SAS® Micro Analytic Service 1.2

Programming and Administration Guide

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2015. *SAS® Micro Analytic Service 1.2: Programming and Administration Guide*. Cary, NC: SAS Institute Inc.

SAS® Micro Analytic Service 1.2: Programming and Administration Guide

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513-2414.

July 2015

Contents

About This Book

Audience

This guide is intended for developers and information technology administrators. Developers can use the information to author SAS DS2 code that extends or customizes the functionality of SAS Enterprise Decision Manager, SAS Business Rules Manager, or SAS Real-Time Decision Manager to meet their business needs. Developers can find information about the SAS Micro Analytic Service runtime environment, as well as tips, best practices, and restrictions on programming DS2 to run in SAS Micro Analytic Service. Information technology administrators can find information about SAS Micro Analytic Service deployment, operation, and tuning.

Accessibility

For information about the accessibility of any of the products mentioned in this document, see the usage documentation for that product.

Chapter 1

Introduction to SAS Micro Analytic Service

What Is SAS Micro Analytic Service?

SAS Micro Analytic Service 1.2 is a memory-resident, high-performance program execution service. As a SAS platform service, it is not available for individual license, but is included in selected SAS solutions. SAS Micro Analytic Service 1.2 provides hosting for DS2 programs and supports a "compile-once, execute-many-times" usage pattern. SAS Micro Analytic Service is multi-threaded and can be clustered for high availability. It can host multiple programs simultaneously, as well as multiple user or business contexts that are isolated from one another.

SAS Micro Analytic Service has a layered architecture that is suitable for a variety of deployment topologies. The core engine is written in C for high performance. Java clients can integrate with its Java Plain Old Java Object (POJO) interface. The SAS Micro Analytic Service POJO interface communicates with the C engine through an optimized JNI layer. A web application with a REST interface is provided for integration with SAS solutions and other client applications, and adds persistence and clustering for scalability and high availability.

For example, SAS Enterprise Decision Manager generates DS2 programs that implement user-created rule sets and rule flows. It can combine SAS analytics, such as score code generated by SAS Enterprise Miner, with business rules in order to form decision logic. SAS Micro Analytic Service is used to compile and execute the generated code.

In addition to providing generated code, SAS Micro Analytic Service enables users of solutions such as SAS Real-Time Decision Manager and SAS Enterprise Decision Manager to author DS2 code that is customized to their specific needs. SAS Micro Analytic Service supports a subset of the DS2 programming language, which includes language features that are suitable for the high-performance execution of transactions. Specific rules and restrictions are detailed in Chapter 3, "DS2 Programming for SAS Micro Analytic Service," on page 11.

Chapter 2

Concepts

Overview

SAS Micro Analytic Service has several component types, which are arranged in a three-level hierarchy.

Note: If you are writing code to deploy to SAS Micro Analytic Service, follow the programming guidelines described in Chapter 3, "DS2 Programming for SAS Micro Analytic Service," on page 11.

1. User or business context
2. Module context
3. Revision

User or Business Context

A *context* is a container for the programs that SAS Micro Analytic Service executes. It is also an isolated execution environment. That is, programs executing in one context are not visible to any other context. Therefore, contexts can be used to provide a separate environment for each user or different business unit, or for any other usage requiring

isolation. The programs hosted by SAS Micro Analytic Service are known as *modules*. A context is a container of modules.

Because business context and user context are interchangeable terms that describe the two common uses of this single component, this document uses the term user context for simplicity.

Module Context

A module represents program code. In the case of DS2, each module represents exactly one DS2 package, and the name of each module is the same as the name of the DS2 package that it represents. If you are unfamiliar with DS2 packages, see "Understanding DS2 Methods and Packages" in *SAS 9.4 DS2 Language Reference*. Every module is owned by exactly one user context.

SAS Micro Analytic Service supports module revisions, and is capable of hosting and executing multiple revisions of a module concurrently. When SAS Micro Analytic Service compiles a DS2 package, it creates a revision of that module. Therefore, a module is a container of revisions. It also houses any compiler warning or error messages generated from the latest revision compilation or compilation attempt.

Note: The Micro Analytic Service 1.2 REST interface supports running the latest revision only. The Java and C interfaces support multiple revisions.

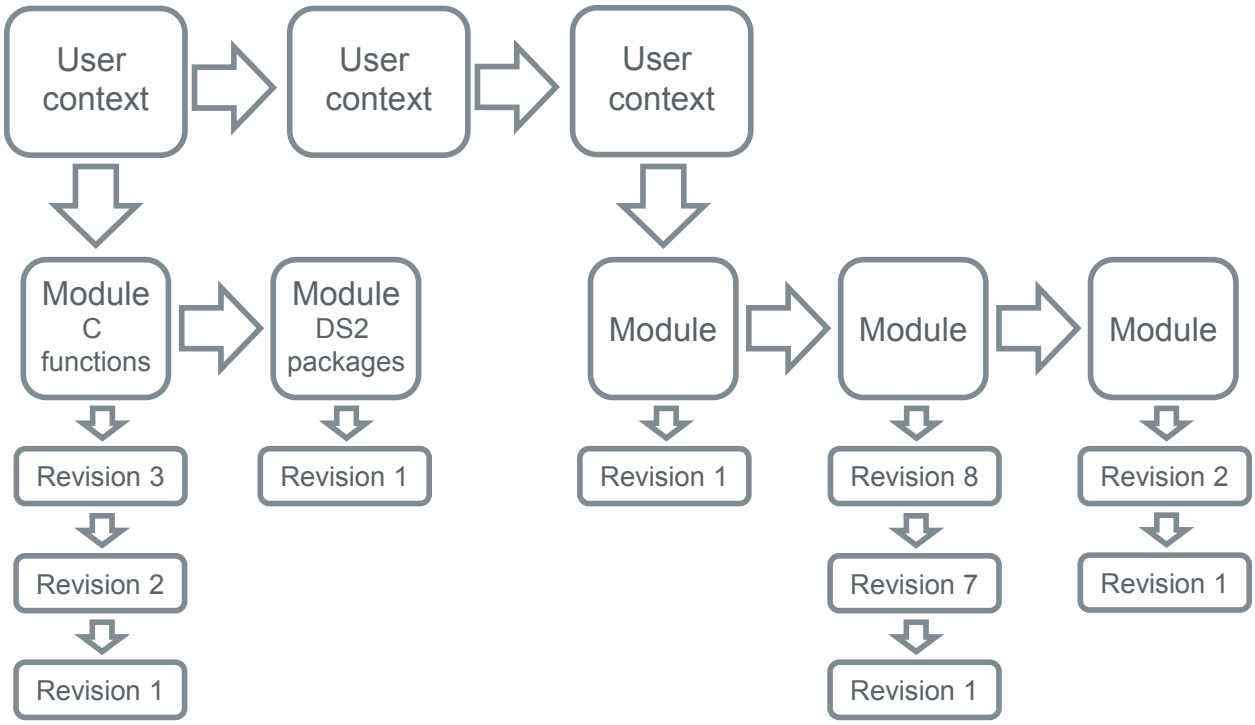
Revision

A revision is at the leaf level of the object hierarchy. Each revision contains source code, an executable code stream (optimized binary executable), and metadata. The metadata describes the methods, method signatures, and entry points in the code.

SAS Micro Analytic Service assigns a revision number to each revision, which is a monotonically increasing integer value beginning with 1. A revision is uniquely identified by module ID and revision number. When you reference a revision, specifying revision number zero selects the latest revision.

Note: A lookup is incurred when revision zero is specified. Therefore, for maximum performance, a nonzero revision number should be used when possible.

Figure 2.1 *Component Hierarchy*



Interfaces

SAS Micro Analytic Service has a layered set of interfaces:

C
: The SAS Micro Analytic Service core engine is written in C for high performance.

  Note: The C interface is not accessible. However, it is important to be aware of it because it does have an impact on threading, tuning, logging configuration, and options for monitoring.

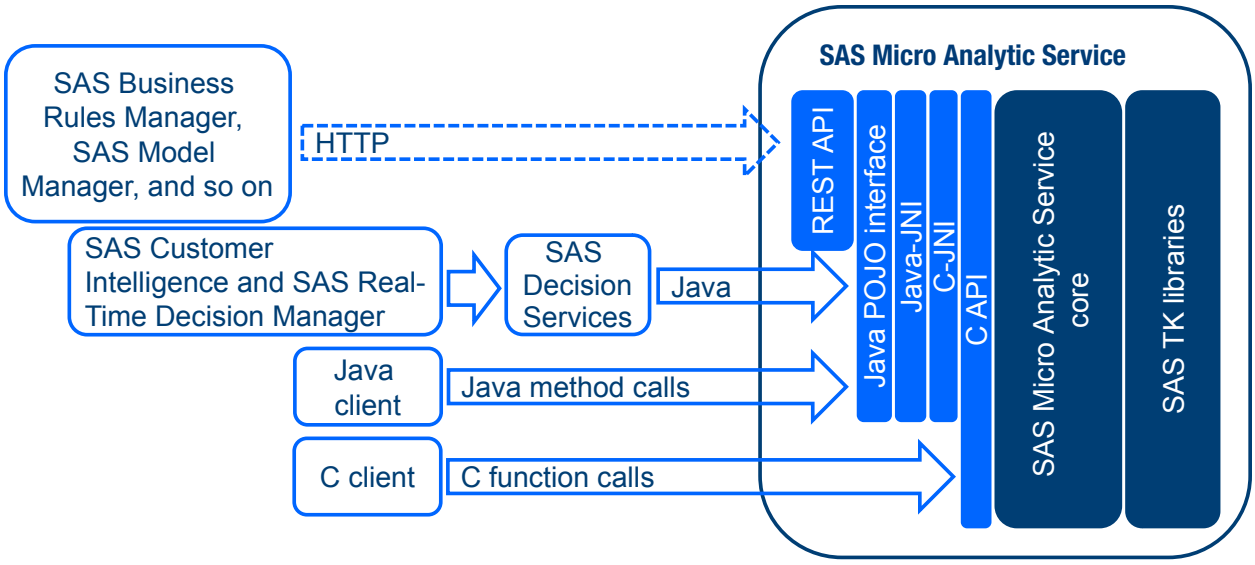
Java
: a thin Java layer that communicates with the C interface through the Java Native Interface (JNI).

REST/JSON
: adds functionality such as persistence and clustering support and communicates with the SAS Micro Analytic Service core engine through the Java interface.

All three interfaces are functionally similar. However, the REST interface handles certain functionality automatically, such as initialization and user context management. By contrast, the Java and C interfaces provide methods to control these elements directly.

The following example illustrates SAS Micro Analytic Service interfaces allowing you to publish code and execute it many times. Although you might be using SAS Micro Analytic Service 1.2 through the REST interface, this example uses the Java interface in order to show all of the steps, including those that the REST interface handles automatically.

Figure 2.2 *Interfaces Example*



Example: JAVA Interface

Instantiate SAS Micro Analytic Service

When using the Java interface, instantiating the tksfjni Java class starts SAS Micro Analytic Service.

Note: When using the REST interface, the service is started automatically when the web application is started.

```
int threads = 4;
TkLight tk = new tksfjni(threads, null);
```

The threads argument creates a threaded kernel thread pool of size 4. The SAS threaded kernel architecture is the internal architecture that enables SAS analytics. SAS Micro Analytic Service uses the worker threads in the threaded kernel thread pool to dispatch code compilation and execution tasks. For most applications, the best performance is achieved by setting the thread pool size to be approximately equivalent to the number of cores in the server where SAS Micro Analytic Service is running. Passing zero as the threads argument causes SAS Micro Analytic Service to set the thread pool size equal to the total number of logical processors that are present on the hosting server. For example, Intel hyper-threaded processors have two logical processors per core. Therefore, if threads is specified as zero on a system that has one Intel quad-core hyper-threaded processor, the thread pool size is 8. Changing the thread pool size requires restarting SAS Micro Analytic Service.

The second argument, which is null in the example above, can be used to specify the location of a logging configuration file that controls SAS Micro Analytic Service logging. SAS Micro Analytic Service uses the SAS 9.4 Logging Facility. For more information, see *SAS 9.4 Logging: Configuration and Programming Reference*.

Create a User Context

All modules are owned by a user context, so you must create a context to contain the module that is published below.

```
// Create a user context
long userCtx  = tk.newUserContext("My user context");
if (userCtx == -1) {
        System.out.println("   User context creation failed.");
        tk.term();
        return;
}
```

Note: When you use the REST interface, the creation of a context is done automatically.

The steps that follow illustrate how to publish code to SAS Micro Analytic Service, where it is compiled and prepared for high-performance execution.

Create Modules

Create a module to hold the revisions of the code by calling the newModuleContext() method, with the user context ID. Doing this causes the new module to be owned by the user context you just created.

```
long moduleCtx = tk.newModuleContext(userCtx, Language.DS2,
                                     "myPgk", 0);
if (moduleCtx == -1) {
          System.out.println("   Module context creation failed.");
          tk.term();
        return;
}
```

The second argument, Language.DS2, specifies that you are using this module to publish a DS2 package. The third argument is the package name, which should match the package name in the source code. If it does not, SAS Micro Analytic Service corrects the name upon successfully creating the revision.

Now you need source code to publish. See Chapter 3, "DS2 Programming for SAS Micro Analytic Service," for information about writing code for SAS Micro Analytic Service. This example uses the following simple program:

```
String myCode = "ds2_options sas;\n" +
            "package myPkg;\n" +
            "method myMth(int i, in_out int j);\n" +
            "   j = i + 5;\n" +
            "end;\n" +
            "endpackage;\n";
```

The source code is passed to SAS Micro Analytic Service as a string argument to the newRevision() method. Notice the escaped newline characters at the end of each line. If you read in the source code from a file, the newline characters are included. However, if you use a literal string as above, the best practice is to include newline characters so that any compiler messages can be easily traced to the given line number.

