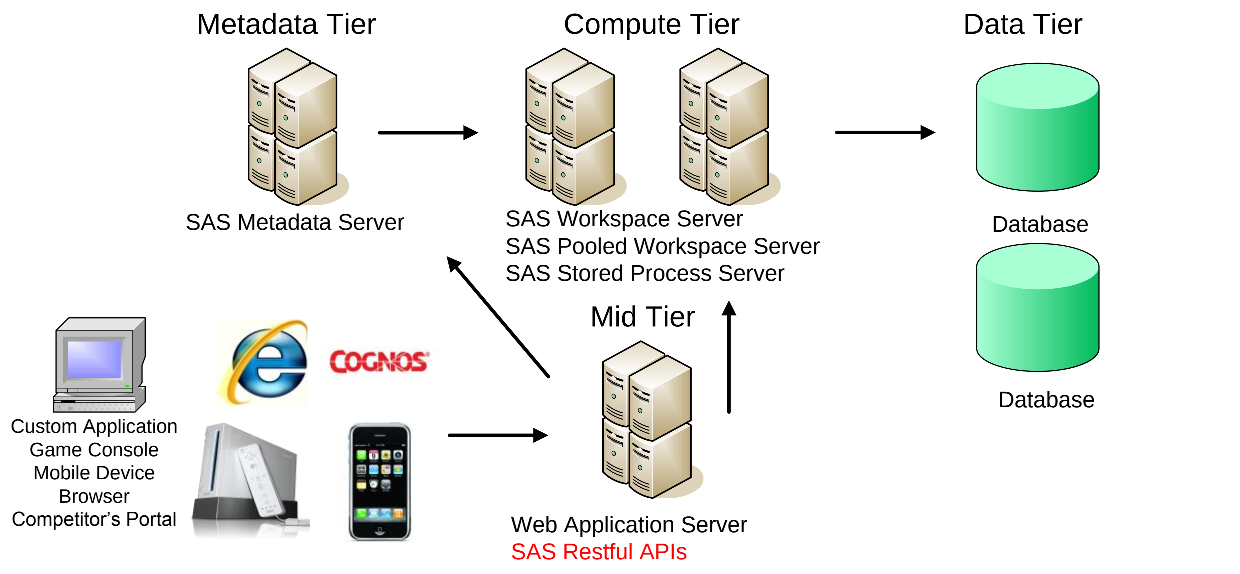
Implementation

Architecture

The application will have the following components:

- A Java class for each object defined in the SAS metadata; in our example we will have classes for libraries, tables and Stored Processes. Each class defines attributes from the SAS metadata as class variables and getter/setter methods for each.
- A Java class with methods to establish a connection to the SAS metadata: retrieves the repositories, collections of objects and their attributes.
- A Java class to execute the query when the data component of a table is requested.
- A Java class to execute a Stored Process and stream back the results to the requestor.
- XSL transformations to create HTML representations from the default XML.

These components are packaged in an application that is deployed on a Web application server. As far as our application is concerned (or from a client point of view), the client-server relation is solely between the varieties of clients and the Web application deployed as a set of REST Web services.



SAS Object Classes

Below is a trimmed down example of a class defined for SAS libraries. As mentioned it simply declares the attributes we are interested in, getter/setter methods and a constructor. The imported packages at the beginning of the code are to implicitly transform our response into XML tags.

```
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;
@XmlRootElement(name = "library")
@XmlType(propOrder = { "objectId", "name", "libref", "engine", "schema", "server", …
})
public class Library {
   private String objectId;
   private String name;
   …
   public void setObjectId(String objectId) {this.objectId = objectId;}
   public String getObjectId() {return this.objectId;}
   public void setName(String name) {this.name = name;}
   public String getName() {return this.name;}
   …
   public Library() {};
        public Library(    String objectId, String name, String libref, String engine,
…) {
        this.objectId = objectId;
        this.name = name;
        …
   }}
```

Metadata Utility

The Metadata Utility class provides a connection to the SAS Metadata Server and can retrieves the list of repositories and/or a collection of objects. Retrieving a collection of tables as in the example below simply means:

