Paper SAS606-2017

Power to the People! Web Service Scoring for the Masses

Chris Upton and Prasenjit Sen, SAS Institute Inc.

ABSTRACT

SAS® Decision Manager includes a hidden gem: a web service for high speed online scoring of business events. SAS® 9.4 M4 represents the third release of the SAS® Micro Analytic Service for scoring SAS® DS2 code decisions in a standard JSON web service. Users will learn how to create decisions, deploy modules to the web service, test the service, and record business events.

INTRODUCTION

A Web Service is a software system that provides a service, delivered with internetworking technologies. The internet is a distributed system built with diverse components that run various software applications on different software platforms and hardware. Web services leverage the internet to provide a standard means for interaction between software applications.

Service consumers, software applications that intend to use the service, are provided with a description of the interface of the Web Service. These applications interact with the Web Service, strictly in the manner described by the interface, in order to utilize the functionality provided.

Originally defined by the World Wide Web Consortium (W3C) consortium to only mean XML and SOAP based technologies, the definition for web services has been extended to include RESTful web services that may use JSON as the standard to describe messages between service providers and consumers.

Many incentives exist for leveraging the internet for service delivery. They include:

- Large scale interoperability between diverse components without requiring every component to know about each other.
- Resiliency of the system to continue to function even when parts of it goes down.
- Ability to support evolution of service providers over time independent of service consumers.

Web services also present an on-demand model of service delivery, where service is provided independently for every request without requiring the service consumer to wait for other requests to be completed. In fact the service is provided concurrently with the service of other requests, with minimal wait time or latency.

Scoring is an application that is well suited for delivery as a web service.

Most scoring logic is entity specific, i.e. there is a score for every customer which normally does not affect the score for another.

Scoring logic is typically developed and refined in an analytical environment, but finally deployed in a business process where it is scaled up to meet the demands of business.

Scoring logic often evolves over time based on new data, but is expected to provide the same interface to the business without major reengineering or disruption of service. Conversely, the scoring logic may also remain stable while new lines of business processes are added to utilize it.

This paper how SAS® Micro Analytic Service executes scoring logic as a RESTful web service for integration into a business process.

TECHNICAL DETAILS

WHAT IS SAS® MICRO ANALYTIC SERVICE

SAS® Micro Analytic Service is a RESTful web service, provided as a component of the SAS® Decision Management suite of products. It is a high performance, low latency, execution service for program logic like score code. Program logic is compiled and held in memory ready for execution in response to requests. This results in low latency on demand execution of complex scoring code.

WEB SERVICE ARCHITECTURE

While there is considerable flexibility in the implementation of a web service provider, a great deal of standardization exists in the interface exposed by the provider. This standardization is extremely important to ensure interoperability with service consumers.

RESTful Service

Representational state transfer (REST) is a software architecture style that models the software application as the manipulation of distributed, structured data called Resources.

Resources are addressable by URI (Uniform Resource Identifier) and presented to consuming application as representations. The state of the application is defined by the consumer of the resources and not the server. A limited but standardized set of verbs are used to manipulate the resources. The set of returned values from operations are also limited and standardized.

REST was defined by Roy Fielding in his 2000 PhD dissertation "Architectural Styles and the Design of Network-based Software Architectures".

A RESTful web service is one that uses REST architecture style to implement web services.

SAS® Micro Analytic Service is implemented as a RESTful web service and stands to benefit from all that the architecture provides.

In addition to the above standardization, the message structure for SAS® Micro Analytic Service is defined using JSON, which is short form of JavaScript Object Notation.

Concurrent Execution

SAS® Micro Analytic Service is implemented as a multithreaded service, which allows concurrent execution of the program logic. Multithreading is important to leverage modern multi-processor hardware that are common today. More importantly, the service is suited for online use by a web application or a call center where the inbound requests drive the system.

High Availability and Scalability

SAS® Micro Analytic Service is implemented as a web application server cluster containing multiple nodes, so that if a particular node fails, the service consumer can fail over to the other nodes.

Additionally, more nodes may be added to the cluster to handle larger workload, without changes to the service consumer code.