To publish your code to SAS Micro Analytic Service, call the newRevision() method, passing in the module ID, source code string, and a description. The last two parameters are for options that are not used with DS2.

```
int rev = tk.newRevision(moduleCtx, myCode,
```

```
                    "my DS2 package", null, 0);
if (rev > 0) {
            System.out.println("   Revision " + rev + " created.");
}
else {
            System.out.println("   Revision not created.");
}
```

If the revision number that is returned is greater than zero, the code compiled correctly and is now ready to be executed. Otherwise, call the following function in order to check the compiler messages for errors:

```
String[] msgs = tk.getCompilationMessages(moduleCtx);
```

The example package contains one method, called “myMth,” which has one input and one output argument. Before executing the method, you must create the method arguments. The Java interface uses a tksfValues object to pass arguments to and from the method to execute.

```
tksfValues args = new tksfValues(2, 1);
```

The first parameter to the tksfValues constructor is the total number of method arguments, including both inputs and outputs. The second parameter specifies the number of output arguments from the method. Because DS2 output arguments are passed by reference, both inputs and outputs must be populated in the tksfValues object. For more information, see Chapter 3, “DS2 Programming for SAS Micro Analytic Service,” on page 11.

```
args.setInt(3);
        args.setOutInt();
```

Note: The REST API does not support method overloading.

Continue to change the value in the argument vector and call the execute() function, as many times as needed. The execute() function takes the following arguments: user context ID, module ID, revision number (passing zero selects the latest revision), the name of the method to execute, and the populated tksfValues object.

```
rc = tk.execute(userCtx, moduleCtx, rev, "myMth", args);
```

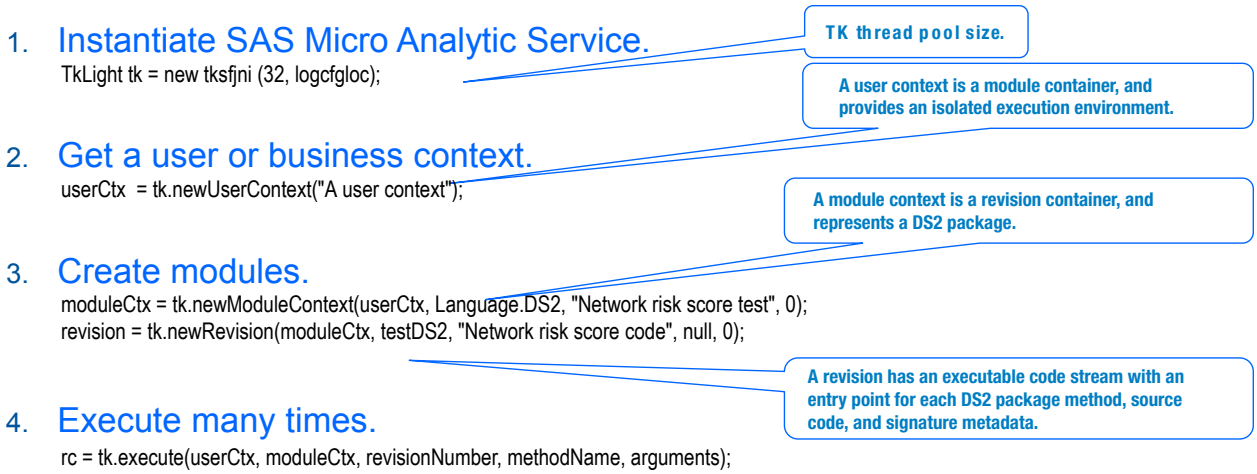
Successful execution returns zero for the return code, and the results can be retrieved from the tksfValues object. Arguments are positional and are retrieved by the zero-based index. In this case, the single integer output value can be retrieved from index position 1 (the second slot in the argument’s object).

```
if (rc == 0)
            int result = args.getInt(1);
```

Basic Steps for Using SAS Micro Analytic Service

Using SAS Micro Analytic Service involves four steps. When you are using the REST interface, the first two are handled automatically.

Figure 2.3 *Annotated Steps*



Chapter 3

DS2 Programming for SAS Micro Analytic Service

Overview

SAS Micro Analytic Service 1.2 supports a subset of the DS2 programming language that is suitable for high-performance transaction processing in real time. This chapter covers only that subset. Note that DS2 batch processing is not supported.

For more information about the DS2 programming language, see *SAS 9.4 DS2 Programming Reference*.

Publishing DS2 Source Code to SAS Micro Analytic Service

The DS2 source code submitted to SAS Micro Analytic Service should begin with

```
"ds2_options sas"
```

It should end with

```
"endpackage"
```

The code cannot contain DATA statements, PROC statements, or FedSQL connection strings. The source code should contain one and only one DS2 package, and this package can contain as many methods as desired.

It is a best practice to include a line feed character at the end of each source code line. This line feed character makes it easier to use compiler warning and error messages that include line numbers.

SAS Micro Analytic Service and SAS Foundation

Although DS2 is supported by both SAS Foundation and SAS Micro Analytic Service, SAS Micro Analytic Service has a lightweight, high-performance engine, which does not support either the full SAS language or PROC statements. Therefore, PROC statements cannot be used. However, here is an effective DS2 authoring and testing mechanism: develop your DS2 packages in SAS Foundation using PROC DS2, and publish those packages to SAS Micro Analytic Service after removing the surrounding PROC DS2 syntax.

It is recommended that every DS2 module that you publish to SAS Micro Analytic Service include the following on the first line of code, just above the PACKAGE statement:

```
ds2_options sas;
```

This option instructs DS2 to use SAS missing value handling, and helps ensure that your DS2 program behaves the same as if it were run in SAS Foundation.

I/O

SAS Micro Analytic Service 1.2 is a limited functionality release that does not support database, SAS data set, or file I/O other than logging. As a result, the following DS2 features are not supported by SAS Micro Analytic Service 1.2:

- Package SQLStmt
- The features of package Hash that enable the population of a hash package from the contents of a file

SAS Micro Analytic Service 1.2 does support the DS2 HTTP package, which can execute HTTP requests to, and get responses from, HTTP and HTTPS web services.

Programming Blocks

Each DS2 module represents exactly one package, so the DS2 PACKAGE statement plays a major role in SAS Micro Analytic Service. A DS2 package contains one or more methods, and methods can contain a wide variety of DS2 language constructs. Package methods work well with rapid transaction processing because they can be called over and over again with little overhead, as transactions flow through the system. By contrast, the DS2 THREAD and TABLE statements are batch-oriented and are not supported.

The following code blocks are supported:

- PACKAGE…ENDPACKAGE
- METHOD…END
- DO…END

The following code blocks are batch-processing oriented and are not supported:

- TABLE…ENDTABLE
- THREAD…ENDTHREAD

Similarly, the following statements are not supported: OUTPUT and SET

- OUTPUT
- SET

Public and Private Methods and Packages

Overview

Private methods and packages are SAS Micro Analytic Service concepts, rather than DS2 features.

SAS Micro Analytic Service can host public DS2 packages and private DS2 packages. Private DS2 packages have fewer restrictions on the DS2 features that can be used than public packages have. Although a private DS2 package cannot be called directly, it can be called by another DS2 package. Private DS2 packages are useful as utility functions, as solution-specific built-in functions, or for solution infrastructure. See your SAS solution documentation for a description of the solution-specific built-in functions that you can use when authoring custom DS2 modules.

As with public packages, SAS Micro Analytic Service 1.2 private packages do not support I/O.

A public DS2 package can contain private methods, as long as it contains at least one public method. Any method that does not conform to the rules for public methods is automatically treated as private. Private methods are allowed and do not produce errors if they contain correct DS2 syntax. Private methods are not callable externally. Therefore, they do not show up when querying the list of methods within a package. However, they can be called internally by other DS2 package methods. Here are several typical uses of private methods:

- Small utility functions that return a single, non-void, result.
- Methods containing DS2 package arguments. These are not callable externally.

Public Method Rules

Public methods must conform to the following rules:

- The return type must be void. Rather than using a single return type, public methods can return multiple outputs, where each output argument specifies the in_out keyword in the method declaration. Non-void methods are treated as private.
- Arguments that are passed by reference (meaning ones that specify in_out) are treated as output only. True update arguments are not supported by public methods.

This restriction results in more efficient parameter marshaling and supports all interface layers, including REST.

- Input arguments must precede output arguments in the method declaration. It is permissible for a method to have only inputs or only outputs. However, if both are present, all inputs must precede the outputs.
- All argument data types are publicly supported. See "Supported DS2 Data Types" on page 16.

Public Method Example

The example below illustrates a valid public method. It has a void return type (no RETURNS clause), uses only publicly supported data types, and treats in_out arguments as output only.

```
method quickSortStep (int lowerIndex, int higherIndex, in_out double numbers[10]);

      dcl int i;
      dcl int j;
      dcl int pivot;
      dcl double temp;

      i = lowerIndex;
      j = higherIndex;

      /* Calculate the pivot number, taking the pivot as the
       * middle index number. */
      pivot = numbers[ceil(lowerIndex+(higherIndex-lowerIndex)/2)];

      /* Divide into two arrays */
      do while (i <= j);
            /**
             * In each iteration, identify a number from the left side that
             * is greater than the pivot value. Also identify a number
             * from the right side that is less than the pivot value.
             * Once the search is done, then exchange both numbers.
             */
            do while (numbers[i] < pivot);
              i = i+1;
            end;
            do while (numbers[j] > pivot);
                j = j-1;
            end;
            if (i <= j) then do;
                temp = numbers[i];
                numbers[i] = numbers[j];
                numbers[j] = temp;

                /* Move the index to the next position on both sides. */
                i = i+1;
                j = j-1;
            end;
        end;

        /* Call quickSort recursively. */
```

```
        if (lowerIndex < j) then do;
            quickSortStep(lowerIndex, j, numbers);
        end;
        if (i < higherIndex) then do;
            quickSortStep(i, higherIndex, numbers);
        end;
    end;
```

Here is another example of a public method that illustrates the use of the HTTP package calling out to a web service using a POST request and then getting a response.

```
method httppost( nvarchar(8192) url,
                 nvarchar(67108864) payload,
                 in_out nvarchar respbody,
                 in_out int hstat, in_out int rc );
  declare package http h();
  rc = h.createPostMethod( url );
  if rc ne 0 then goto Exit;
  rc = h.setRequestContentType( 'application/json;charset=utf-8' );
  if rc ne 0 then goto Exit;
  rc = h.addRequestHeader( 'Accept', 'application/json' );
  if rc ne 0 then goto Exit;
  rc = h.setRequestBodyAsString( payload );
  if rc ne 0 then goto Exit;
  rc = h.executeMethod();
  if rc ne 0 then goto Exit;
  hstat = h.getStatusCode();
  if hstat lt 400 then h.getResponseBodyAsString( respbody, rc );
  else respbody = '';
  Exit:
  h.delete();
end;
```

Private Method Example

The example below generates a private method in SAS Micro Analytic Service. It has a non-void return type. That is, it has a RETURNS clause in the declaration, which specifies a single integer return value.

```
method isNull(double val) returns int;
  return null(val) OR missing(val);
end;
```

Method Overloading

SAS Micro Analytic Service supports method overloading. In DS2, when two or more methods in the same package have the same name, those methods are said to be *overloaded*. When overloaded methods are used, the method signature (list of input and output parameters and their types) is used to select the correct method to execute. Because a method signature includes both input and output parameters, any output parameter types must always be set in tksfValues before executing the method.

Note: Each module constitutes a separate name space and corresponds to one DS2 package. Therefore, two DS2 methods with the same name, in different modules, are not considered overloaded.

Note: The C language does not support method overloading. Syntax errors occur if two C functions with the same name exist in the source code of the same C module. Therefore, only DS2 package methods can be overloaded in SAS Micro Analytic Service 1.2.

The following functions enable you to query information about overloaded methods. For more information about these methods, see Chapter 5, "Java Interface Reference," on page 25.

getStepInputs()
: The version of getStepInputs() that takes an index parameter retrieves descriptions of the input parameters of the overloaded method indicated by name and index value. That is, you can use getStepInputs() to query the input arguments for overloaded method 1, 2, 3, and so on. To do this, specify 1, 2, 3, and so on, for the index value.

getStepOutputs()
: works similarly to getStepInputs(), but retrieves descriptions of the specified method's output parameters.

getOverloadedStepCount()
: returns the number of overloaded methods that exist in the specified module having the specified name.

isOverloaded()
: returns True if the specified method is overloaded and False if not.

Argument Types Supported in Public Methods

Overview

SAS Micro Analytic Service supports a subset of the DS2 data types for use as public method arguments. Data types in the unsupported list can still be used in the body of a (public or private) DS2 package method, and as arguments to private methods. The lists of publicly supported and unsupported data types are given below.

Note: Any additional types added to the DS2 programming language in future releases should be considered unsupported unless otherwise stated in the SAS Micro Analytic Service documentation.

Supported DS2 Data Types

The following are the supported DS2 data types:

- BIGINT
- CHAR(n)
- DOUBLE
- FLOAT
- INTEGER
- NCHAR(n)
- NVARCHAR(n)
- REAL

- SMALLINT
- VARCHAR(n)

Unsupported DS2 Data Types

The following are the unsupported DS2 data types:

- BINARY(n)
- DATE
- DECIMAL(p, s)
- NUMERIC(p, s)
- TIME(p)
- TIMESTAMP(p)
- TINYINT
- VARBINARY(n)
- PACKAGE

Chapter 4

Best Practices for DS2 Programming

Overview

This section describes best practices that are recommended when programming in DS2 for any environment. They are not unique to SAS Micro Analytic Service.

Global Packages Versus Local Packages

Overview

The scope of a package instance makes a difference. Package instances that are created in the global scope typically are created and deleted (allocated and freed) once and used over and over again. Package instances that are created in a local scope are created and deleted each time the scope is entered and exited. For example, a package instance that is created in a method's scope is created and deleted each time a method is called. The creation and deletion time can be costly for some packages.

The following examples use the hash package. This technique can be used for all packages.

Fast

This example creates a hash package instance that is global, created and deleted with the package instance, and reused between calls to load_and_clear.

```
/** FAST **/
package mypack;
  dcl double k d;
  dcl package hash h([k], [d]);

  method load_and_clear();
    dcl double i;
    do k = 1 to 100;
      d = 2*k;
      h.add();
    end;
    h.clear();
  end;
endpackage;
```

Slow

This example creates a hash package instance that is local to the method and created and deleted for each call to load_and_clear.

```
/** SLOW **/
package mypack;
  dcl double k d;

  method load_and_clear();
    dcl package hash h([k], [d]);
    dcl double i;
    do k = 1 to 100;
      d = 2*k;
      h.add();
    end;
    h.clear();
  end;
endpackage;
```

Replacing SCAN (and TRANWRD) with DS2 Code

Consider the following code:

```
i = 1;
onerow = TRANWRD(SCAN(full_table, i, '|'), ';;', ';-;');
do while (onerow ~= '');
    j = 1;
    elt = scan(onerow, j, ';');
    do while (elt ~= '');
        * processing of each element in the row;
        j = j+1;
        elt = SCAN(onerow, j, ';');
```

```
    end;
    i = i+1;
    onerow = TRANWRD(SCAN(full_table, i, '|'), ';;', ';-;');
end;
```

You can make the following observations:

- SCAN consumes adjacent delimiters. Therefore, TRANWRD is required to manipulate each row into a form that can be traversed element by element.
- SCAN starts at the front of the string each time. Therefore, the aggregate cost is O(N^2).
- SCAN and TRANWRD require NCHAR or NVARCHAR input. If full_table is declared as a CHAR or VARCHAR input, it must be converted to NVARCHAR, then processed, and then converted back to VARCHAR in order to be captured into the onerow value.