- Requesting a handle on a list of PhysicalTable objects
- Getting their attributes
- Creating a new instance of the Tables class.

```
public Boolean connectToServer() { //refer to SAS documentation for sample code }

public CMetadata getRepositories() { //refer to SAS documentation for sample code }

…

public ArrayList<Table> getTables(CMetadata repository) {
       ArrayList<Table> tables = new ArrayList<Table>();
       try {
              String reposID = repository.getFQID();
              MdObjectStore store = _factory.createObjectStore();
              int flags = MdOMIUtil.OMI_GET_METADATA | MdOMIUtil.OMI_ALL_SIMPLE;
              List tableList = _factory.getOMIUtil().getMetadataObjectsSubset
                                   (store, reposID, MetadataObjects.PHYSICALTABLE,
                                   flags, "" );
              Iterator iter = tableList.iterator();
              while( iter.hasNext()) {
                     PhysicalTable tableObject = (PhysicalTable)iter.next();
                     AssociationList alt = tableObject.getTrees();
                     Tree t = ((Tree)alt.firstElement());
                     Vector f = new Vector();
                     f.add(t.getName());
                     while(t.getParentTree() != null) {
                            t = t.getParentTree();
                            f.add(0, t.getName());
                     }
                     tables.add(new Table(tableObject.getId(), …));
              }
              store.dispose();
       }
       catch (Exception e) { e.printStackTrace();}
       return tables;
   }
```

We have created similar methods to retrieve a list of libraries and a list of Stored Processes.

Resource Classes

Each SAS object also has a resource class which functions to parse the URI and determine which resource and in what representation to fetch a response. For instance, the following code is “called” whenever the URI of the HTTP request is …/Models/xml. Note that in our application, all requests are of the GET type.

```
@Path("xml")
@GET
public StreamingOutput getModelsXML() {
       SASMetadataConnection smdc = new SASMetadataConnection();
       smdc.connectToServer();
       CMetadata repository = smdc.getRepositories();
       ArrayList<Model> models = smdc.getAllStoredProcesses(repository);
       models.addAll(modelMap.values());
       return getListXML(models); }
```

Similarly, an annotation such as below would call the code to retrieve the input (prompts) of a Stored Process identified by the parameter {id}, in an HTML representation.

```
@Path("/{id}/input/html")
@GET
@Produces("text/html")
```