Security

SAS® Micro Analytic Service supports token based security in addition to transport layer security to meet data center security standards. More information about using token based security with SAS® Micro Analytic Service Web Service is available in the paper titled “REST at Ease with SAS®: How to Use SAS to Get Your REST”, referenced below.

Calling RESTful Web Services

RESTful Web Services may be called by using any software that is able to make HTTP calls. Web browsers often do not support the full gamut of HTTP verbs and are not as capable. Command-line

utilities like cURL (https://curl.haxx.se/) are open source and popular. It is also possible for client applications to include the HTTP client code to present an integrated feel.

PROGRAMMING MODEL

The program logic in SAS® Micro Analytic Service is organized as a language independent 2 level hierarchy.

A ***step*** is a unit of program logic that may execute when provided input data and returns results.

The data accepted and returned by a step are of specific types like integer, strings and decimal.

The type of data that is accepted and returned by a step is defined by the ***signature*** of the step.

A ***module*** is a collection of steps that is deployed as a unit. Modules may be complex with many steps contained within them or could only contain one step within the entire module.

Modules may be tagged as ***private*** or ***public***. Only steps of public modules are available for execution through the SAS® Micro Analytic Service REST API.

It is also possible for private modules to be swapped out for more refined implementations, while the modules that call them may remain the same.

Scoring code

As mentioned above, program logic for scoring has aspects that focus on analytical models, business rules and decision logic. The programs that implement these aspects are developed and evolve differently. Analytical models are refined with the collection of new data. Business Rules may change due to changes in business. Creating separate private and public modules for each part is the recommended approach to separate these concerns and allow updates to individual parts of score code.

All scoring code created by the SAS® Decision Manager contains multiple modules that include a public decision logic module and one or more private modules for business rules and models. SAS® Decision Manager also includes the client code to directly deploy scoring code into SAS® Micro Analytic Service Web Service.

DS2 Programming Model

DS2 is a SAS® proprietary programming language that is appropriate for advanced data manipulation. SAS® Micro Analytic Service supports DS2 as the implementation for the program logic that runs in it. The DS2 language is described in detail in the SAS® 9.4 DS2 Language Reference.

This section outlines how SAS® Micro Analytic Service constructs map to the DS2 language constructs. Not all DS2 language constructs are supported for use in SAS® Micro Analytic Service.

Modules in SAS® Micro Analytic Service correspond to ***packages*** in the DS2 language, while steps correspond to package **methods**.

The ***parameters*** of the package method define the signature of the step. Method parameters that are designated as `in_out` are used to return values.

To be used in SAS® Micro Analytic Service, the DS2 code must be organized into packages with package methods. In addition, the methods that are intended to be called by the consumers of the RESTful Web Service may only contain variables supported by the JSON interface, which includes integer, decimal and string. DS2 package methods support return values. However, to be called through the REST interface, all return values must be returned using `in_out` parameters and not as the return value of the method.

DS2 packages deployed in SAS® Micro Analytic Service may contain other methods that return values or use parameter types that are not supported by the JSON interface. These could be used to compose more complex programming logic. However these methods are not exposed as steps through the interface of the RESTful Web Service, and thus may not be accessed by the consumers of the REST interface.

Example of DS2 module

While scoring code is usually more complex and involves multiple modules, the following is a simple example of DS2 code that is suitable for hosting in SAS® Micro Analytic Service:

```
ds2_options sas;
package scalars_test /overwrite=yes;
   method test_all_types(
      varchar(32767) in_string,
      char           in_char,
      bigint         in_bigint,
      int            in_int,
      double         in_double,
      in_out varchar out_string,
      in_out char    out_char,
      in_out bigint  out_bigint,
      in_out int     out_int,
      in_out double  out_double);

   /* Simple echo test */
   dcl int i;
   dcl varchar(256) test_string;
   test_string='/*SOME TEXT*/';
   out_string=in_string;
   out_char=in_char;
   out_bigint=in_bigint;
   out_int=in_int;
   out_double=in_double;
end;
endpackage;
```

SAS® MICRO ANALYTIC SERVICE REST API

The capabilities of the SAS® Micro Analytic Service are exposed through the REST API. A description of the REST API is available in the SAS® MICRO ANALYTIC SERVICE 2.2: Programming and Administration Guide pp 61.