Here is code that replaces this type of loop with a native DS2 solution and that thus avoids these problems by collecting the necessary details into a package:

```
dcl package STRTOK row_iter();
dcl package STRTOK col_iter();
row_iter.load(full_table, '|');
do while (row_iter.hasmore());
    row_iter.getnext(onerow);
    col_iter.load(onerow, ';');
    do while (col_iter.hasmore());
        col_iter.getnext(elt)
        * processing of each element;
    end;
end;
```

The supporting package, STRTOK, is shown below. It can be used to replace SCAN and TRANWRD pairs anywhere in DS2.

```
/** STRTOK package - extract subsequent tokens from a string.
 * So named because it mirrors (in a safe way) what is done by the original
 * strtok(1) function available in C.
 */
package sasuser.strtok/overwrite=yes;
  dcl varchar(32767) _buffer;
  dcl int strt blen;
  dcl char(1) _delim;

  /* Loads the current object with the supplied buffer and delimiter
   * information. This avoids the cost of constructing and destructing the
   * object, and allows the declaration of a STRTOK outside of the loop in which
   * it is used.
   */
  method load(in_out varchar bufinit, char(1) delim);
    _buffer = bufinit .. delim;
    _delim = delim;
    strt = 1;
    blen = length(_buffer);
  end;

  /* Are there more fields? 1 means there are more fields. 0 means there are
   * no more fields.
   */
```

```
  method hasmore() returns integer;
    if (strt >= blen) then return 0;
    return 1;
  end;

  /* The void-returning GETNEXT method places the next token in the supplied
   * variable, tok.
   */
  method getnext(in_out varchar tok);
    dcl char(1) c;
    dcl int e;
    tok = '';
    if (hasmore()) then do;
      e = strt;
      c = substr(_buffer,e,1);
      do while (c ~= _delim);
        tok = tok .. c;
        e = e + 1;
        c = substr(_buffer,e,1);
      end;
      strt = e + 1;
    end;
  end;

  /* The value-returning GETNEXT method returns the next token. This version is
   * more computationally expensive because it requires an extra copy, as opposed to
   * the void-returning version, above.
   */
  method getnext() returns varchar(32767);
    dcl varchar(32767) tok;
    getnext(tok);
    return tok;
  end;

  /* Construct a STRTOK object using the parameters as initial values.
   */
  method strtok(varchar(32766) bufinit, char(1) delim);
    load(bufinit, delim);
  end;

  /* Construct a STRTOK object without an initial buffer to be consumed.
   */
  method strtok();
    strt = 0; blen = 0;
  end;
endpackage; run;
```

Using STRTOK instead of SCAN and TRANWRD avoids the CHAR to NCHAR conversions and reduces CPU because of how STRTOK retains the intermediate state between calls to the getnext() methods. Therefore, it is O(N) instead of O(N^2).

Hash Package

With both the DATA step and DS2, note the size of the key. A recent program carried out many hash lookups with a 356-byte key. Hashing is an O(1) algorithm; the "1" with the hash package is the length of the key. The longer the key, the longer the hash function takes to operate.

```
dcl char(200) k1 k2;
dcl double d1 d2;

/* If k1 and k2 are always smaller than 200, then    */
/*  size them smaller to reduce the time spent in    */
/*  the hash function when adding and finding values */
/*  in the hash package.                             */
dcl package hash([k1 k2], [d1 d2]);
```

Character-to-Numeric Conversions

When converting a string to a numeric value, note the encoding of the string. When the string is a single-byte encoding, DS2 translates the value to a TKChar (UCS-2 or UCS-4) for conversion. The longer the string, the longer the time it takes to do the conversion.

```
dcl char(512) s;
dcl nchar(512) ns;
dcl double x;
s = '12.345';
ns = '12.345';

x = s;                /* slow */
x = substr(s,1,16);   /* faster */
x = substr(ns,1,16);  /* even faster, avoids transcoding */
```

Passing Character Values to Methods

In SAS Micro Analytic Service, DS2 method input parameters are passed by value. What this means is that a copy of the value is passed to the method. When passing character parameters, a copy of the parameter is made to ensure that the original value is not modified. Making sure that character data is sized appropriately ensures that less copying occurs.

DS2 method output parameters, which are specified by the in_out keyword, are passed by reference. Therefore, no copy is made.

```
method copy_made(char(256) x);
  ...
end;

method no_copy(in_out char x);
```

```
  ...
end;
```

Performing the Computation Once

If a computation is repeated multiple times to compute the same value, you can perform the computation once and save the computed value. For example, the following code block performs the computation, compute(x), four times:

```
if compute(x) > computed_max then computed_max = compute(x);
if compute(x) < computed_min then computed_min = compute(x);
```

If compute(x) always computes the same value for a given value of x, then the code block can be modified to perform the computation once and save the computed value:

```
computed_x = compute(x);
if computed_x > computed_max then computed_max = computed_x;
if computed_x < computed_min then computed_min = computed_x;
```

Moving Invariant Computations Out of Loops

If a computation inside a loop computes the same value for each iteration, improve performance by moving the computation outside the loop. Compute the value once before the loop begins and use the computed value in the loop. For example, in the following code block, compute(x) is evaluated during each iteration of the DO loop:

```
do i = 1 to dim(a);
  if (compute(x) eq a[i]) then ...;
end;
```

If compute(x) is invariant (meaning that it always computes the same value for each iteration of the loop), then the code block can be modified to perform the computation once outside the loop:

```
computed_x = compute(x);
do i = 1 to dim(a);
  if (computed_x eq a[i]) then ...;
end;
```

Chapter 5

Java Interface Reference

Overview

The Java interface allows tightly coupled Java client applications to drive SAS Micro Analytic Service directly though Java method calls. This is made possible because the Java interface provides fine-grained control of SAS Micro Analytic Service, and does not hide detailed functionality. By contrast, the REST interface, in the interest of usability and simplicity, handles many interactions automatically. A typical SAS Micro Analytic Service 1.2 client uses the REST interface, which in turn uses the Java interface described in this chapter.

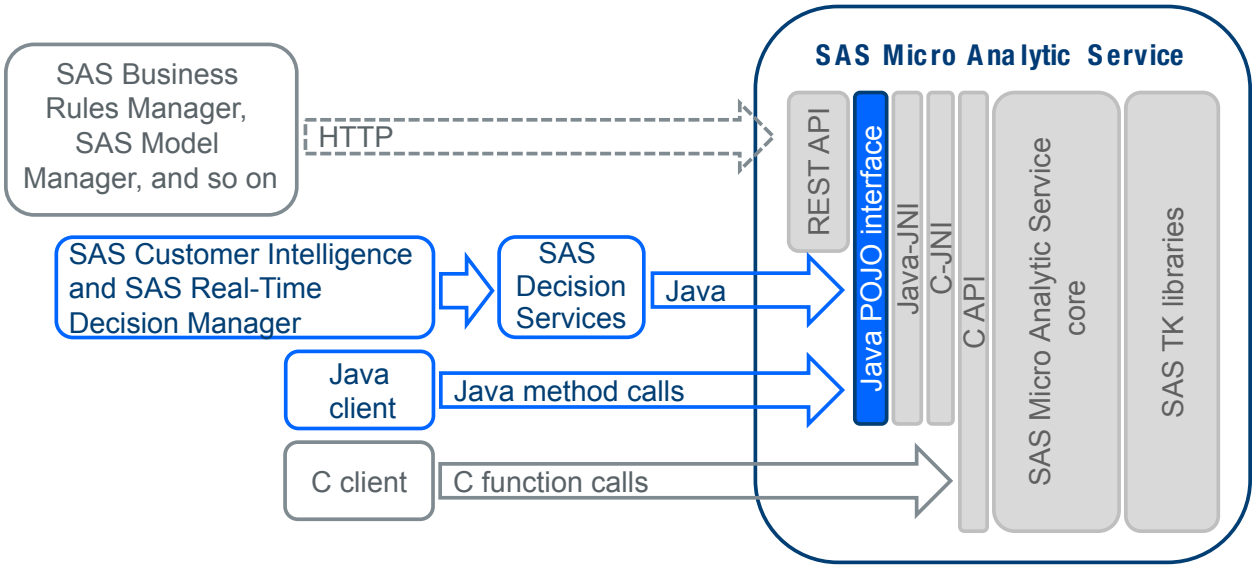
The Java interface enables client-supplied DS2 and C programs to be published to SAS Micro Analytic Service, where they are compiled into modules and made available for repeated execution. The interface also includes methods for querying information about

currently loaded artifacts, such as user contexts, modules, methods, or functions (sometimes called steps), and step signatures (such as input and output arguments).

Topology

As you can see from the following figure, the Java interface is positioned between the REST interface and the C interface. Dependencies between the three interface layers are strictly one way. The C interface does not depend on the REST or Java interfaces. In fact, C clients can omit the Java layers altogether. The Java interface communicates with SAS Micro Analytic Service strictly through the C interface and does not depend on the REST interface. Similarly, the REST interface communicates with SAS Micro Analytic Service strictly through the Java interface.

Figure 5.1 *Java Interface*



Start-Up

The SAS Micro Analytic Service 1.2 Java interface is a public Java interface named TkLight, obtained by instantiating Java class tksfjni, which implements the interface.

```
int threads = 4;
String logconfigloc = null;
TkLight tk = new tksfjni(threads, logconfigloc);
```

CAUTION:

The tksfjni instance must be kept a singleton, as it is not advisable to start two instances of SAS Micro Analytic Service within the same process space. This would yield unpredictable results and is not supported.

The first tksfjni constructor argument specifies the size of the threaded kernel thread pool to be maintained by SAS Micro Analytic Service. For best performance, the size of the thread pool should be about equal to the number of processor cores in the server

hosting SAS Micro Analytic Service. The exact number of threaded kernel threads to specify for best performance varies somewhat, depending on the characteristics of the code to be published. When the number of threads that is specified exceeds four times the number of cores of the hosting server, SAS Micro Analytic Service fails to start.

The second tksfjni constructor argument, when non-null, specifies the location of a logging configuration file, which controls SAS Micro Analytic Service logging. SAS Micro Analytic Service uses the SAS 9.4 Logging Facility. For more information, see *SAS 9.4 Logging: Configuration and Programming Reference*. Your SAS solution might provide a default logging configuration. For more information, see your solution's documentation.

Shutdown

SAS Micro Analytic Service is shut down by calling term().

```
tk.term();
```

Because SAS Micro Analytic Service is an in-memory service, all published artifacts (user contexts, modules, and revisions) are purged. If SAS Micro Analytic Service is restarted when using the Java interface, any previously purged artifacts are not restored and must be re-published if desired. However, the REST interface includes an artifact persistence feature that restores the state of SAS Micro Analytic Service automatically. For more information, see Chapter 6, "SAS Micro Analytic Service REST API," on page 49.

User Context Methods

User (or business) contexts have two primary functions:

- A user context is a container of modules. Every module is parented to one and only one user context.
- User contexts provide isolated execution environments. The operations of one user context are not visible to any other user context.

Note: Recall that there is no functional difference between a user context and a business context. Whether contexts are used to provide each user with their own environment, or whether they are used to insulate business units from one another, is application-specific. The set of functionality is the same. SAS Micro Analytic Service uses the UserContext construct for both business and user contexts.

Here is a list of available methods:

newUserContext

```
long newUserContext(java.lang.String displayName)
```

Creates a new user or business context and returns an opaque pointer to it. Modules are organized by user context. User contexts are isolated from one another.

Parameter

displayName - The name that is used in log messages.

Return

handle - The opaque context identifier (opaque pointer) that is used to pass to functions that require a user context identifier.

deleteUserContext

```
void deleteUserContext(long userContext)
```

Deletes a user context and all modules that are associated with it.

Parameter

userContext - The opaque user context handle of the context to delete.

userContextExists

```
boolean userContextExists(long userContext)
```

Queries the existence of a user context.

Parameter

userContext - The opaque user context handle.

Return

Boolean - True if the context exists, False otherwise.

getUserContextDisplayName

```
java.lang.String getUserContextDisplayName(long userContext)
```

Retrieves the display name of the given user context.

Parameter

userContext - The opaque user context handle.

getModuleIDs

```
long[] getModuleIDs(long userContext)
```

Retrieves the IDs of all modules that are currently loaded in the system.

Return

The long array of the module IDs.

getModuleNames

```
java.lang.String[] getModuleNames(long userContext)
```

Retrieves the names of all modules that are currently loaded in the system.

Returns

The string array of the module IDs.

Note: The C modules are named according to the value of the displayName argument passed to newModuleContext(). DS2 modules are named for the DS2 package that the module represents (such as the package name given in the source code).

Note: If the value of the displayName argument to newModuleContext() differs from the DS2 package name that was given in the source code, the module name changes when the first revision of the DS2 package is created.

getUserContextCreationDateTime

```
java.util.Date getUserContextCreationDateTime(long userContext)
```

Gets the user context creation datetime.

Parameter

userContext - The opaque user context handle.

Return

The creation datetime in SAS form (GMT seconds since 1960).

getUserContextLastModifiedDateTime

```
java.util.Date getUserContextLastModifiedDateTime(long userContext)
```

Gets the user context last modified datetime.

Parameter
: userContext - The opaque user context handle.

Return
: The datetime of the last modification in SAS form (GMT seconds since 1960).

Here is an example of creating a new user context:

```
long userCtx  = tk.newUserContext("A user context");
if (userCtx <= 0) {
        System.out.println("   User context creation failed.");
        return;
}
```

The user context handle that is returned from newUserContext() actually represents a C language pointer that is maintained in the SAS Micro Analytic Service core. The pointer is opaque to Java and represented as a value of type long. This value can be used to pass to any method that requires a user context identifier. It should not be modified.