XSL Transformations

Because the Java classes implicitly create an XML representation of the objects retrieved, we can re-use the same methods when an HTML representation is requested. Simply apply an XSL transformation that converts XML tags to HTML tags. The example below converts prompts defined for a SAS Stored Process (limited to text entry in this example) to the appropriate HTML tags that represent a form where users can enter parameters and execute the Stored Process.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="html"/>
<xsl:param name="objid" />
       <xsl:template match="/">
       <html>
       <form id="myForm" action="url" method='GET' target="view">
       <input type='hidden' name="sp" value="{$objid}" />
       <table>
       <xsl:for-each select="PromptGroup/DefinitionsAndSubgroups/TextDefinition">
              <tr><td><xsl:value-of select="Label/Text" /></td><td>
              <xsl:choose>
              <xsl:when test="ValueProvider/StaticValueProvider/Values">
              <xsl:choose>
              <xsl:when test="@maxValueCount>0">
              <input name="{@name}" id="{@name}" type="hidden" value="" />
              <select name="outgraphs2" id="outgraphs2" multiple="multiple" size="5">
              <xsl:for-each select="ValueProvider/StaticValueProvider/Values/String">
              <option value="{@value}" label="{@value}"></option>
              </xsl:for-each>
              </select>
              </xsl:when>
              <xsl:otherwise>
              <select name="{@name}" id="{@name}">
              <xsl:for-each select="ValueProvider/StaticValueProvider/Values/String">
                     <option value="{@value}" label="{@value}"></option>
              </xsl:for-each>
              </select>
              </xsl:otherwise>
              </xsl:choose>
              </xsl:when>
              <xsl:otherwise>
              <input type='text' name="{@name}" value="{DefaultValue/String/@value}" />
              </xsl:otherwise>
              </xsl:choose>
              </td></tr>
       </xsl:for-each>
<tr><td>Format</td><td>
<select name="_odsdest">
<option value="HTML">HTML</option>
<option value="PDF">PDF</option>
<option value="RTf">RTF</option>
</select>
</td></tr>
<tr><td>Style</td><td><input type="text" name="_odsStyle" value="" /></td></tr>
<tr><td colspan='2'><input type='button' value='Execute'
onClick="javascript:wdaExecuteModel(this.form);"/></td></tr>
</table></form>
</html>
</xsl:template>
</xsl:stylesheet>
```

Applications

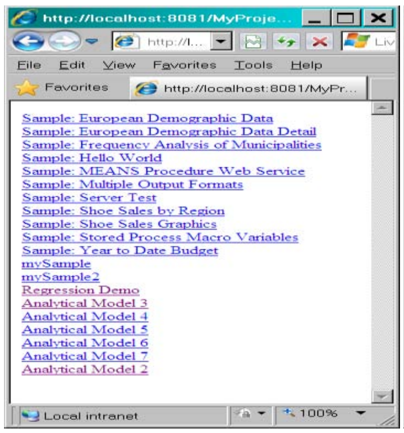
Internet Explorer/Portals

My first test was to call the Web services from a simple browser. Not having access to a competitor's portal, I wanted to demonstrate that any client that can make HTTP requests and display the resulting HTML could connect to SAS and "syndicate" its content. Most portals have the equivalent of the SAS URL Display portlet which can be used as a client of our application.

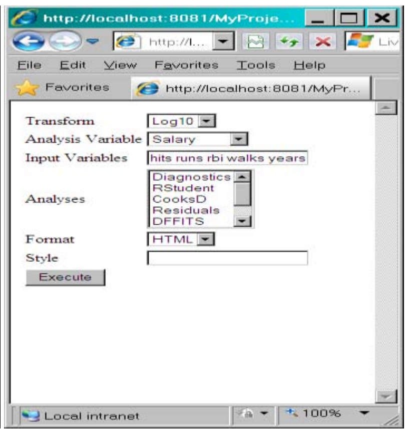