The REST API is described in terms of the following 3 things:

1. Resources and resource collections that the API exposes and their URIs.
2. HTTP verbs that may be applied to each collection or resource.
3. The JSON representation that is used to manipulate the resources and collections.

Both ***modules*** and ***steps*** appear as resources in the SAS® Micro Analytic Service API.

The modules collection is a top level collection containing all modules deployed in the system. A module is a child resource of this collection. The steps collection is a child resource of a module and a step is a child of the steps collection.

Scoring consumers interact with this API by invoking standard HTTP verbs like GET and POST. The API accepts and returns representations of the resources and collections as JSON documents. This is often done by some agent software on behalf of the user. This article focuses on the use of SAS Decision Manager and the curl utility to submit requests to the REST service.

Deploying DS2 code to SAS® Micro Analytic Service

The first step in making program logic available for execution in SAS® Micro Analytic Service is to deploy the module using the REST web service.

Modules are deployed into the SAS® Micro Analytic Service by sending a JSON representation of the module, known as the module definition, to the REST API using the POST method on the modules collection. The URI of the modules collection looks like as follows:

```
http://<host>:<port>/SASMicroAnalyticService/rest/modules
```

A command-line utility like cURL may be used to POST data to the above URI. The following is an example that uses the cURL utility to POST data contained in the file.

```
$ curl –F modulefile \
http://<host>:<port>/SASMicroAnalyticService/rest/modules
```

For clarity, parameters that deal with the security credentials are not shown.

The modulefile referenced in the example contains the JSON content for the module definition and is used to create the module.

An example of the JSON content that creates a module is as follows:

```
{
   "version": "1",
   "description": "A description of the module",
   "code":  "package varTest /overwrite=yes;\n declare package hash h1();
method execute(varchar(32767) in_string, bigint in_int, double in_float,
bigint in_boolean, in_out varchar out_string, in_out bigint out_int, in_out
double out_float, in_out bigint out_boolean);\n out_int=in_int + 2;\n
out_float=in_float + 1.11;\n if (in_boolean = 0) then out_boolean=1;\n else
out_boolean=0;\n out_string=reverse(in_string);\n end;\n endpackage;\n
run;\n",
   "scope": "public",
   "type": "text/application.source.ds2",
   "properties": [ {"name" : "connectionString", "value" : "connection
information"} ]
}
```

The code section contains the actual DS2 code suitably formatted for inclusion in the JSON document.

The DS2 code in the example above creates a module called `varTest`, with a single step called `execute`.

During deployment, the program code for the module is compiled and checked for errors. If there are no errors, the compiled code is held in memory ready for execution.

The API will return an HTTP status code of 201 that denotes that the module has been created. The API will also return a representation of the module that contains a summary description of the module, including the module id, the name of the module, the description string used to create the module, the scope of the module, creation and modification timestamps, and other metadata about the module.

Since module names are unique, if another module with the same name exists, then this request to create a new module would be rejected.

In the current version of SAS® Micro Analytic Service, the module id is the same as the name of the module which is the same as the name of the DS2 package. The description of the module and other

attributes are sourced from the module definition, while the timestamps reflect the actual time when the module is created.

After the module is created, a call to list all the modules in the system will show the module in the modules collection.

The steps available for the module may be viewed by using the HTTP verb GET on the steps collection of the module using the following URI:

```
http://<host>:<port>/SASMicroAnalyticService/rest/modules/varTest/steps
```

This will return a JSON representation of the steps collection which is expected to contain a single step called `execute`.

If your program logic is contained within a model and rule flow registered within SAS® Decision Manager, you can use a user interface to publish DS2 code for modules at the push of a button, instead of using HTTP POST requests as described above.

See Figure 1 below for an example of the publish interface of SAS® Decision Manager.