Module Context Methods

The terms *module* and *module context* are used interchangeably. In SAS Micro Analytic Service 1.2, a module context represents either of the following:

named C module
: a collection of related C functions

DS2 package
: a collection of related DS2 methods (also referred to in SAS Micro Analytic Service as a *DS2 module*).

A module context contains revisions of C or DS2 code. The revisions within a module context might contain different source code, but all revisions must represent the same named DS2 package or C module. That is, for DS2 modules the package name that is given in the source code is not allowed to change from revision to revision. If it does, an error is returned and the revision is not created.

In addition to zero or more revisions, a module context maintains information such as a description, highest revision number used, number of revisions present, programming language (C or DS2, for SAS Micro Analytic Service), messages from the most recent compilation, creation date and time, and last modified date and time.

Here is a list of available module contexts:

newModuleContext
```
long newModuleContext(long userContext,
           TkLight.Language language,
           java.lang.String displayName,
           java.util.EnumSet<TkLight.ModuleOptions> options)
```

Creates a new module context within the given user context. A module is a collection of related steps (a DS2 package or collection of C routines). Add code to the module context by calling newRevision().

Note: For DS2 packages, the module displayName is assigned the DS2 package name when the first revision is created.

Note: DS2 packages containing non-void methods or methods with DS2 package arguments are treated as internal and are not returned in queries for the list of methods. It is permissible to include private methods that do not produce errors,

as long as the code is valid. See "Public and Private Methods and Packages" on page 13.

Parameters

userContext - The opaque handle returned from newUserContext().

language - Source language (C or DS2).

displayName - For DS2, set it to the DS2 package name. For C, any text identifier is acceptable.

options - If the EnumSet contains enum INTERNAL, the module context and all revisions are private.

Return

module context - The module context created (opaque handles), returns zero on error.

Note: The module context handle that is returned from newModuleContext() actually represents a C language pointer in SAS Micro Analytic Service core. This handle is opaque to Java and represented as a value of type long. This value is used to pass to any method that requires a module context identifier, and should not be modified.

deleteModuleContext

```
void deleteModuleContext(long moduleContext)
```

Deletes the specified module context and its modules and revisions.

Parameter

moduleContext - The opaque module context handle

moduleContextExists

```
boolean moduleContextExists(long moduleContext)
```

Queries the existence of a module context.

Parameter

moduleContext - The opaque module context handle

Return

Boolean - True if context exists, False otherwise.

getModuleContextDisplayName

```
java.lang.String
getModuleContextDisplayName(long moduleContext)
```

Retrieves the display name of the given module context.

Parameter

moduleContext - The opaque module context handle.

getModuleContextByName

```
long getModuleContextByName
(long userContext,java.lang.String name)
```

Retrieves the module context opaque handle, when given the display name. This method is provided as a convenience for clients that choose to identify modules by name. Callers are responsible for guaranteeing name uniqueness.

Parameters

userContext - The opaque handle returned from newUserContext().

name - DS2 package name or C module display name.

Note: For DS2 packages, the module context name is the name of the DS2 package, which is set when the first revision is created. If a module context must be retrieved by name before the first revision is created, then the displayName argument to newModuleContext() must be set to the correct name (the name of the DS2 package to be subsequently published) when the module context is created.

getModuleContextCreationDateTime

```
java.util.Date
getModuleContextCreationDateTime(long moduleContext)
```

Gets the module context creation datetime.

Parameter

moduleContext - The opaque handle returned from newModuleContext().

Returns

Creation datetime in SAS form (GMT seconds since 1960).

getModuleContextLastModifiedDateTime

```
java.util.Date
getModuleContextLastModifiedDateTime(long moduleContext)
```

Gets the module context's last modified datetime.

Parameter

moduleContext - The opaque handle returned from newModuleContext().

Return

The creation datetime in SAS form (GMT seconds since 1960).

registerModuleName

```
int registerModuleName(long moduleContext,
                       java.lang.String name)
```

Registers a module name so that it can participate in calls across code streams. Call this function to enable other C modules to call functions on the given module. Calls across code streams can be made by C modules only.

Note: Although DS2 cannot call across code streams, DS2 package methods can call methods in other packages. An in-memory DS2 program repository enables DS2 package references to be resolved at compile time, so that a complete set of referenced packages is compiled into every code stream. Therefore, calls across code streams are not necessary with DS2 modules. However, C modules require external modules to be registered, using this method, in order to locate the external module at run time.

Parameter

moduleContext - The opaque handle returned from newModuleContext().

name - The name to assign the module (must be unique within the user context).

Return

result code - Zero indicates successful registration nonzero indicates registration failed.

Revision Methods

Overview

A revision represents the executable code of a module. A module might contain zero or more revisions. More than one revision of a given module can be executed concurrently by different callers, or at different times by the same caller. Passing in zero for a revision number gets the latest revision of the specified module.

For example, caller A might rely on specific behavior in revision 3 of module X. You might set up caller B to always use the latest version of module X. To execute the desired code, caller A would specify the ID of module X and revision 3:

```
revision = 3;
rc = tk.execute(userCtx, moduleX, revision, methodName, args);
```

Caller B would specify the same module ID and revision zero:

```
revision = 0;
rc = tk.execute(userCtx, moduleX, revision, methodName, args);
```

Revision numbers are monotonically increasing values starting at 1. Revisions can be created and deleted. However, revision numbers are never reused. This behavior prevents the code of a revision from unexpectedly changing out from under a caller. Once a revision has been deleted, attempts to execute or query that revision receive a result code indicating the revision was not found.

Parameter Descriptions

Some of the revision-specific methods, such as getStepInputs() and getStepOutputs(), return instances of the public class tksfParmdef, which stands for parameter definition. Each tksfParmdef instance describes one parameter of a C function or DS2 package method. tksfParmdef contains the following member variables, along with getters and setters for each:

```
public int         index;  // parameter position (zero-based)
    public String     name;      // parameter name
    public sftype     type;      // parameter type
    public int        dim;      // if array, dimension; ignored otherwise
    public int        size;      // if varchar, max length; ignored otherwise
```

The parameter type member variable, sftype, is defined as the following enumeration, which represents the Java data types that are supported by SAS Micro Analytic Service:

```
// Supported data types
    public enum sftype {
        string_t,
        char_t,
        long_t,
        double_t,
        int_t,
        stringArray_t,
        charArray_t,
        longArray_t,
        doubleArray_t,
```

```
intArray_t
```

Method Descriptions

Here are the available revision methods:

newRevision

```
int newRevision(long moduleContext,
     java.lang.String sourceCode,
     java.lang.String description,
     java.lang.String[] CEntryPoints,
     java.util.EnumSet<TkLight.RevisionOptions> options)
```

The source code is compiled into executable form and, if it is successful, a revision number is assigned by SAS Micro Analytic Service. Revisions are uniquely identified by module context ID and revision number. Once created, a revision can be executed many times.

If source code compilation fails during newRevision(), the revision is not created, and any compiler warning or error messages are maintained by the owning module context until the next time newRevision() is called for the given module context.

Note: For DS2 packages, the parent module context displayName is assigned the DS2 package name (given in the source code) when the first revision of that module context is successfully created. After the first revision is created, the name cannot be changed. Attempts to create a revision of a DS2 module with a different package name than the original revision result in an error.

Note: DS2 packages containing non-void methods or methods with DS2 package arguments are treated as private and are not returned in queries for methods. It is permissible to include private methods, and private methods do not produce errors as long as the code is valid. For more information, see "Public and Private Methods and Packages" on page 13.

Parameters

moduleContext - The opaque handle returned from newModuleContext().

sourceCode - The string containing the source code (must match language of original source code).

description - User-supplied text.

CEntryPoints - If C code, this refers to the names of the entry points (functions) to make available externally.

Return

revisionNumber - The new revision number.

deleteRevision

```
void deleteRevision(long moduleContext,
                    int revision)
```

Deletes the module revision that is specified by module context and revision number.

Parameter

moduleContext - The opaque handle returned from newModuleContext().

getCompilationMessages

```
java.lang.String[]
getCompilationMessages(long moduleContext)
```

Retrieves any compiler diagnostic messages from the latest compilation (newRevision call) of the given module context.

Note: Only compilation messages from the most recent compilation, per module context, are retained.

Parameter
moduleContext - The opaque handle returned from newModuleContext().

Return
An array of strings containing compiler diagnostic messages.

revisionIsValid

```
boolean revisionIsValid(long moduleContext,
                        int revision)
```

Returns True if the revision is valid, False if otherwise.

Parameters
moduleContext - The opaque handle returned from newModuleContext().

revision - The revision number returned from newRevision().

Return
Boolean - True if the revision is valid, and False if otherwise.

getStepIDs

```
java.lang.String[] getStepIDs
(long moduleContext, int revision)
```

Retrieves the IDs of steps (DS2 package methods or C functions) contained by the revision.

Parameters
moduleContext - The opaque handle returned from newModuleContext().

revision - The revision number returned from newRevision().

Return
The string array containing a list of step IDs (DS2 package method names or C function names).

getStepDescription

```
java.lang.String getStepDescription
(long moduleContext, int revision, java.lang.String stepId)
```

Retrieves the step description.

Parameters
moduleContext - The opaque handle returned from newModuleContext().

revision - The revision number returned from newRevision().

stepId - The name of the DS2 package method or C function.

Return
The description of the step.

getStepInputs

```
java.util.ArrayList<tksfParmdef> getStepInputs
(long moduleContext, int revision, java.lang.String stepId)
```

Retrieves the step input arguments.

Parameters
moduleContext - The opaque handle returned from newModuleContext().

revision - The revision number returned from newRevision().

stepId - The name of the DS2 package method or C function.

Return
An ArrayList of parameter definitions. For more information, see "Parameter Descriptions" on page 32.

getStepInputs

```
java.util.ArrayList<tksfParmdef> getStepInputs
(long moduleContext, int revision, java.lang.String stepId, int index)
```

Retrieves step input arguments for an overloaded method, at a zero-based index.

Note: This method is valid only for DS2, which supports method overloading.

Note: Overloaded methods are methods within a DS2 package that have the same name.

Parameters
moduleContext - The opaque handle returned from newModuleContext().

revision - The revision number returned from newRevision().

stepId - The name of the DS2 package method or C function.

index - The zero-based index of the overloaded method.

Return
An ArrayList of parameter definitions. For more information, see "Parameter Descriptions" on page 32.

getStepOutputs

```
java.util.ArrayList<tksfParmdef> getStepOutputs
(long moduleContext, int revision, java.lang.String stepId)
```

Retrieves the step output arguments.

Parameters
moduleContext - The opaque handle returned from newModuleContext().

revision - The revision number returned from newRevision().

stepId - The name of the DS2 package method or C function.

Return
An ArrayList of parameter definitions. For more information, see "Parameter Descriptions" on page 32.

getStepOutputs

```
java.util.ArrayList<tksfParmdef> getStepOutputs
(long moduleContext, int revision, java.lang.String stepId, int index)
```

Retrieves the step output arguments for an overloaded method, at a zero-based index.

Note: This method is valid only for DS2, which supports method overloading.

Note: Overloaded methods are methods within a DS2 package that have the same name.

Parameters
moduleContext - The opaque handle returned from newModuleContext().

revision - The revision number returned from newRevision().

stepId - The name of the DS2 package method or C function.

index – The zero-based index of the overloaded method.

Return
An ArrayList of parameter definitions. For more information, see "Parameter Descriptions" on page 32.

getOverloadedStepCount

```
int getOverloadedStepCount(long moduleContext,
                           int revision,
                           java.lang.String stepID)
```

Retrieves the number of overloaded steps matching the given step ID.

Note: This method is valid only for DS2, which supports method overloading.

Note: Overloaded methods are methods within a DS2 package that have the same name.

Parameters

moduleContext - The opaque handle returned from newModuleContext().

revision - The revision number returned from newRevision().

stepId - The name of the DS2 package method or C function.

Return

The number of overloaded steps (DS2 package methods having the same name).

isOverloaded

```
boolean isOverloaded(long moduleContext,
                     int revision,
                     java.lang.String stepID)
```

Retrieves the number of overloaded steps matching the given step ID.

Note: This method is valid only for DS2, which supports method overloading.

Note: Overloaded methods are methods within a DS2 package that have the same name.

Parameters

moduleContext - The opaque handle returned from newModuleContext().

revision - The revision number returned from newRevision().

stepId - The name of the DS2 package method or C function.

Return

True, if the method is overloaded, False otherwise.

getRevisionCreationDateTime

```
java.util.Date getRevisionCreationDateTime
(long moduleContext, int revision)
```

Gets the revision creation datetime.

Parameters

moduleContext - The opaque handle returned from newModuleContext().

revision - The revision number returned from newRevision().

Return

The creation datetime in SAS form (GMT seconds since 1960).

Execution Methods

Overview

As previously discussed, code is published to SAS Micro Analytic Service by calling newRevision(), which returns a revision number greater than zero upon successful compilation. When a revision number less than or equal to zero is returned, a revision was not created. In this case, messages output by the compiler during the failed compilation attempt can be retrieved by calling getCompilerMessages() with the module context ID.

Note: Compiler messages from the most recent call to newRevision() are maintained by the module context. Therefore, if newRevision() fails to create a revision, the compiler message can still be retrieved. Each call to newRevision() overwrites any previously saved compiler messages.

When a revision has been created, its methods can be executed repeatedly. To execute a method, call execute(), passing in the module context ID, revision number (or zero for latest revision), method name, and method arguments.

Note: The method name is also referred to as an entry point.

```
rc = tk.execute(userCtx, moduleCtx, revisionNumber, "myMethod", args);
```

Revision numbers are assigned by SAS Micro Analytic Service. The first revision number is always 1. Subsequent revisions are assigned numbers 2, 3, 4, and so on. Revision numbers are never reused. Therefore, if four revisions of a module have been created and revision 3 is deleted, the next revision number assigned is 5 (3 is never reused). Specifying zero for revision number gets the latest revision.

Note: If you know the number of the revision that you want to execute, always pass that number to execute() rather than zero. Passing in zero specifies the latest revision, and causes SAS Micro Analytic Service to look up the latest revision number. This lookup takes time and can be bypassed altogether by passing in the explicit, nonzero revision number.