The first screenshot shows the output of pointing the browser to http://.../Models/html. The application retrieves the list of stored processes and presents them in an HTML list. Each list element is a link to the input page of the stored process, which provides the connectedness attribute of REST-based Web services.

When clicking any of the Stored Processes, the HTTP call is to http://.../Models/objectid/input/html. The application retrieves the prompts (conveniently stored as XML in the metadata repository) of the Stored Process identified as [objected]. It then applies an XSL transformation and presents the input page of the stored process (the second screenshot).

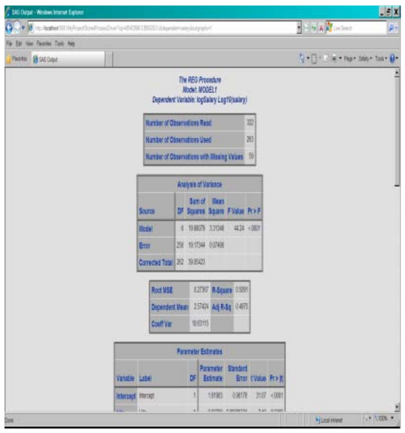
Users can then tweak the parameters and press the "Execute" button which makes the third HTTP request, http://.../Models/objectid/output. In this case the Stored Process is executed on the server and its output streamed back to the client.


http://localhost:8081/MyProje...
File Edit View Favorites Tools Help
Favorites http://localhost:8081/MyPr...
Local intranet
100%

Sample: European Demographic Data
Sample: European Demographic Data Detail
Sample: Frequency Analysis of Municipalities
Sample: Hello World
Sample: MEANS Procedure Web Service
Sample: Multiple Output Formats
Sample: Server Test
Sample: Shoe Sales by Region
Sample: Shoe Sales Graphics
Sample: Stored Process Macro Variables
Sample: Year to Date Budget
mySample
mySample2
Regression Demo
Analytical Model 3
Analytical Model 4
Analytical Model 5
Analytical Model 6
Analytical Model 7
Analytical Model 2

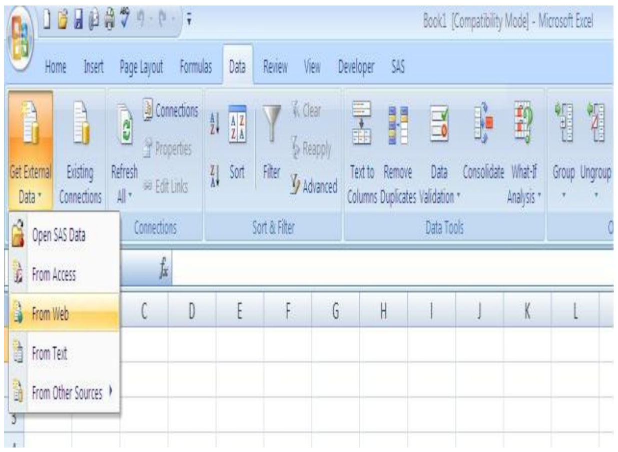

http://localhost:8081/MyProje...
File Edit View Favorites Tools Help
Favorites http://localhost:8081/MyPr...
Execute
Local intranet
100%

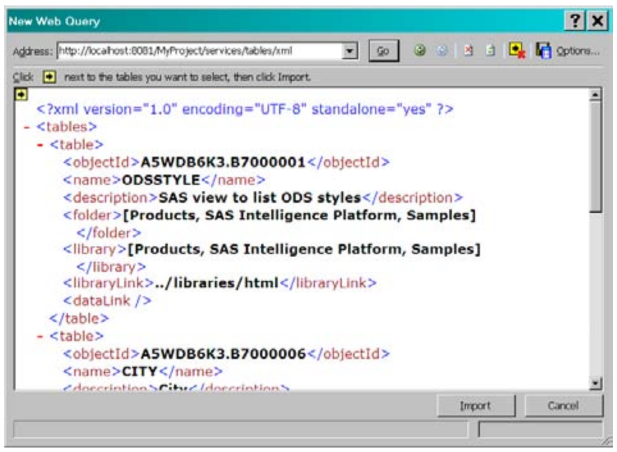
Transform	Log10
Analysis Variable	Salary
Input Variables	hits runs rbi walks years
Analyses	Diagnostics RStudent CooksD Residuals DFFITS
Format	HTML
Style	



MS-Excel

Microsoft Excel has a menu option to retrieve data over the Web. When prompted for an address, we can simply enter one of our URIs defined in the resource template. In this example, we want the list of tables in XML format. We choose XML because MS-Excel will convert the tags to a table automatically.



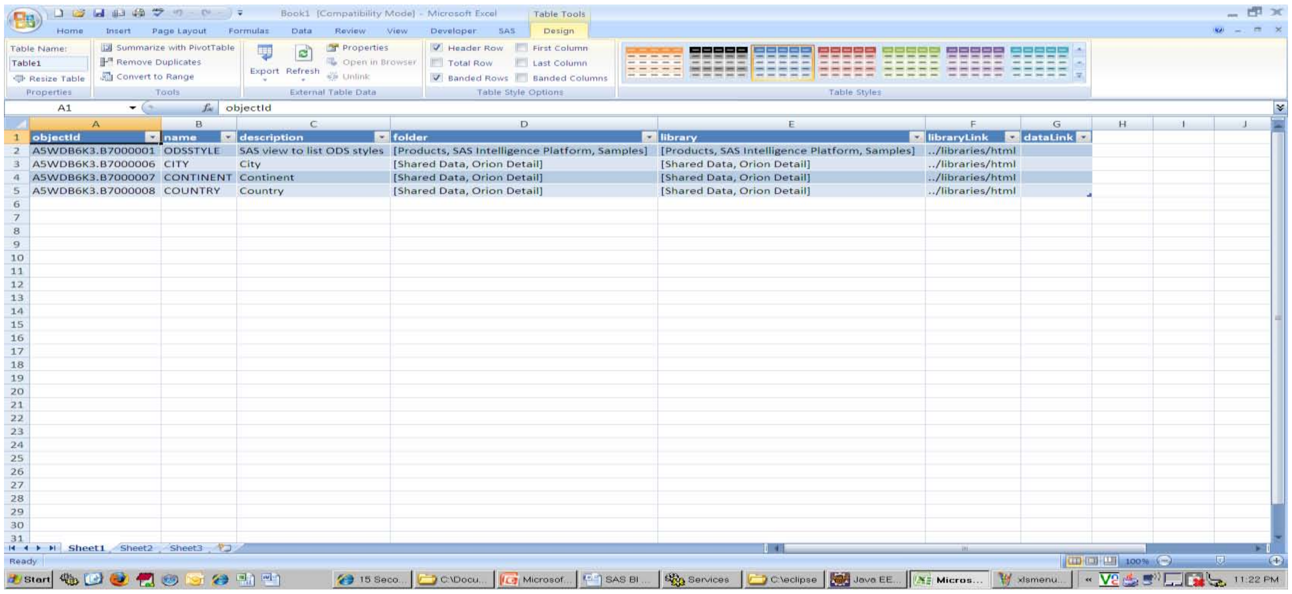


Address: http://localhost:8081/MyProject/services/tables/xml

Click next to the tables you want to select, then click Import.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <tables>
  - <table>
      <objectId>A5WDB6K3.B7000001</objectId>
      <name>ODSSTYLE</name>
      <description>SAS view to list ODS styles</description>
      <folder>[Products, SAS Intelligence Platform, Samples]
        </folder>
      <library>[Products, SAS Intelligence Platform, Samples]
        </library>
      <libraryLink>../libraries/html</libraryLink>
      <dataLink />
    </table>
  - <table>
      <objectId>A5WDB6K3.B7000006</objectId>
      <name>CITY</name>
```