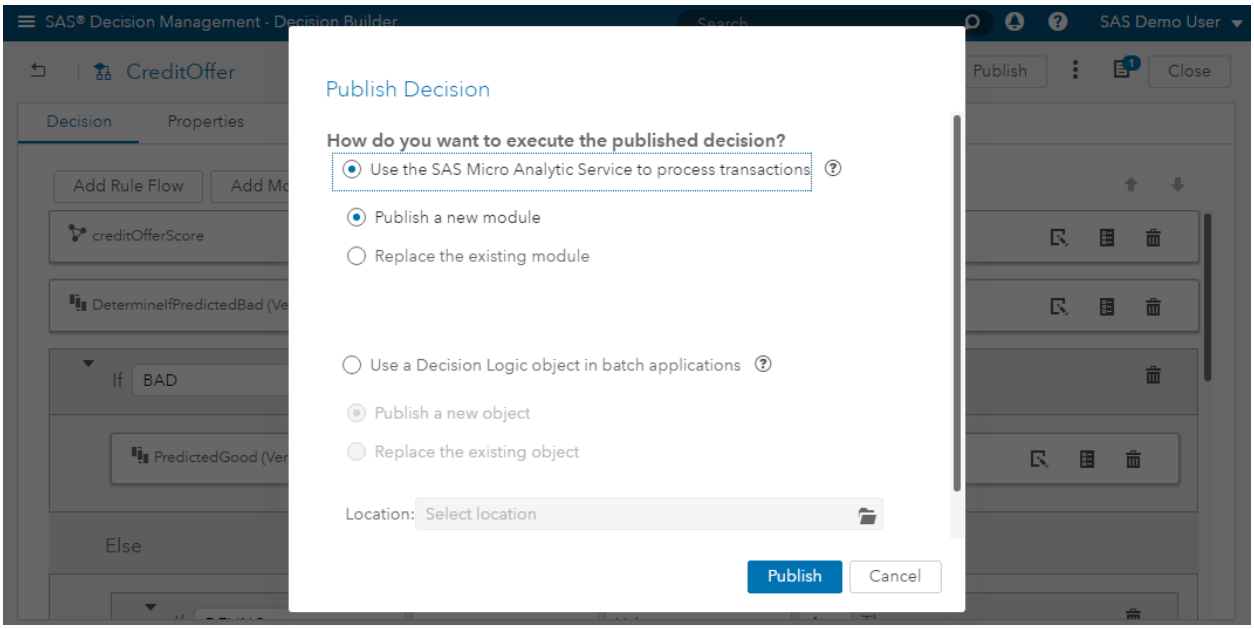

Figure 1. SAS® Decision Manager publish interface.

Executing STEPS in SAS® Micro Analytic Service

After a public module containing the DS2 code is successfully deployed without errors, the REST API is ready to accept requests to execute the code. Program logic is executed by sending a JSON message representing the input to the step using the POST http method on the REST API.

Continuing with the above example, the URI to where this message is POSTed is:

```
http://<host>:<port>/SASMicroAnalyticService/rest/modules/varTest/steps/execute
```

A command-line utility like cURL may be used to POST data to the above URI. The following is an example that uses the cURL utility to POST data contained in the file.

```
$ curl -F inputfile \
http://<host>:<port>/SASMicroAnalyticService/rest/modules/varTest/steps/execute
```

For clarity, parameters that deal with the security credentials are not shown.

The following is an example of input data to execute the step `execute` of module `varTest`:

```
{
   "inputs": [
     {
       "name": "in_string",
       "value": "the string"
     },
     {
       "name": "in_int",
       "value": 2
     },
     {
       "name": "in_float",
       "value": 2.2
     },
     {
       "name": "in_boolean",
       "value": 1
     }
   ]
}
```

If the module is deployed with private visibility or not deployed at all, the client will receive the HTTP status code of 404 Not Found. It will also return 404 Not Found if the module is deployed, but does not contain the step, or if the package method is not suitable for use in the REST interface.

In case the data contained in the file is not formatted correctly, or contains incorrect value for the parameters (e.g. supplying a string value when an integer is expected), the HTTP status code of 400 Bad Request is returned.

Since both the module and step exists in SAS® Micro Analytic Service, the REST API would run the DS2 package method called `execute` and return the HTTP status code of 200 OK.