Java Data Types

Here are the Java scalar data types supported by SAS Micro Analytic Service:

- String
- Character
- Long
- Integer
- Double

The following Java array types are supported:

- String[]
- long[]
- int[]
- double[]

Method Arguments

Method arguments, including input and output parameters, are passed to SAS Micro Analytic Service as a com.sas.mas.tksfValues Java object.

```
tksfValues p = new tksfValues(numArgsTotal, numInputs);
```

The first parameter to the tksfValues constructor is the total number of arguments, including both inputs and outputs. The second parameter specifies the number of input arguments of the method.

Note: Before calling execute(), all method arguments, including outputs, must be set on the tksfValues instance. Output arguments are set by calling the setOut<type> methods of tksfValues.

Output arguments must be set for two reasons:

- DS2 passes output arguments by reference. Therefore, the correct types must be allocated in order to receive the output values. Because DS2 follows this pass-by-reference convention, SAS Micro Analytic Service follows the same convention regardless of the programming language.
- SAS Micro Analytic Service supports method overloading. When overloaded methods are used, the method signature (list of input and output parameters and their types) is used to select the correct method to execute. Because a method signature includes both input and output parameters, the output parameter types must be set in tksfValues.

In DS2, when two or more methods in the same package have the same name, those methods are said to be overloaded. Each module constitutes a separate namespace. Therefore, two DS2 methods that have the same name, but are in different packages, are not considered overloaded. Similarly, two C functions with the same name, but in different C modules, will not have a name conflict.

Note: The C language does not support method overloading. Syntax errors occur if two C functions with the same name exist in the source code of the same C module.

DS2 in_out parameters are treated strictly as output. This convention is necessary to support the REST interface, which does not support true in/out arguments.

When you are coding in DS2 or C, all input arguments must be positioned before any output arguments. Failure to do so causes your method to malfunction. SAS Micro Analytic Service might not detect this error because the DS2 compiler does not carry this restriction.

Note: Arrays are always passed by reference in DS2, regardless of whether the in_out keyword is specified. However, SAS Micro Analytic Service treats arrays without the in_out keyword as input and arrays with the in_out keyword as output, as long as all inputs precede outputs in the method signature.

Incorrect:

```
method solve(double tempRate, in_out int finalRate, int custId);
```

Correct:

```
method solve(double tempRate, int custId, in_out int finalRate);
```

tksfValues provides methods for setting and getting arguments, including methods that enable the use of explicit or implicit argument indices and methods that support missing values. These methods are described in the sections that follow.

Argument Setter Methods

Overview

Use the setter methods to prepare input and output arguments before calling execute(). Output variables, of the correct types, must be set before calling execute(). Here are the reasons:

- Output variables are passed by reference. Therefore, variables must exist to receive the output values.
- DS2 supports method overloading (two or more methods having the same name but different signatures). The entire signature must be reflected in the arguments list to allow selection of the correct same-named method.
- TKG does not supply metadata for C functions. Therefore, the data type must be set for every argument, including the output arguments.

Setters with an Implicit Index

Use the tksfValues methods below to set the values of input arguments.

Arguments are positional and referenced by a zero-based index. These methods use implied indexes. The first setter method call uses index 0, the next index 1, and so on.

```
public void setString(String value);
public void setChar(Character value);
public void setLong(Long value);
public void setInt(Integer value);
public void setDouble(Double value);
public void setStringArray(String[] value);
public void setLongArray(long[] value);
public void setIntArray(int[] value);
public void setDoubleArray(double[] value);
```

Setters with an Explicit Index

Use the tksfValues methods below to set the values of input arguments.

These methods use explicit indexes and set the value of the input argument at the zero-based index position that is given.

```
public void setString(int index, String value);
public void setChar(int index, Character value);
public void setLong(int index, Long value);
public void setInt(int index, Integer value);
public void setDouble(int index, Double value);
public void setStringArray(int index, String[] value);
public void setLongArray(int index, long[] value);
public void setIntArray(int index, int[] value);
public void setDoubleArray(int index, double[] value);
```

Missing Value Setters with an Implicit Index

Use the tksfValues methods below to set the values of input arguments to missing.

Arguments are positional and referenced by a zero-based index. These methods use implied indexes. The first setter method call uses index 0, the next index 1, and so on.

The argument to the array setters (dim) specifies the size of the one-dimensional array. Each element of the array is set to a missing value.

```
public void setMissingString();
public void setMissingChar();
public void setMissingLong();
public void setMissingInt();
public void setMissingDouble();
public void setMissingStringArray(int dim);
public void setMissingLongArray(int dim);
public void setMissingIntArray(int dim);
public void setMissingDoubleArray(int dim);
```

Missing Value Setters with an Explicit Index

Use the tksfValues methods below to set the values of input arguments to missing.

These methods use explicit indexes, and set the value of the input argument at the zero-based index position given.

The argument to the array setters (dim) specifies the size of the one-dimensional array. Each element of the array is set to a missing value.

```
public void setMissingString(int index);
public void setMissingChar(int index);
public void setMissingLong(int index);
public void setMissingInt(int index);
public void setMissingDouble(int index);
public void setMissingStringArray(int index, int dim);
public void setMissingLongArray(int index, int dim);
public void setMissingIntArray(int index, int dim);
public void setMissingDoubleArray(int index, int dim);
```

Output Setters with an Implicit Index

Use the tksfValues methods below to set the type of an output argument (and to set its value to missing) before calling execute().

Arguments are positional and referenced by a zero-based index. These methods use implied indexes. The first setter method call uses index 0, the next index 1, and so on.

The argument to the array setters (dim) specifies the size of the one-dimensional array. Each element of the array is set to a missing value.

```
public void setOutString();
public void setOutChar();
public void setOutLong();
public void setOutInt();
public void setOutDouble();
public void setOutStringArray(int dim);
public void setOutLongArray(int dim);
public void setOutIntArray(int dim);
public void setOutDoubleArray(int dim);
```

Output Setters with an Explicit Index

Use the tksfValues methods below to set the type of an output argument, and to set its value to missing, before calling execute().

These methods use explicit indexes, and set the value of the input argument at the zero-based index position given.

The argument to the array setters (dim) specifies the size of the one-dimensional array. Each element of the array is set to a missing value.

```
public void setOutString(int index);
public void setOutChar(int index);
public void setOutLong(int index);
public void setOutInt(int index);
public void setOutDouble(int index);
public void setOutStringArray(int index, int dim);
public void setOutLongArray(int index, int dim);
public void setOutIntArray(int index, int dim);
public void setOutDoubleArray(int index, int dim);
```

Argument Getter Methods

Overview

Use the getter methods to retrieve results after calling execute(). A return code of zero from execute() indicates successful execution.

tksfValues provides methods for retrieving method output values, and for checking to see whether an output argument has been set to missing.

Similar to the getter methods, setter methods use either implicit or explicit indices.

Missing Value Check

The following method can be used to see whether a given output argument contains a missing value. isMissing() checks the argument at the given zero-based position for missing, regardless of the arguments data type.

```
public boolean isMissing(int ndx);
```

Getters with an Implicit Index

Use the tksfValues methods below to get the values of output arguments.

Note: The first output argument often follows one or more input arguments. Therefore, in order to use implicit indices to retrieve output values, it is often necessary to call setIndex() before calling the first getter method. setIndex() sets the implicit zero-based index to the given value, enabling the first getter method (with implicit index) to retrieve the first output argument.

```
public void setIndex(int ndx);
```

Arguments are positional and referenced by a zero-based index. These methods use implied indexes. The first getter method call uses index 0, the next index 1, and so on.

```
public String getString();
public Character getChar();
public Long getLong();
public Integer getInt();
public Double getDouble();
public String[] getStringArray();
public long[] getLongArray();
public int[] getIntArray();
public double[] getDoubleArray();
```

Getters with an Explicit Index

Use the tksfValues methods below to get the values of output arguments.

These methods use explicit indexes and get the value of the output argument at the zero-based index position given.

```
public String getString(int pos);
public Character getChar(int pos);
public Long getLong(int pos);
public Integer getInt(int pos);
public Double getDouble(int pos);
public String[] getStringArray(int pos);
public long[] getLongArray(int pos);
public int[] getIntArray(int pos);
public double[] getDoubleArray(int pos);
```

Miscellaneous tksfValues Methods

Use clear() to reset all argument values in the tksfValues instance and to set the implicit index to zero.

```
public void clear();
```

Use the following methods to retrieve the total number of arguments and the number of input arguments, respectively, of the tksfValues instance:

```
public int getSize();
    public int getInputCount();
```

Execute Method

The following example executes a method (or step). The results are returned in output arguments. The return code indicates the success or failure of step execution.

```
int execute(long userContext,
                long moduleContext,
                int revision,
                java.lang.String entryPoint,
                tksfValues arguments)
```

Here are the parameters:

userContext
: The opaque handle returned from newUserContext().

moduleContext
: The opaque handle returned from newModuleContext().

revision
: The revision number returned from newRevision().

entryPoint
: The name of the DS2 package method or C function to execute.

arguments
: The tksfValues object containing input arguments and placeholders for output arguments.

Revision Monitoring Methods

Here are the revision monitoring methods:

getRevisionHitCount

```
long getRevisionHitCount(long moduleContext,
                                        int revision)
```

Returns the aggregate total number of times any method of the revision has been called since the revision was created.

Parameters

moduleContext - The opaque pointer returned from newModuleContext().

revision - The revision number returned from newRevision().

Return

The number of times the revision methods have been called.

getRevisionAvgLatency

```
double getRevisionAvgLatency(long moduleContext,
                                                int revision)
```

Returns the average execution latency, in seconds, of method calls on the specified revision.

Parameters

moduleContext - The opaque pointer returned from newModuleContext().

revision - The revision number returned from newRevision().

Return

The average length of time, in seconds, spent in method execution. The average includes all methods of the specified revision.

getTotalRevisionExecutionTimeSeconds

```
double getTotalRevisionExecutionTimeSeconds(long moduleContext,
                                                                    int revision)
```

Returns the total time, in seconds, spent executing methods of the specified revision.

Parameters

moduleContext - The opaque pointer returned from newModuleContext().

revision - The revision number returned from newRevision().

Return

The total length of time, in seconds, spent in method execution. The total includes all methods of the specified revision.

Complete Java Example

This section examines a simple but complete Java program that uses SAS Micro Analytic Service to call a simple DS2 package method. Comments within the body of the code explain each step.

The example code performs the following sequence of steps:

1. Starts SAS Micro Analytic Service
2. Creates a user context
3. Creates a module context
4. Creates a new revision (DS2 package is compiled)
5. Checks for compiler messages and prints any that are found to the console
6. Retrieves and prints metadata about the DS2 package
7. Prepares method arguments
8. Calls a method
9. Retrieves and prints results
10. Prepares a different set of argument values
11. Calls a different method of the package
12. Retrieves and prints results of the second call
13. Shuts down SAS Micro Analytic Service

The console output from running the example follows the source code.

```
package com.sas.mas.test;

import java.util.ArrayList;

import com.sas.mas.TkLight;
import com.sas.mas.tksfParmdef;
import com.sas.mas.tksfValues;
import com.sas.mas.TkLight.Language;
import com.sas.mas.jni.tksfjni;

public class SimpleDS2Example {

    /*
       This is a simple DS2 package with two methods.  The code could have
       been read from a file, but is included here for easy reference.

       The source code starts with "ds2_options sas" and ends with
       with "endpackage."  This pattern should be used with all DS2 to be
       published to SAS Micro Analytic Service.

       Note: each source code line ends with a line-end character.
       This best practice facilitates use of the line numbers included in
       compiler messages, making it easier to locate syntax errors.
    */
    static String DS2 =
"ds2_options sas;                                                          \n" +
"package simple_example /overwrite=yes;                                    \n" +
"                                                                          \n" +
"   method str2double (char(12) numericString, in_out double number);      \n" +
"      number = put( numericString, 8.0);                                  \n" +
"      /* Include an undeclared variable to illustrate a compiler warning */ \n" +
"      anotherNumber = number;                                             \n" +
"   end;                                                                   \n" +
"                                                                          \n" +
```

```
"   method flip_string(varchar(32767) in_string, in_out varchar out_string); \n" +
"      /* Reverse the input string and set in output string */               \n" +
"      out_string=reverse(in_string);                                        \n" +
"   end;                                                                     \n" +
"                                                                            \n" +
"endpackage;                                                                 \n";

    /**
     * @param parms
     */
    public static void main(String[] parms) {
        int         rc;
        String   stringToReverse    = new String("This is a test...");
        String   stringToConvert    = new String("0.9997");
        long     userCtx            = -1;
        long     moduleCtx            = -1;

        // Start SAS Micro Analtyic Service with four threads and
        // no logging configuration file location (null second argument)
        int threads = 4;
        TkLight tk = new tksfjni(threads, null);

        System.out.println("*** Simple example of using SAS Micro Analytic Service ***");

        // Create a user context.
        userCtx = tk.newUserContext("A user context");
        if (userCtx <= 0) {
            System.out.println("   User context creation failed.");
            return;
        }
        else {
            System.out.println("   User context created at " +
                    tk.getUserContextCreationDateTime(userCtx) + ".");
        }

        /* Create a module context.
            This module context is owned by the user context just created.
            The language is specified as DS2. Therefore, all revisions
         of the module must be DS2.
            The module is named for the DS2 package it represents, or
             "simple_example" in this case.
            Pass null for the last argument, which specifies the default options.
         */
        moduleCtx = tk.newModuleContext(userCtx,
                                        Language.DS2,
                                        "simple_example",
                                        null);
        if (moduleCtx <= 0) {
            System.out.println("   Module context creation failed.");
            tk.term();
            return;
        }
        else {
            System.out.println("   Module context created at " +
                    tk.getModuleContextCreationDateTime(moduleCtx) + ".");
        }
```

```
        /* Publish the DS2 to SAS Micro Analytic Service by calling newRevision().
            Pass in the module context ID, the source code String, and an
            optional description.
            Pass null for the fouth argument, a list of entry points, which
            is only used for C modules.
            Pass null for the last argument, which specifies default options.
         */
        int revision = tk.newRevision(moduleCtx, DS2,
                "A simple DS2 example to illustrate SAS Micro Analytic Service usage.",
                null, null);
        if (revision <= 0) {
            System.out.println("   Compilation failed");
            String[] messages = tk.getCompilationMessages(moduleCtx);
            if (messages.length > 0) {
                System.out.println("   Compiler messages:");
                for (String msg : messages) {
                    System.out.println("         " + msg);
                }
            }
            System.out.println("   Revision creation failed.");
            tk.term();
            return;
        }
        else {
            System.out.println("   Revision " + revision + " created at " +
                        tk.getRevisionCreationDateTime(moduleCtx, revision)
+ ".");
        }

        System.out.println("   DS2 Package: " +
                           tk.getModuleContextDisplayName(moduleCtx));

        /* Even successful compilations can produce warning messages from the compiler.
           The undeclared variable included in the source code generates such a
           warning. However, it will not prevent the code from publishing and executing
           properly.
         */
        String[] messages = tk.getCompilationMessages(moduleCtx);
        if (messages.length > 0) {
            System.out.println("      Compiler messages:");
            for (String msg : messages) {
                System.out.println("         " + msg);
            }
        }

        /* Once published, SAS Micro Analtic Service can be queried for information
        about the revision. Here you are asked for the inputs to the str2double
        method. Parameter metadata is represented by class tksfParmdef.
         */
        ArrayList<tksfParmdef> inputs = tk.getStepInputs(moduleCtx,
                                                         revision,
                                                         "str2double");
        System.out.println("   Input arguments to method 'str2double':");
        for (tksfParmdef p : inputs) {
            System.out.println("   Input name: '" + p.name + "' type: " +
```

```
p.getType());
  }

  /* Setup arguments to call method "str2double", which has one String input and
     one double output.
  Note: Setters are being used with implicit indices.
   */
  int        numArgs   = 2;
  int        numInputs = 1;
  tksfValues args = new tksfValues(numArgs, numInputs); // DS2 method arguments
  args.setString(stringToConvert);
  args.setOutDouble();
  String methodName = "str2double";

  // Execute the DS2 package method "str2double"
  rc = tk.execute(userCtx, moduleCtx, revision, methodName, args);
  if (rc != 0) {
      System.out.println("   Bad return code from execute:" + rc);
      if (rc == 29) {
          System.out.println("   Exception occurred during execution of " +
                        methodName + " in the TK environment.");
      }
      tk.term();
      return;
  }
  else {
      // Print results (Getters are being used with explicit indices.)
      System.out.println("   Results of calling str2double:");
      System.out.println("      Input String:            " + args.getString(0));
      System.out.println("      Output double (rounded): " + args.getDouble(1));
  }

  // Setup arguments to call the method "flip_string", which has one String input
  // and one double output. Setters are being used with implicit indices.
  numArgs   = 2;
  numInputs = 1;
  args = new tksfValues(numArgs, numInputs);        // DS2 method arguments
  args.setString(stringToReverse);
  args.setOutString();
  methodName = "flip_string";

  // Execute the DS2 package method "flip_string"
  rc = tk.execute(userCtx, moduleCtx, revision, methodName, args);
  if (rc != 0) {
      System.out.println("   Bad return code from execute:" + rc);
      if (rc == 29) {
          System.out.println("   Exception occurred during execution of " +
                        methodName + " in the TK environment.");
      }
      tk.term();
      return;
  }
  else {
      // Print results (Getters are being used with explicit indices.)
      System.out.println("   Results of calling flip_string:");
      System.out.println("      Input String:  " + args.getString(0));
```

```
            System.out.println("      Output String: " + args.getString(1));
        }
        System.out.println("*** Simple DS2 example complete ***\n");
        // Shutdown
        tk.term();
    }
}
```