This allows us to download data sets (SAS data sets or RDBMS tables or anything else defined in the metadata repository) without the need for the Add-in for MS Office which requires a SAS BI license.

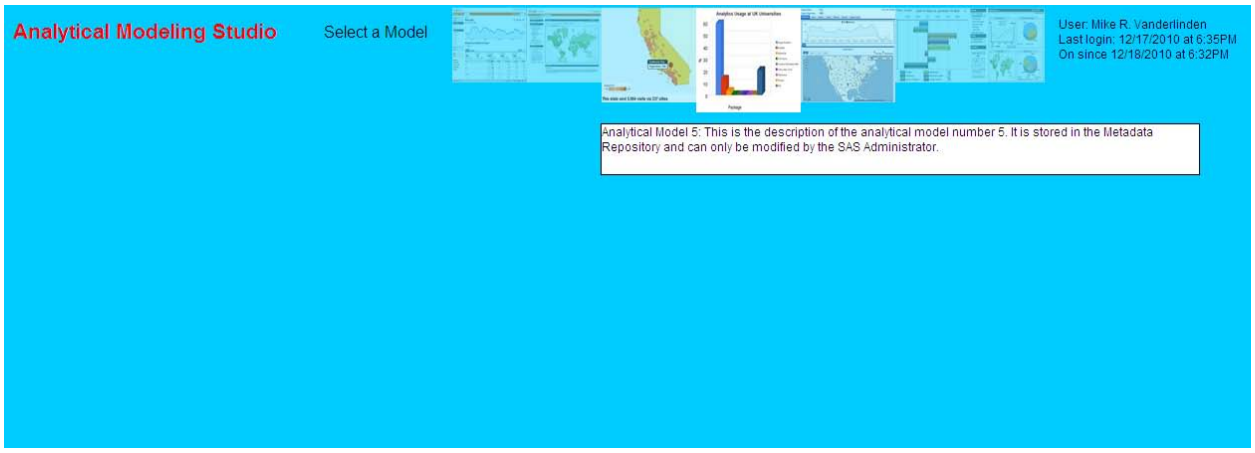


objectId	name	description	folder	library	libraryLink	dataLink
A5WDB6K3.B7000001	ODSSTYLE	SAS view to list ODS styles	[Products, SAS Intelligence Platform, Samples]	[Products, SAS Intelligence Platform, Samples]	../libraries/html	
A5WDB6K3.B7000006	CITY	City	[Shared Data, Orion Detail]	[Shared Data, Orion Detail]	../libraries/html	
A5WDB6K3.B7000007	CONTINENT	Continent	[Shared Data, Orion Detail]	[Shared Data, Orion Detail]	../libraries/html	
A5WDB6K3.B7000008	COUNTRY	Country	[Shared Data, Orion Detail]	[Shared Data, Orion Detail]	../libraries/html	

Custom Web Application

Because the application is designed to send XML or HTML as an output, application developers can request XML responses and focus their time and effort on building a rich GUI interface. With a little less than 120 lines of code (and admittedly heavy usage of variety of JavaScript frameworks), a small footprint application presents the users with a list of Stored Processes represented as icons with a looking-glass effect when “moused over”.

At the start of the application, a call is made to http:/…/Models/xml to retrieve the list of stored processes. Then a few lines of code parse the XML to transform it into a list of interactive icons as below.



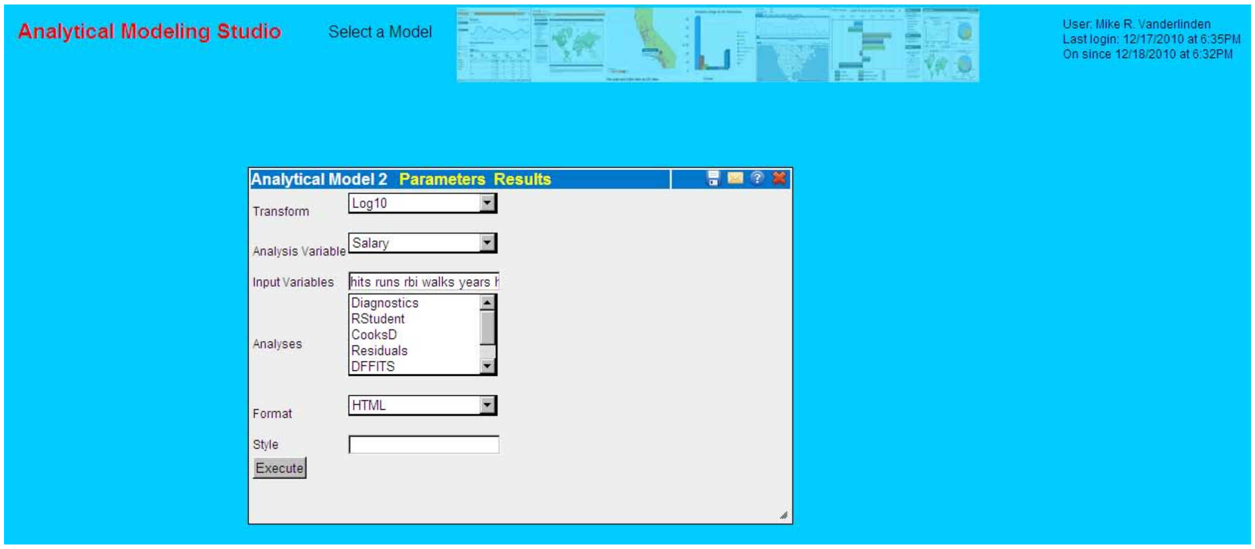
Analytical Modeling Studio

Select a Model

User: Mike R. Vanderlinden
Last login: 12/17/2010 at 6:35PM
On since 12/18/2010 at 6:32PM