Here is an example of the output:

```
{
   "moduleId":"varTest",
   "moduleName":"varTest",
   "stepId":"execute",
   "links":[{
       "method":"POST",
       "rel":"self",
       "href":
"http://host:port/SASMicroAnalyticService/rest/modules/varTest/steps/execute"
,
       "uri":"/modules/varTest/steps/execute",
       "type":"application/vnd.sas.microanalytic.module.step.output"
     },
     {
```

```
          "method":"GET",
          "rel":"up",
          "href":
"http://host:port/SASMicroAnalyticService/rest/modules/varTest/steps",
          "uri":"/modules/varTest/steps",
          "type":"application/vnd.sas.collection"
        },
        {
          "method":"GET",
          "rel":"step",
          "href":
"http://host:port/SASMicroAnalyticService/rest/modules/varTest
/steps/execute",
          "uri":"/modules/varTest/steps/execute",
          "type":"application/vnd.sas.microanalytic.module.step"
         }],
    "version": 1,
    "outputs": [
      {
        "name": "out_string",
        "value": " gnirts eht"
      },
      {
        "name": "out_int",
        "value": 4
      },
      {
        "name": "out_float",
        "value": 3.31
      },
      {
        "name": "out_boolean",
        "value": 0
      }
    ]
}
```


SAS DECISION MANAGER GENERATED MODULES

The modules generated from SAS® Decision Manager have a specific structure and understanding this important for SAS® Micro Analytic Service users as it is the primary creator of modules within the SAS® platform and will help when users desire to directly POST modules to SAS® Micro Analytic Service in avoiding naming collisions.

Scoring logic in SAS® Decision Manager is published to SAS® Micro Analytic Service when contained within a decision. Decisions are powerful constructs that allow you to encapsulate a simple model for scoring or orchestrate between a combination of conditioned business rules and models.

See Figure 2 below which shows a decision to determine how to offer credit to a customer applying for a loan.