Here is the console output from running the example code above:

```
*** Simple example of using SAS Micro Analytic Service ***
   User context created at Tue Apr 07 17:50:28 EDT 2015.
   Module context created at Tue Apr 07 17:50:28 EDT 2015.
   Revision 1 created at Tue Apr 07 17:50:28 EDT 2015.
   DS2 Package: simple_example
      Compiler messages:
         Line 6: No DECLARE for assigned-to variable anothernumber;
         creating it as a global variable of type double.
   Input arguments to method 'str2double':
   Input name: 'numericString' type: string_t
   Results of calling str2double:
      Input String:                                          0.9997
      Output double (should round to nearest whole number): 1.0
   Results of calling flip_string:
      Input String:  This is a test...
      Output String: ...tset a si sihT
*** Simple DS2 example complete ***
```

Chapter 6

SAS Micro Analytic Service REST API

Overview

The SAS Micro Analytic Service REST API provides an interface for web client applications to compile and execute micro analytic modules into steps that provide near real-time analytic capabilities. The REST API supports the execution of DS2 source and provides the ability to run SAS Enterprise Miner score code (converted from a SAS DATA step to DS2) and user-written functions.

The API provides the following POST methods:

Create module
: publishes analytic code in memory with a request body containing the DS2 source code as input.

Validate steps
: validates the request body of input values required by the DS2 source code and returns validation results.

Execute step
: validates and executes the micro analytic step with a request body of input values required by the DS2 source code.

The API provides the following PUT method:

Update module
: publishes updated analytic code in memory with a request body containing the DS2 source code as input.

The API provides the following DELETE method:

Delete module
: removes analytic code from memory.

The API provides the following GET methods:

Query an individual module
: returns detailed information about a module

Query steps by module
: returns a list of steps available by module.

Query an individual step by module
: returns detailed information about the inputs required by the step and the outputs produced by the step.

Retrieve module details
: returns information such as the module's name, current revision, and a list of compiled steps.

The implementation supports only JSON resource representations.

Note: The REST API does not support method overloading.

Terminology

Micro Analytic Service

A small footprint, near real-time or machine-embedded, analytical service providing the ability to embed SAS analytics into very small portable systems requiring near real-time or transactional analytics.

Micro Analytic Module

A collection item that contains multiple steps of analytical logic. The SAS Micro Analytic Service REST API representation of a collection of units of step code to execute analytical logic.

Micro Analytic Step

A unit of analytical logic that is executed. It includes input and output values. Here is an example: the name value pairs that contain the input values required to execute the step and the output values that are generated as a result of its execution. For DS2 source, a step is defined as a method. When the step is executed, a specific method in the module is executed.

Package

An assembly of methods defined by a DS2 source.

Method

A unit of DS2 source that has input and output variables.

Signature

Variables defined as inputs into a method and outputs from the execution of a method.

Input Signature

A description of the input values required to execute the step. The attributes of the input signature include the input variable, its data type, and the dimensions where applicable.

Output Signature

A description of the output values. Here is an example: the name value pairs that describe the name of the output variable, its data type, and the dimensions where applicable.

Module

A container of units of analytical code to be executed. For a DS2 source, it is a package.

Module ID

A generated unique string that identifies a module in an installation. When the installation is a cluster, no two modules created on two different cluster nodes have the same ID.

Module Name

A name associated with a module. For a DS2 source, this corresponds to the package name. A DS2 package name can be quoted. Because of that, it is not convenient to use it on the URL to specify the module for an HTTP operation. Even though the module name is not used to identify a module, each module name has to be unique in an installation.

Step

A unit of analytical code to be executed. For a DS2 source, it is a method.

Step ID

The name of a step that is included in the micro analytic module. For a DS2 source, this corresponds to the name of a method. The combination of module ID and step ID is used to retrieve the individual step.

Source Code

The input analytic source code that is compiled into a micro analytic module containing one or more steps.

Client Application Features

Post Load or Create Modules

To load or create a micro analytic module, the client application posts a module with a request body that contains the DS2 source code to the module's resource collection.

The DS2 source code is represented as a source code representation that compiles into one DS2 package. The package is represented as a micro analytic module with multiple methods that are represented as steps in the REST API. Therefore, a module contains multiple steps. These modules and steps are stored in memory. The response body that is returned contains a module resource for the module.

Get Input or Output Step Signatures

The client application references a step directly by using an ID of the module generated by the REST server. This ID is referred to as the module ID, and the name of the step (compiled DS2 method) is referred to as the step ID.

Before executing the step, the client application performs a GET method on the step to retrieve these signatures:

- The signature describing the input variables or types that must be put in the request body to execute the step.
- The signature describing the output variables or types that the step returns in its response body.

Post Validate Input Variables

The client application posts to the step's validations resource, along with a request body that contains the input values that are required to execute the step (compiled DS2 method).

When the POST is received, the input values are validated against the input signature of the step. A validation error is reported to the client as a response body that contains the validation results. This allows the client to validate its input before execution.

Post Execute Modules

The service supports a synchronous way to execute a step (compiled DS2 method). In this case, the client application posts to the step resource, along with a request body that contains the input values that are required to execute the step (compiled DS2 method).

Put Update Modules

The client application creates a new revision of a module through its module ID.

Delete Modules

The client application deletes a module through its module ID.

Security and Authentication

To reduce Cross Site Request Forgery (CSRF) attack, a filter is used to check whether the HTTP referrer header value of an incoming request is registered in the white list that is set up during product configuration. A referrer identifies the page that caused the incoming request to be sent. If the referrer header is used but the referring address does not match any of the patterns allowed in the white list, the request is rejected with an HTTP 403 error. For more information, see *SAS 9.4 Intelligence Platform Middle-Tier Administration Guide*.

Note: If you encounter white list issues, from SAS Management Console navigate to **Application Management** ⇨ **SAS Application Infrastructure**, and then right-click and select **Properties**. On the **Advanced** tab, add trusted hosts to the white list. For example, the value *.example.com added to the white list allows requests originating from the example.com domain to get through.

The creation and execution of the analytical logic are tasks controlled through security. In an enterprise application, the API uses authentication supported by the SAS platform to create tickets and use them with the API. The API internally processes user roles and authorization and returns a status of 401 if the operation is not allowed for a particular user. However, it will not specify implementation or representation.

All modules are discoverable and usable by an authenticated user.

Life Cycle

A compiled micro analytic module remains compiled during the lifetime of the server session in which it was compiled, even when dependent modules are updated afterward.

The REST server manages the persistence of the modules by keeping metadata about the modules. Therefore, when the REST server restarts, there is enough information to recreate the existing modules. The module IDs remain the same. However, when the modules are loaded into memory again they can be put in addresses that are different from the last time. Furthermore, each reload of the modules requires them to be recompiled.

The compilation of the modules is delayed until necessary (for example, when a module is to be executed).

Media Types

Externally Defined Media Types

application/vnd.sas.collection

The application/vnd.sas.collection media type represents a collection of resources. The collection is usually a page of limit items from a larger collection.

Here are the link relations for the application/vnd.sas.collection media type.

Relationship	HTTP Method	Description
self	GET	The current page of the (filtered) collection. URI: {SASApi}/rest/collectionUri?start=startIndex&limit=limitIndex Media type: application/vnd.sas.collection
next	GET	The next page of resources. It should be omitted if the current view is on the last page. URI: {SASApi}/rest/collectionUri?start=startIndex&limit=limitIndex Media type: application/vnd.sas.collection
first	GET	The first page of resources. It should be omitted if the current view is on the first page. URI: {SASApi}/rest/collectionUri?start=startIndex&limit=limitIndex Media type: application/vnd.sas.collection

Relationship	HTTP Method	Description
last	GET	The last page of resources. It should be omitted if the current view is on the last page. URI: {SASApi}/rest/collectionUri?start=startIndex&limit=limitIndex[modifiers] Media type: application/vnd.sas.collection
up	GET	The resource that this collection resides in. URI: {SASApi}/rest/containerUri Media type: application/vnd.sas.collection

Here is an example of application/vnd.sas.collection+json and application/vnd.sas.collection+json;version=2:

```
{
  "version" : 2,
  "accept": "space-separated media type names allowed in this collection",
  "count" : integer,
  "start" : integer,
  "limit" : integer,
  "name" : "items",
  "items": [
    { resource1 fields }, ...,
    { resourceN fields }
    ],
  "links" : [
          { link representation }, ...
          { link representation },
          ]
}
```

Note: The order of the fields can vary.

application/vnd.sas.error

Here are attributes for application/vnd.sas.error:

errorCode
: The system error code for reference (64-bit integer). It is often used for correlation with back-end service error message identifiers.

httpStatusCode
: The HTTP status code error number (integer), 1xx, 2xx, 3xx, 4xx, or 5xx values.

message
: The back-end system error message string. The message should be localized as per the Accept-Language of the request.

details
: Detailed information specific to this error, in a list of strings. If appropriate, these strings should be localized as per the Accept-Language of the request.

remediation
: Recommended actions to resolve the error, in a list of strings. The remediation string should be localized as per the Accept-Language of the request.

version
Version information for this error format (integer, value 1).

links
An array of application/vnd.sas.link objects.

application/vnd.sas.link

application/vnd.sas.link is a media type used to denote a link to a resource.

text/vnd.sas.source.ds2

text/vnd.sas.source.ds2 is a media type used to denote SAS source code consisting of DS2 code.

SAS Micro Analytic Service Media Types

application/vnd.sas.microanalytic.module

The application/vnd.sas.microanalytic.module media type describes the module that is returned by the SAS Micro Analytic Service when source code is posted or put to the module's collection.

Here are the link relations for the application/vnd.sas.microanalytic.module media type.

Relationship	HTTP Method	Description
self	GET	A link to the individual module. URI: SASMicroAnalyticService/rest/modules/{*moduleId*} Media type: application/vnd.sas.microanalytic.module
up	GET	A link to the collection of modules. URI: SASMicroAnalyticService/rest/modules Media type: application/vnd.sas.collection
steps	GET	A link to the collection of steps. This is created when a module is compiled. URI: SASMicroAnalyticService/rest/modules/{*moduleId*}/steps Media type: application/vnd.sas.collection
source	GET	A link to the source code that was used to compile a module. URI: SASMicroAnalyticService/rest/modules/{*moduleId*}/source Media type: application/vnd.sas.microanalytic.module.source

Relationship	HTTP Method	Description
update	PUT	A link to update a module. URI: SASMicroAnalyticService/rest/modules/{*moduleId*} Media type: application/vnd.sas.microanalytic.module
delete	DELETE	A link to remove a module. URI: SASMicroAnalyticService/rest/modules/{*moduleId*}

The application/vnd.sas.microanalytic.module media type contains the following members.

Name	Type	Description
version	integer	The media type's schema version number. This representation is version 1.
id	string	A generated unique string identifying a module in an installation.
description	string	Text describing the rules and logic performed by the module. The description is specified in the POST or PUT request body and carried over.
name	string	The name associated with the module.
creationTimeStamp	string	The formatted time stamp that tells when the module was initially created.
modifiedTimeStamp	string	The formatted time stamp that tells when the module was last revised.
revision	integer	The revision number of the module. It is a whole number starting from one and increases by one each time the module is revised.
scope	string (ENUM)	The scope restricts how a module can be used. There are two possible values: public The module is available to be called outside another module. private The module can be called only from within another module.
steps	array of string	An array of step IDs in the module.