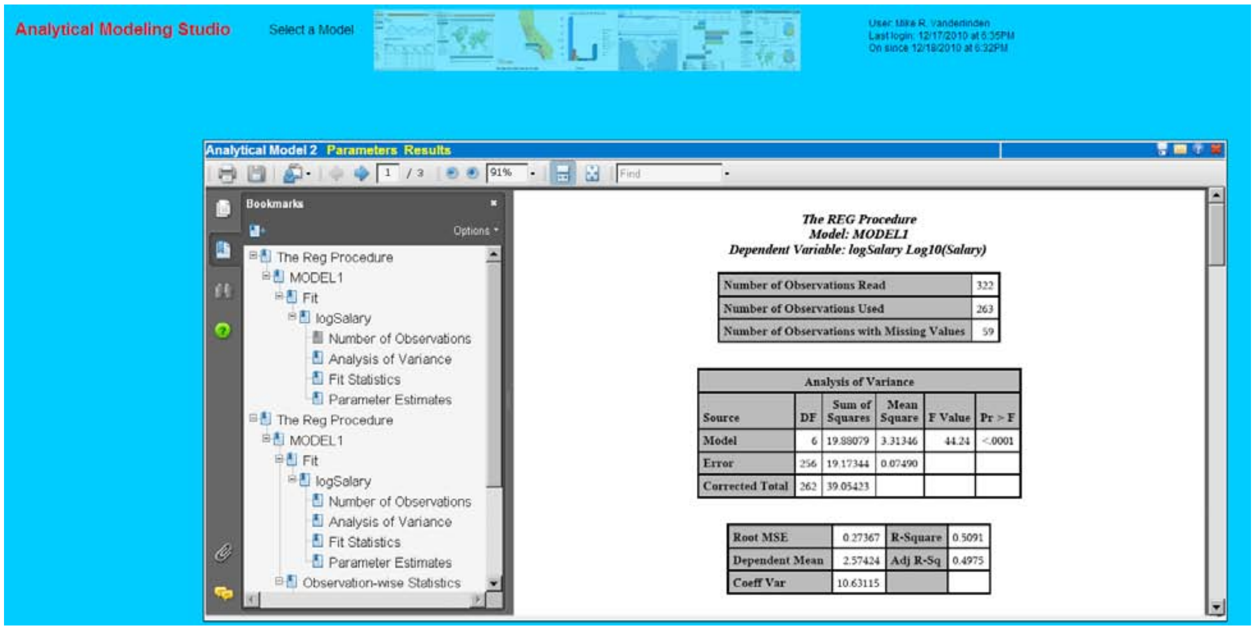
Analytical Model 5: This is the description of the analytical model number 5. It is stored in the Metadata Repository and can only be modified by the SAS Administrator.

When an icon is clicked, a second Web service call is made to http://.../Models/objectID/input/xml. The response contains the prompt definition from the metadata repository. A fully resizable and "draggable" window opens and the set of XML tags is applied an XSL transformation to turn it into an HTML form to allow users to modify the parameters of the model. The window has an input tab to present the HTML form and an output tab as a place holder for the output.


Analytical Modeling Studio
Select a Model
User: Mike R. Vanderlinden
Last login: 12/17/2010 at 6:35PM
On since 12/18/2010 at 6:32PM
Analytical Model 2 Parameters Results
Execute

Transform	Log10
Analysis Variable	Salary
Input Variables	hits runs rbi walks years h
Analyses	Diagnostics RStudent CooksD Residuals DFFITS
Format	HTML
Style	

Here the user has requested a PDF representation of the same resource which can then be saved or emailed to a recipient.