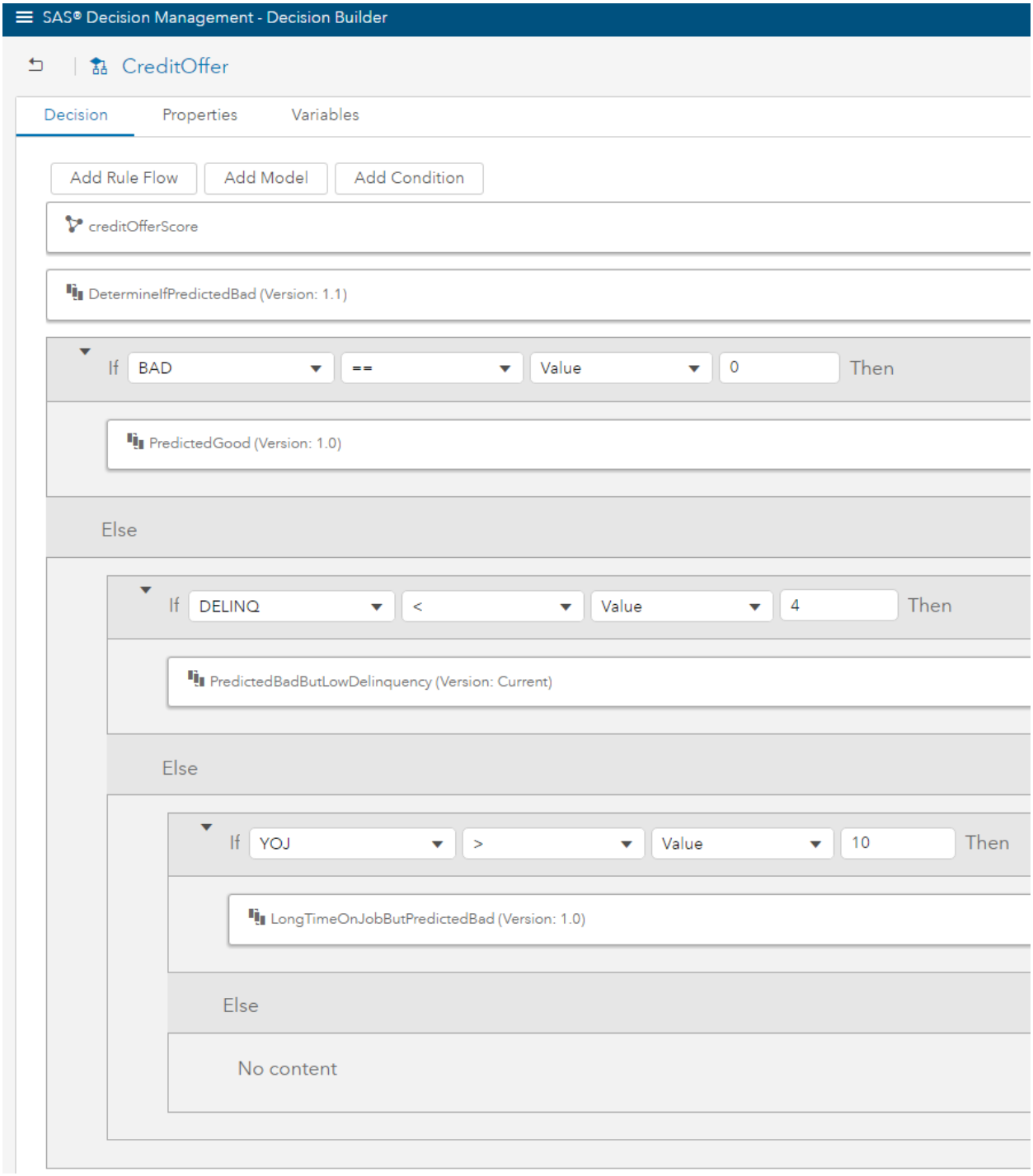

Figure 2. Example Credit Decision

It contains one model and 4 rule flows. Three of the rule flows are conditionally executed.

When this is published to the SAS® Micro Analytic Service there are 6 modules created. 5 private modules to encapsulate the 4 rule flows and one model and one public for the overall decision which is meant to be the execution target. The names of the private modules are encoded to create unique names to ensure that the decision's content does not affect other published decisions.

The public module though has a name which is representative of the decision name to ensure clear identification when looking at the module via the publish module in REST.

SAS® Decision Manager provides options when redeploying to overwrite the previously published modules or generate all new modules with a version suffix which is the only case where the module names will vary from the standard naming above.

Knowing this a user can quickly identify the public decision module when requesting the list of modules from SAS® Micro Analytic Service using the filter parameter of 'name'. e.g. `http://host:port/SASMicroAnalyticService/rest/modules?name=decision_name`, and then understand the dependent private module names by examining the code and looking at the declared package instances. See the example below demonstrates the dependent module package declarations:

```
package creditoffer / overwrite=yes;
     dcl package b_EHLCKEJHM5D3XBXQWNMKJH326I ruleflow114();
     dcl package m_SI2TF5EIYNGNHJAP5CA3MOFU64 model110();
     dcl package b_AB2X3IMUYNFTZH32FGXDHSWHFM ruleflow111();
     dcl package b_3KRCTXACCVEXNPD6RGS2LW5GBA ruleflow112();
     dcl package b_YJ42RFPXMFHK3FTJ6DYPHQ43CU ruleflow113();
… more package content …
```

Within each public decision module the step or method which is the scoring method is called 'execute', this is the target step that integrators should execute to trigger the decisions scoring logic. See the example execute method below which matches the code generated by the decision in Figure 2:

```
package creditoffer / overwrite=yes;

… package declarations …

method execute (
          double "DEBTINC_",
          double "DELINQ_",
          varchar(100) "JOB",
          double "MORTDUE_",
          double "VALUE_",
          double "YOJ",
          in_out double "BAD",
          in_out double "DEBTINC",
          in_out double "DELINQ",
          in_out double "MORTDUE",
          in_out double "Offer",
          in_out double "OfferAmount",
          in_out double "P_BAD",
          in_out double "VALUE");
    … scoring logic …
End;
… more package content …
```

USING SAS® MICRO ANALYTIC SERVICE FOR SCORING

As seen in the previous section, real-life score code typically spans multiple modules. In this section we will use the example of a multi-module scoring code that has been deployed to a SAS® Micro Analytic Service using SAS® Decision Manager or some other REST client.

Even though the real life use cases require the deployment of multiple modules, the task for the client to utilize the service is not much different. All the client needs to do is to execute the appropriate step of the public module which will in turn call other modules as needed.