Name	Type	Description
properties	array	The properties that were specified for the module. Here are the representation members: name string - The name of the property. value string - The value of this property.
warnings	object	Optional object, as described in "application/vnd.sas.error" on page 55. This is included if the compiling of this resource produces any warning.
links	array of link objects	Zero or more link objects. See the table above for a description of the link types.

Here is an example of application/vnd.sas.microanalytic.module+json:


```
{
   "links":[
      {
         "method":"GET",
         "rel":"self",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 359fb21e-c65d-4b8d-81e0-216d95cb0825",
         "uri":"/modules/359fb21e-c65d-4b8d-81e0-216d95cb0825",
         "type":"application/vnd.sas.microanalytic.module"
      },
      {
         "method":"GET",
         "rel":"up",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules",
         "uri":"/modules",
         "type":"application/vnd.sas.collection"
      },
      {
         "method":"GET",
         "rel":"source",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 359fb21e-c65d-4b8d-81e0-216d95cb0825/source",
         "uri":"/modules/359fb21e-c65d-4b8d-81e0-216d95cb0825/source",
         "type":"application/vnd.sas.microanalytic.module.source"
      },
      {
         "method":"GET",
         "rel":"steps",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 359fb21e-c65d-4b8d-81e0-216d95cb0825/steps",
         "uri":"/modules/359fb21e-c65d-4b8d-81e0-216d95cb0825/steps",
         "type":"application/vnd.sas.collection"
      },
      {
         "method":"PUT",
         "rel":"update",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
```

```
                359fb21e-c65d-4b8d-81e0-216d95cb0825",
         "uri":"/modules/359fb21e-c65d-4b8d-81e0-216d95cb0825",
         "type":"application/vnd.sas.microanalytic.module"
      },
      {
         "method":"DELETE",
         "rel":"delete",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                359fb21e-c65d-4b8d-81e0-216d95cb0825",
         "uri":"/modules/359fb21e-c65d-4b8d-81e0-216d95cb0825"
      }
   ],
   "scope":"public",
   "description":"575",
   "id":"359fb21e-c65d-4b8d-81e0-216d95cb0825",
   "steps":[
      "execute",
      "executeFinalRuleSets",
      "executeFirstDotRuleSets",
      "executeInitRuleSets",
      "executeLastDotRuleSets",
      "initRuleFiredRecording",
      "initializeLookupHash",
      "recordRuleFired",
      "resetRuleFiredHash",
      "term"
   ],
   "properties":[
      {
         "name":"connectionString",
         "value":"DRIVER=base;"
      }
   ],
   "revision":1,
   "creationTimeStamp":"2015-04-16T16:05:38.000-0400",
   "modifiedTimeStamp":"2015-04-16T16:05:38.000-0400",
   "name":"Rule575",
   "version":1
}
```


application/vnd.sas.microanalytic.module.definition

The application/vnd.sas.microanalytic.module.definition media type describes the resource that is used to define a revision of the SAS Micro Analytic Service module in the module's collection. It is used in the request body of POST and PUT in the module's collection.

The application/vnd.sas.microanalytic.module.definition media type contains the following members.

Name	Type	Description
version	integer	This media type's schema version number. This representation is version 1.

Name	Type	Description
description	string	The text describing the logic of the module.
code	string	The source code. (For example, DS2 source code)
type	string	The source code type. In this version, the only valid value is text/vnd.sas.source.ds2.
properties	array	This can be used to define connection strings. If a property definition is not needed, this can be omitted or specified as an empty array. Here are the representation members: name string - The name of the property. It cannot contain spaces and must be unique. Version 1 allows only connectionString as the value. value string - The value of this property.
scope	string (ENUM)	The scope restricts how a module can be used. There are two possible values: public The module is available to be called outside another module. private The module can be called only from within another module.

Here is an example of application/vnd.sas.microanalytic.module.definition+json:

```
{
"version": "1",
"description": "Sample module",
"scope" : "public",
"type" : "text/vnd.sas.source.ds2",
"properties" : [],
"code" : "ds2_options sas;\n package sampleModule / overwrite=yes;
          \n \n method copy_charN_array(char(12) in_array[4], in_out char(12)
          out_array[4]);\n out_array := in_array;\n end;\n \n
          method copy_varchar_array(varchar(512) in_array[3],
          in_out varchar out_array[3]);\n out_array := in_array;\n end;\n \n
          method copy_int_array(int in_array[5], in_out int out_array[5]);\n
          out_array := in_array;\n end;\n \n method copy_float_array(double in_array[2],
          in_out double out_array[2]);\n out_array := in_array;\n end;\n \n
          method copy_bigint_array(bigint in_array[1], in_out bigint out_array[1]);\n
          out_array := in_array;\n end;\n \n method copy_arrays( char(12)
          in_charN_array[4],\n varchar(512) in_varchar_array[1],\n int in_int_array[5],
          \n double in_double_array[2], \n bigint in_bigint_array[1], \n
          in_out char(12)  out_charN_array[4],\n in_out varchar(512)
          out_varchar_array[1],\n in_out int out_int_array[5],\n
          in_out double out_double_array[2],\n in_out bigint out_bigint_array[1]);\n \n
          copy_charN_array(in_charN_array, out_charN_array);\n copy_int_array(in_int_array,
          out_int_array);\n copy_float_array(in_double_array, out_double_array);\n
          copy_bigint_array(in_bigint_array, out_bigint_array);\n \n end;\n \n
```

```
        endpackage;\n \n \n"
}
```

Note: There are many \n strings throughout the source code. They help to signal line breaks to the DS2 compiler. Line breaks are useful because, in JSON representation, the entire source code must be presented as one long string and the \n returns the line breaks to you. If there are errors, the compiler messages will not all refer to line 1. If your platform is UNIX or Linux, you can use the sed command to convert \n into a real line break character. Here is the pattern for the sed command: **`-e "s#\\n# \n#g"`**.

application/vnd.sas.microanalytic.module.source

The application/vnd.sas.microanalytic.module.source media type describes the source code resource that is created by the SAS Micro Analytic Service when a POST or PUT is performed on the module's collection.

Here are the link relations for the application/vnd.sas.microanalytic.module.source media type.

Relationship	HTTP Method	Description
self	GET	A link to the source code that was used to compile the module. URI: SASMicroAnalyticService/rest//modules/{*moduleId*}/source Media type: application/vnd.sas.microanalytic.module.source
up	GET	A link back to the module. URI: SASMicroAnalyticService/rest//modules/{*ModuleID*} Media type: application/vnd.sas.microanalytic.module

The application/vnd.sas.microanalytic.module.source media type contains the following members.

Name	Type	Description
version	integer	This media type's schema version number. This representation is version 1.
moduleId	string	A generated unique string identifying a module in an installation.
source	string	The source code used to create the module.
links	array of link objects	Zero or more link objects. See the table above for a description of the link types.

Here is an example of application/vnd.sas.microanalytic.module.source +json:

```
{
```

```
   "moduleId":"36af8e3c-6a37-4494-a8e0-9cc96ad62232",
   "source":"ds2_options sas;\n package sampleModule / overwrite=yes; \n \n
    method copy_charN_array(char(12) in_array[4], in_out char(12)  out_array[4]);\n
    out_array := in_array;\n end;\n \n method copy_varchar_array(varchar(512) in_array[3],
    in_out varchar out_array[3]);\n out_array := in_array;\n end;\n \n
    method copy_int_array(int in_array[5], in_out int out_array[5]);\n out_array := in_array;\n
    end;\n \n method copy_float_array(double in_array[2], in_out double out_array[2]);\n
    out_array := in_array;\n end;\n \n method copy_bigint_array(bigint in_array[1],
    in_out bigint out_array[1]);\n out_array := in_array;\n end;\n \n method copy_arrays( char(12)
    in_charN_array[4],\n varchar(512) in_varchar_array[1],\n int in_int_array[5],
    \n double in_double_array[2], \n bigint in_bigint_array[1], \n in_out char(12)
    out_charN_array[4],\n in_out varchar(512) out_varchar_array[1],\n in_out int out_int_array[5],\n
    in_out double out_double_array[2],\n in_out bigint out_bigint_array[1]);\n \n
    copy_charN_array(in_charN_array, out_charN_array);\n copy_int_array(in_int_array, out_int_array);\n
    copy_float_array(in_double_array, out_double_array);\n copy_bigint_array(in_bigint_array,
    out_bigint_array);\n \n end;\n \n endpackage;\n \n \n",
   "links":[
      {
         "method":"GET",
         "rel":"self",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 36af8e3c-6a37-4494-a8e0-9cc96ad62232/source",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/source",
         "type":"application/vnd.sas.microanalytic.module.source"
      },
      {
         "method":"GET",
         "rel":"up",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
          36af8e3c-6a37-4494-a8e0-9cc96ad62232",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232",
         "type":"application/vnd.sas.microanalytic.module"
      }
   ],
   "version":1
}
```


application/vnd.sas.microanalytic.module.step

The application/vnd.sas.microanalytic.module.step media type describes the step that is returned by SAS Micro Analytic Service when a GET is performed on the step's collection. Step instances are created by posting a module to the module's collection.

Here are the link relations for the application/vnd.sas.microanalytic.module.step media type.

Relationship	HTTP Method	Description
self	GET	A link to the individual step of a specific module. URI: SASMicroAnalyticService/rest/modules/{*moduleId*}/steps/{*stepId*} Media type: application/vnd.sas.microanalytic.module.step

Relationship	HTTP Method	Description
up	GET	A link back to the module's collection of steps. URI: SASMicroAnalyticService/rest/modules/{*moduleId*}/steps Media type: application/vnd.sas.collection
validate	POST	A link used to validate that the input values are correct for a specific step of a module. URI: SASMicroAnalyticService/rest/modules/{*moduleId*}/steps/{*stepId*}/validations Media type: application/vnd.sas.microanalytic.module.step.input.validity
execute	POST	A link used to execute a specific step of a module. URI: SASMicroAnalyticService/rest/modules/{*moduleId*}/steps/{*stepId*} Media type: application/vnd.sas.microanalytic.module.step.output

The application/vnd.sas.microanalytic.module.step media type contains the following members.

Name	Type	Description
version	integer	This media type's schema version number. This representation is version 1.
id	string	The name of a step that is included in the compiled module.
moduleId	string	A generated unique string identifying a module in an installation.
description	string	Text describing the rules and logic performed by the step.

Name	Type	Description
inputs	array	Provides information about the specific input values that should be specified in the request body when executing a step. Here are the representation members: name string - The name of a variable that is expected to be passed into the step. type string (ENUM) - This is the data type of the variable. If the variable's type is (array of) integer, long, or decimal, the value must be a JSON (array of) number. If the variable's type is (array of) string or char, the value must be a JSON (array of) string. Only arrays with one dimension are supported. Null is used to represent missing values. The following data types are supported: • decimal - For DS2, this corresponds to the double data type. • bigint • integer • string • decimalArray • bigintArray • integerArray • stringArray size integer - For a string type, this field indicates the length of the string, which is at least one. For a non-string type, this field has the value of zero. dim integer - For an array type, this field indicates the length of the array, which is one or greater. For a non-array type, this field has a value of zero.

Name	Type	Description
outputs	array	Provides information about the specific output values that should be expected in the response body of a step execution. Here are the representation members: name string - The name of a variable that is expected to receive output from the step. type string (ENUM) - This is the data type of the variable. If the variable's type is (array of) integer, long, or decimal, the value must be a JSON (array of) number. If the variable's type is (array of) string or char, the value must be a JSON (array of) string. Only arrays with one dimension are supported. The following data types are supported: • decimal - For DS2, this corresponds to the double data type. • bigint • integer • string • decimalArray • bigintArray • integerArray • stringArray size integer - For a string type, this field indicates the length of the string. For a non-string type, this field has the value of zero. For DS2, the variable length is not required since an output variable is passed by reference. A zero is reported if a length is not specified. Otherwise, the length specified is reported. dim integer - For an array type, this field indicates the length of the array, which is one or greater. For a non-array type, this field has a value of zero.
links	array of link objects	Zero or more link objects. See the table above for a description of the link types.

Here is an example of application/vnd.sas.microanalytic.module.step+json:

```
{
   "links":[
      {
         "method":"GET",
         "rel":"self",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_arrays",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_arrays",
         "type":"application/vnd.sas.microanalytic.module.step"
```

```
    },
    {
       "method":"GET",
       "rel":"up",
       "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
       "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
       "type":"application/vnd.sas.collection"
    },
    {
       "method":"POST",
       "rel":"validate",
       "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_arrays/validations",
       "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_arrays/validations",
       "type":"application/vnd.sas.microanalytic.module.step.input.validity"
    },
    {
       "method":"POST",
       "rel":"execute",
       "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_arrays",
       "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_arrays",
       "type":"application/vnd.sas.microanalytic.module.step.output"
    }
 ],
 "id":"copy_arrays",
 "moduleId":"36af8e3c-6a37-4494-a8e0-9cc96ad62232",
 "inputs":[
    {
       "name":"in_charN_array",
       "type":"stringArray",
       "dim":4,
       "size":12
    },
    {
       "name":"in_varchar_array",
       "type":"stringArray",
       "dim":1,
       "size":512
    },
    {
       "name":"in_int_array",
       "type":"integerArray",
       "dim":5,
       "size":0
    },
    {
       "name":"in_double_array",
       "type":"decimalArray",
       "dim":2,
       "size":0
    },
    {
       "name":"in_bigint_array",
       "type":"bigintArray",
```

```
         "dim":1,
         "size":0
      }
   ],
   "outputs":[
      {
         "name":"out_charN_array",
         "type":"stringArray",
         "dim":4,
         "size":12
      },
      {
         "name":"out_varchar_array",
         "type":"stringArray",
         "dim":1,
         "size":512
      },
      {
         "name":"out_int_array",
         "type":"integerArray",
         "dim":5,
         "size":0
      },
      {
         "name":"out_double_array",
         "type":"decimalArray",
         "dim":2,
         "size":0
      },
      {
         "name":"out_bigint_array",
         "type":"bigintArray",
         "dim":1,
         "size":0
      }
   ],
   "description":null,
   "version":1
}
```


application/vnd.sas.microanalytic.module.step.input

The application/vnd.sas.microanalytic.module.step.input media type describes the input values that are required by SAS Micro Analytic Service step when a POST is used to validate or execute a step.

The application/vnd.sas.microanalytic.module.step.input media type contains the following members.