The REG Procedure
Model: MODEL1
Dependent Variable: logSalary Log10(Salary)

Number of Observations Read	322
Number of Observations Used	263
Number of Observations with Missing Values	59

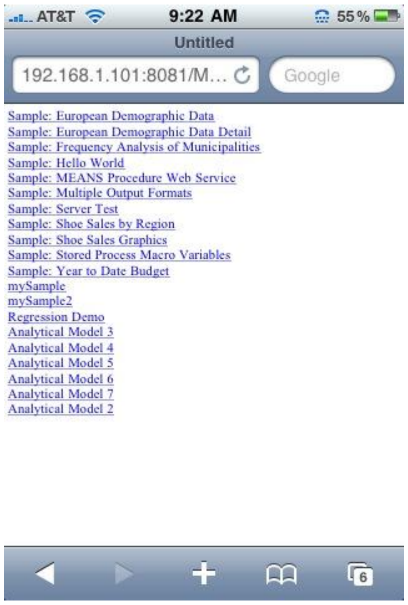
Analysis of Variance					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	6	19.88079	3.31346	44.24	<.0001
Error	256	19.17344	0.07490		
Corrected Total	262	39.05423			

Root MSE	0.27367	R-Square	0.5091
Dependent Mean	2.57424	Adj R-Sq	0.4975
Coeff Var	10.63115		

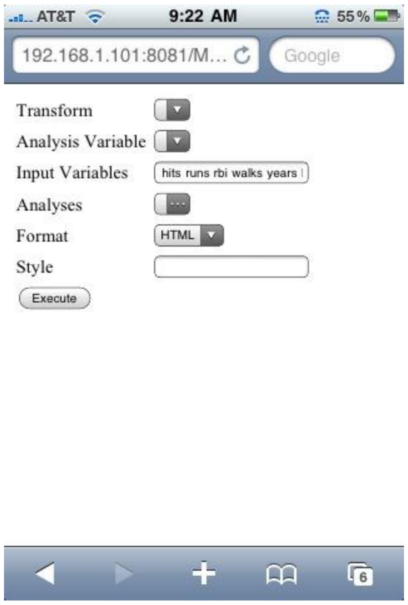
Mobile Devices

My relationship with my iPhone is a “love and hate” one. Yes I love this little gadget and most likely couldn’t live without it for more than a mere few hours, but developing an application for it entails learning yet another language, and investing in a development platform that I don’t currently own. What’s more, once I would have developed a nice app that interfaces with SAS, it wouldn’t work on any other platform/device.

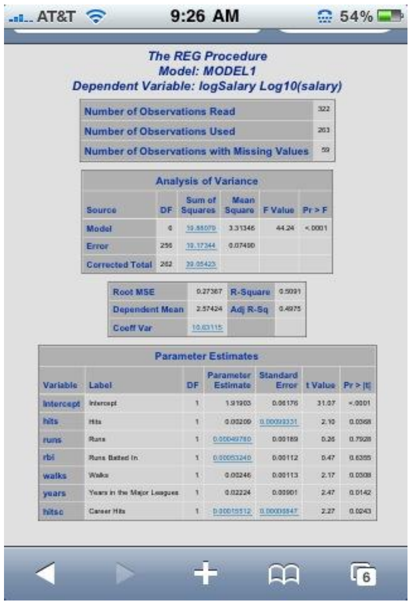
On the other hand, there is a set of RESTful Web services that returns HTML tags. This allows me to interface with SAS from any mobile device regardless of its operating system, software, etc., all I need is a Web browser.



Sample: European Demographic Data
Sample: European Demographic Data Detail
Sample: Frequency Analysis of Municipalities
Sample: Hello World
Sample: MEANS Procedure Web Service
Sample: Multiple Output Formats
Sample: Server Test
Sample: Shoe Sales by Region
Sample: Shoe Sales Graphics
Sample: Stored Process Macro Variables
Sample: Year to Date Budget
mySample
mySample2
Regression Demo
Analytical Model 3
Analytical Model 4
Analytical Model 5
Analytical Model 6
Analytical Model 7
Analytical Model 2



Transform	
Analysis Variable	
Input Variables	hits runs rbi walks years
Analyses	...
Format	HTML
Style	



The REG Procedure
Model: MODEL1
Dependent Variable: logSalary Log10(salary)

Number of Observations Read	322
Number of Observations Used	263
Number of Observations with Missing Values	59

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	6	19.88079	3.31346	44.24	<.0001
Error	256	19.17344	0.07490		
Corrected Total	262	39.05423			