To run the execute step of the `creditoffer` module the following cURL command may be run as follows after substituting appropriate values for host and port.

```
$ curl –F inputfile \
http://host:port/SASMicroAnalyticService/rest/modules/creditoffer/steps/execute
```

The following is an example of the contents of the `inputfile`:

```
{
   "inputs": [
     {
       "name": "DEBTINC_",
       "value": 200.00
     },
     {
       "name": "DELINQ_",
       "value": 2.0
     },
     {
       "name": "JOB",
       "value": "Senior Manager"
     },
     {
       "name": "MORTDUE_",
       "value": 10000.0
     },
     {
       "name": "VALUE_",
       "value": 100.0
     },
     {
       "name": "YOJ",
       "value": 1.0
     }
   ]
}
```

Submitting the above will execute the step `execute` of the module whose module id is `creditoffer` assuming the module named `creditoffer` has been deployed in the system.

The following is an example of the output received after successful execution:

```
{
  "moduleId":"creditoffer",
  "moduleName":"creditoffer",
  "stepId":"execute",
```

```
     "links":[{
         "method":"POST",
         "rel":"self",
         "href":
"http://host:port/SASMicroAnalyticService/rest/modules/creditoffer/steps/exec
ute",
         "uri":"/modules/creditoffer/steps/execute",
         "type":"application/vnd.sas.microanalytic.module.step.output"
       },
       {
         "method":"GET",
         "rel":"up",
         "href":
"http://host:port/SASMicroAnalyticService/rest/modules/creditoffer/steps",
         "uri":"/modules/creditoffer/steps",
         "type":"application/vnd.sas.collection"
       },
       {
         "method":"GET",
         "rel":"step",
         "href":
"http://host:port/SASMicroAnalyticService/rest/modules/creditoffer/steps/exec
ute",
         "uri":"/modules/creditoffer/steps/execute",
         "type":"application/vnd.sas.microanalytic.module.step"
       }],
     "version":1,
     "output":[
        {"name":"\"BAD\"","value":0.0},
        {"name":"\"DEBTINC\"","value":200.0},
        {"name":"\"DELINQ\"","value":2.0},
        {"name":"\"MORTDUE\"","value":10000.0},
        {"name":"\"Offer\"","value":1.0},
        {"name":"\"OfferAmount\"","value":10000.0},
        {"name":"\"P_BAD\"","value":0.96491228070175},
        {"name":"\"VALUE\"","value":100.0}]
   }
```

CONCLUSION

SAS® Micro Analytic Service is a RESTful Web Service that provides a high performance vehicle to execute custom scoring logic or published decisions from SAS® Decision Manager in the context of a business transaction. The RESTful interface enables ease of integration with external production systems that can benefit with the delivery of the scoring logic to the business process.

REFERENCES

Fielding, Roy Thomas (2000). "Architectural Styles and the Design of Network-based Software Architectures". Dissertation. University of California, Irvine. Available at https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

W3C Working Group Note 11 February 2004. “Web Services Architecture” . Accessed March 1, 2017. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211.

Hypertext Transfer Protocol (HTTP) Status Code Registry. Accessed March 1, 2017. https://tools.ietf.org/html/rfc2616.

SAS6363-2016 - Joseph Henry, SAS Institute Inc. “REST at Ease with SAS®: How to Use SAS to Get Your REST.” Available at http://support.sas.com/resources/papers/proceedings16/SAS6363-2016.pdf

ACKNOWLEDGMENTS

The authors of this paper would like to thank the members of the various teams in R&D, Product Management, Marketing, and of course, our customers.

RECOMMENDED READING

- *SAS® Micro Analytic Service 2.2: Programming and Administration Guide. Available at http://support.sas.com/documentation/cdl/en/masag/69596/PDF/default/masag.pdf*
- *SAS® 9.4 DS2 Language Reference, Sixth Edition. Available at http://supportprod.unx.sas.com/documentation/cdl/en/ds2ref/69739/PDF/default/ds2ref.pdf*
- *SAS® Decision Manager 2.2 Administrator’s Guide. Available at http://support.sas.com/documentation/cdl/en/edmag/67011/PDF/default/edmag.pdf*

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Prasenjit Sen
SAS Institute
919-531-0279
prasenjit.sen@sas.com

Chris Upton
SAS Institute
919-531-2316
chris.upton@sas.com