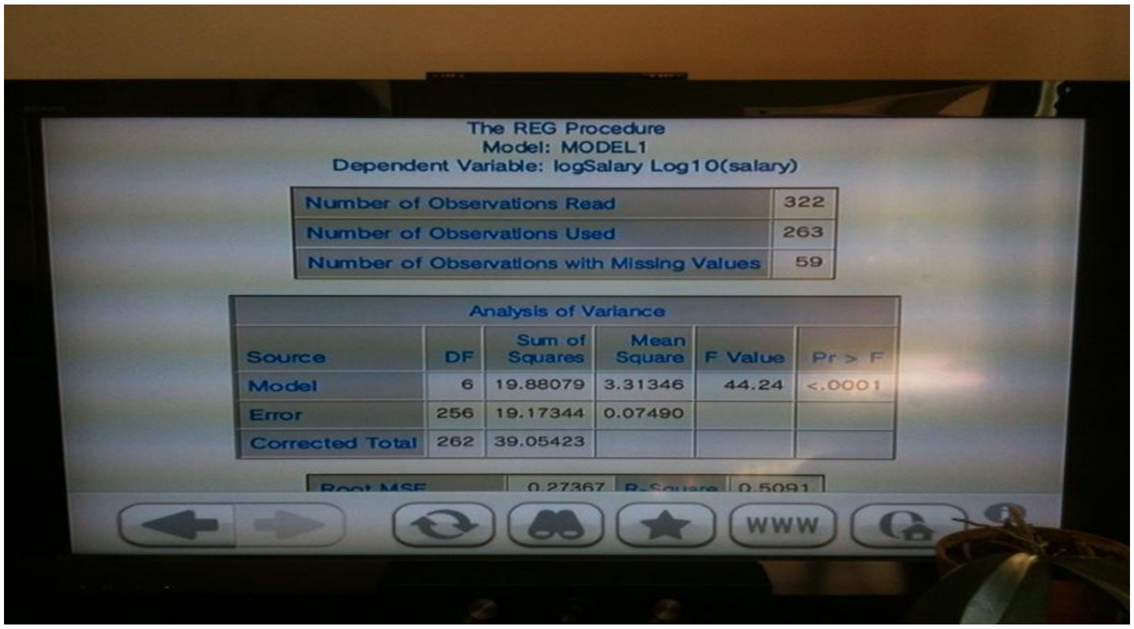
Root MSE	0.27367	R-Square	0.5091
Dependent Mean	2.57424	Adj R-Sq	0.4975
Coeff Var	10.63115		

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	1.91903	0.06176	31.07	<.0001
hits	Hits	1	0.00209	0.00099331	2.10	0.0368
runs	Runs	1	0.00049780	0.00189	0.26	0.7928
rbi	Runs Batted In	1	0.00053240	0.00112	0.47	0.6355
walks	Walks	1	0.00246	0.00113	2.17	0.0308
years	Years in the Major Leagues	1	0.02224	0.00901	2.47	0.0142
hitsc	Career Hits	1	0.00015512	0.00006847	2.27	0.0243

Game Console

Since we’re trying to talk to SAS and syndicate its content to any “HTTP enabled” device, why not a game console? I don’t expect any SAS user to start programming, developing and modeling on their Wii console, but one has to admit that a PROC REG on a 46” HDTV through a Wii console is pretty cool!!!



The REG Procedure
Model: MODEL1
Dependent Variable: logSalary Log10(salary)

Number of Observations Read	322
Number of Observations Used	263
Number of Observations with Missing Values	59

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	6	19.88079	3.31346	44.24	<.0001
Error	256	19.17344	0.07490		
Corrected Total	262	39.05423			

Root MSE	0.27367	R-Square	0.5091

Conclusion

Designing my SAS BI Content Syndication application relied on two main concepts; technical tools that allowed me to build a prototype that meets the challenge I set for myself. For one, the rich set of APIs that SAS offers to interact with its servers and particularly, the openness of its metadata architecture. Anything defined in your SAS environment can be retrieved, manipulated or removed with just a few simple API calls. Secondly, the REST architectural style provides a much simpler alternative to the complex SOAP protocol to implement Web services. It is because of its simplicity and the constraint of a uniform interface to interact with the Web services that the SAS BI content can be syndicated from any device that can communicate with them, in my case any device that implements the HTTP protocol.

References

Architectural Styles and the Design of Network-based Software Architectures, University of California, Irvine, 2000 by Roy Thomas Fielding, Ph.D.

Recommended Reading

SAS 9.2 Integration Technologies: Java Client Developer's Guide, SAS Documentation
SAS 9.2 Foundation Services: Administrator's Guide, SAS Documentation
SAS AppDev Studio 3.4 Eclipse Plug-ins: User's Guide, SAS Documentation
RESTful Java with Jax-RS, O'Reilly, Bill Burke,
RESTful Web Services Cookbook, O'Reilly, Subbu Allamaraju

Contact Information

Mike Vanderlinden
Business Analytics, Experis
5220 Lovers Lane, Suite 200
Portage, MI 49002
(734) 756-1083
Mike.vanderlinden@experis.com
http://www.experis.com