Name	Type	Description
version	integer	This media type's schema version number. This representation is version 1.

Name	Type	Description
inputs	array	Holds the values that are to be passed to the step for input validation or execution. The order of the variables should match the order presented in the input signature. Here are the representation members: name string - The name of an input variable for the step. value varies - This represents the actual value to set on the variable. If the variable's type is (array of) integer, long, or decimal, the value must be a JSON (array of) number. If the variable's type is (array of) string, the value must be a JSON (array of) string.

Here is an example of application/vnd.sas.microanalytic.module.step.input+json:

```
{
  "version" : 1,
  "inputs":[
      {
         "name":"supported_browsers",
         "value":[
            "Apple Safari",
            "Google Chrome",
            "Microsoft Internet Explorer",
            "Mozilla Firefox"
         ]
      },
      {
         "name":"random_integers",
         "value":[
            10,
            15,
            3
         ]
      },
      {
         "name": "AMBALANCE",
         "value" : 1055.93
      }
  ]
}
```

application/vnd.sas.microanalytic.module.step.input.validity

The application/vnd.sas.microanalytic.module.step.input.validity media type describes the output values that are returned by SAS Micro Analytic Service for a POST to validate the inputs required to execute a step.

The application/vnd.sas.microanalytic.module.step.input.validity media type contains the following members.

Name	Type	Description
version	integer	This media type's schema version number. This representation is version 1.
moduleId	string	A generated unique string identifying a module in an installation.
stepId	string	The name of a step.
valid	Boolean	The value is true if all the input parameters are valid. If any parameter is invalid, the value is false.
results	objects	The object contains a member for each input parameter that is invalid. The name of the member is that of an input parameter. The value is the reason why the input is invalid. The object is empty if there is no invalid input parameter.

Here is an example of application/vnd.sas.microanalytic.module.step.input.validity+json:

```
{
  "version" : 1,
  "moduleId": "83e7d274-fe17-429e-92ca-93ec2153c731",
  "stepId":"predict",
  "valid":false,
  "results":
    {
      "s2": "String value expected but found string array value [String].",
      "s4": "Bigint value expected but found double value 77.0."
    }
}
```

application/vnd.sas.microanalytic.module.step.output

The application/vnd.sas.microanalytic.module.step.output media type describes the output values that are returned by SAS Micro Analytic Service when a step is executed.

The application/vnd.sas.microanalytic.module.step.output media type contains the following members.

Name	Type	Description
version	integer	This media type's schema version number. This representation is version 1.
moduleId	string	A generated unique string identifying a module in an installation.
stepId	string	The name of the step.

Name	Type	Description
outputs	array	Holds the output values returned from executing a step. The order of the variables matches the order presented in the output signature. Here are the representation members: name string - The name of the variable that is expected to receive output from the step. value This represents the actual value returned from the step execution.

Here is an example of application/vnd.sas.microanalytic.module.step.output+json:

```
{
  "moduleId": "70a58acd-5618-4dc3-9d7a-9e675e8e13bb",
  "stepId": "test_all_types",
  "outputs": [
    {
      "name": "out_string",
      "value": "This is a test..."
    },
    {
      "name": "out_bigint",
      "value": 987654321
    },
    {
      "name": "out_int",
      "value": 7654321
    },
    {
      "name": "out_double",
      "value": 0.9997
    },
    {
      "name": "string_arr",
      "value": [
        "John Jacob Hale",
        "Male",
        "Master Swimmer"
      ]
    },
    {
      "name": "bigint_arr",
      "value": [
        1078653221,
        2256390877,
        9719886300
      ]
    },
    {
      "name": "int_arr",
      "value": [
        77,
        436702,
```

```
          67552
        ]
      },
      {
        "name": "double_arr",
        "value": [
          0.9997,
          1.0,
          0.0023
        ]
      }
    ],
    "version": 1
  }
```

Resources and Collections

resource /

The root / returns links to the top-level resources surfaced through this API. The module's collection is the only top-level resource. The GET link is for querying the module's collection. The POST link is for creating a module.

The / resource uses the GET / method, which requires authentication, and has a request URL of GET http://*www.example.com*/SASMicroAnalyticService/rest/.

The response to the GET request is a collection of links to the resources. In this version, the module's collection is the only top-level resource.

Here is a JSON representation of the top-level resource containing links:

```
{
   "version":1,
   "links":[
      {
         "method":"GET",
         "rel":"modules",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules",
         "uri":"/modules"
      },
      {
         "method":"POST",
         "rel":"createModule",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules",
         "uri":"/modules"
      }
   ]
}
```

Here are the HTTP response codes:

200
: OK

401
: Unauthorized

500
Server error.

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

GET / returns the application/json media type representation by setting the Accept: header of the request.

Collection /modules

The /modules resource collection is a collection of modules that are loaded in memory by SAS Micro Analytic Service.

The /modules resource allows the GET method, which requires authentication, and has a request URL of GET http://*www.example.com*/SASMicroAnalyticService/rest/modules.

Each module object in the collection contains fields and links that enable you to get detailed information about a specific module.

Here are the HTTP response codes:

200
OK

401
Unauthorized

500
Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

Here are the query parameters for /modules:

Name	Type	Description
?start	integer	The starting index of the first item in a page. The index is 0-based. The default is 0.
?limit	integer	The maximum number of modules to return in this page of results. The actual number of returned modules might be less, if the collection has been exhausted. The default is 10.
?label	string	Filter by the name of the modules. Each module is checked if its name contains the label.

Here is an example of the JSON representation:

```
{
   "links":[
      {
         "method":"GET",
         "rel":"self",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules",
         "uri":"/modules",
         "type":"application/vnd.sas.collection"
      },
      {
```

```
      "method":"GET",
      "rel":"first",
      "href":"http://www.example.com/SASMicroAnalyticService/rest/modules?start=0&limit=5",
      "uri":"/modules?start=0&limit=5",
      "type":"application/vnd.sas.collection"
   },
   {
      "method":"GET",
      "rel":"last",
      "href":"http://www.example.com/SASMicroAnalyticService/rest/modules?start=0&limit=5",
      "uri":"/modules?start=0&limit=5",
      "type":"application/vnd.sas.collection"
   }
],
"name":"items",
"accept":"application/vnd.sas.microanalytic.module",
"start":0,
"count":5,
"items":[
   {
      "links":[
         {
            "method":"GET",
            "rel":"self",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                    3eadfae7-583f-44ee-8c37-e201184c94da",
            "uri":"/modules/3eadfae7-583f-44ee-8c37-e201184c94da",
            "type":"application/vnd.sas.microanalytic.module"
         },
         {
            "method":"GET",
            "rel":"up",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules",
            "uri":"/modules",
            "type":"application/vnd.sas.collection"
         },
         {
            "method":"GET",
            "rel":"source",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                    3eadfae7-583f-44ee-8c37-e201184c94da/source",
            "uri":"/modules/3eadfae7-583f-44ee-8c37-e201184c94da/source",
            "type":"application/vnd.sas.microanalytic.module.source"
         },
         {
            "method":"GET",
            "rel":"steps",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                    3eadfae7-583f-44ee-8c37-e201184c94da/steps",
            "uri":"/modules/3eadfae7-583f-44ee-8c37-e201184c94da/steps",
            "type":"application/vnd.sas.collection"
         },
         {
            "method":"PUT",
            "rel":"update",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
```

```
                3eadfae7-583f-44ee-8c37-e201184c94da",
            "uri":"/modules/3eadfae7-583f-44ee-8c37-e201184c94da",
            "type":"application/vnd.sas.microanalytic.module"
         },
         {
            "method":"DELETE",
            "rel":"delete",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                3eadfae7-583f-44ee-8c37-e201184c94da",
            "uri":"/modules/3eadfae7-583f-44ee-8c37-e201184c94da"
         }
      ],
      "description":"Module A",
      "version":1,
      "scope":"public",
      "id":"3eadfae7-583f-44ee-8c37-e201184c94da",
      "steps":[
         "falls_on"
      ],
      "properties":[
      ],
      "revision":1,
      "creationTimeStamp":"2015-05-06T22:37:44.000-0400",
      "modifiedTimeStamp":"2015-05-06T22:37:44.000-0400",
      "name":"pkga"
   },
   {
      "links":[
         {
            "method":"GET",
            "rel":"self",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                de279ebf-f2a6-42ec-9342-29c363866a08",
            "uri":"/modules/de279ebf-f2a6-42ec-9342-29c363866a08",
            "type":"application/vnd.sas.microanalytic.module"
         },
         {
            "method":"GET",
            "rel":"up",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules",
            "uri":"/modules",
            "type":"application/vnd.sas.collection"
         },
         {
            "method":"GET",
            "rel":"source",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                de279ebf-f2a6-42ec-9342-29c363866a08/source",
            "uri":"/modules/de279ebf-f2a6-42ec-9342-29c363866a08/source",
            "type":"application/vnd.sas.microanalytic.module.source"
         },
         {
            "method":"GET",
            "rel":"steps",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                de279ebf-f2a6-42ec-9342-29c363866a08/steps",
```

```
            "uri":"/modules/de279ebf-f2a6-42ec-9342-29c363866a08/steps",
            "type":"application/vnd.sas.collection"
         },
         {
            "method":"PUT",
            "rel":"update",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                   de279ebf-f2a6-42ec-9342-29c363866a08",
            "uri":"/modules/de279ebf-f2a6-42ec-9342-29c363866a08",
            "type":"application/vnd.sas.microanalytic.module"
         },
         {
            "method":"DELETE",
            "rel":"delete",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                   de279ebf-f2a6-42ec-9342-29c363866a08",
            "uri":"/modules/de279ebf-f2a6-42ec-9342-29c363866a08"
         }
      ],
      "description":"Module B",
      "version":1,
      "scope":"public",
      "id":"de279ebf-f2a6-42ec-9342-29c363866a08",
      "steps":[
         "this_year"
      ],
      "properties":[
      ],
      "revision":1,
      "creationTimeStamp":"2015-05-06T22:37:45.000-0400",
      "modifiedTimeStamp":"2015-05-06T22:37:45.000-0400",
      "name":"pkgb"
   },
   {
      "links":[
         {
            "method":"GET",
            "rel":"self",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                   f1dcd1af-6ab2-4ac0-a5c6-5c64d5c09016",
            "uri":"/modules/f1dcd1af-6ab2-4ac0-a5c6-5c64d5c09016",
            "type":"application/vnd.sas.microanalytic.module"
         },
         {
            "method":"GET",
            "rel":"up",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules",
            "uri":"/modules",
            "type":"application/vnd.sas.collection"
         },
         {
            "method":"GET",
            "rel":"source",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                   f1dcd1af-6ab2-4ac0-a5c6-5c64d5c09016/source",
            "uri":"/modules/f1dcd1af-6ab2-4ac0-a5c6-5c64d5c09016/source",
```

```
            "type":"application/vnd.sas.microanalytic.module.source"
         },
         {
            "method":"GET",
            "rel":"steps",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                   f1dcd1af-6ab2-4ac0-a5c6-5c64d5c09016/steps",
            "uri":"/modules/f1dcd1af-6ab2-4ac0-a5c6-5c64d5c09016/steps",
            "type":"application/vnd.sas.collection"
         },
         {
            "method":"PUT",
            "rel":"update",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                   f1dcd1af-6ab2-4ac0-a5c6-5c64d5c09016",
            "uri":"/modules/f1dcd1af-6ab2-4ac0-a5c6-5c64d5c09016",
            "type":"application/vnd.sas.microanalytic.module"
         },
         {
            "method":"DELETE",
            "rel":"delete",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                   f1dcd1af-6ab2-4ac0-a5c6-5c64d5c09016",
            "uri":"/modules/f1dcd1af-6ab2-4ac0-a5c6-5c64d5c09016"
         }
      ],
      "description":"Module C",
      "version":1,
      "scope":"public",
      "id":"f1dcd1af-6ab2-4ac0-a5c6-5c64d5c09016",
      "steps":[
         "get_date"
      ],
      "properties":[
      ],
      "revision":1,
      "creationTimeStamp":"2015-05-06T22:37:46.000-0400",
      "modifiedTimeStamp":"2015-05-06T22:37:46.000-0400",
      "name":"pkgc"
   },
   {
      "links":[
         {
            "method":"GET",
            "rel":"self",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/
                   modules/617aad65-36fa-4079-b1cb-03fe948874d4",
            "uri":"/modules/617aad65-36fa-4079-b1cb-03fe948874d4",
            "type":"application/vnd.sas.microanalytic.module"
         },
         {
            "method":"GET",
            "rel":"up",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules",
            "uri":"/modules",
            "type":"application/vnd.sas.collection"
```

```
      },
      {
         "method":"GET",
         "rel":"source",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                617aad65-36fa-4079-b1cb-03fe948874d4/source",
         "uri":"/modules/617aad65-36fa-4079-b1cb-03fe948874d4/source",
         "type":"application/vnd.sas.microanalytic.module.source"
      },
      {
         "method":"GET",
         "rel":"steps",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                617aad65-36fa-4079-b1cb-03fe948874d4/steps",
         "uri":"/modules/617aad65-36fa-4079-b1cb-03fe948874d4/steps",
         "type":"application/vnd.sas.collection"
      },
      {
         "method":"PUT",
         "rel":"update",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                617aad65-36fa-4079-b1cb-03fe948874d4",
         "uri":"/modules/617aad65-36fa-4079-b1cb-03fe948874d4",
         "type":"application/vnd.sas.microanalytic.module"
      },
      {
         "method":"DELETE",
         "rel":"delete",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                617aad65-36fa-4079-b1cb-03fe948874d4",
         "uri":"/modules/617aad65-36fa-4079-b1cb-03fe948874d4"
      }
   ],
   "description":"Module D",
   "version":1,
   "scope":"public",
   "id":"617aad65-36fa-4079-b1cb-03fe948874d4",
   "steps":[
      "holiday_reminder"
   ],
   "properties":[
   ],
   "revision":1,
   "creationTimeStamp":"2015-05-06T22:37:46.000-0400",
   "modifiedTimeStamp":"2015-05-06T22:37:46.000-0400",
   "name":"pkgd"
},
{
   "links":[
      {
         "method":"GET",
         "rel":"self",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232",
         "type":"application/vnd.sas.microanalytic.module"
```

```
    },
    {
       "method":"GET",
       "rel":"up",
       "href":"http://www.example.com/SASMicroAnalyticService/rest/modules",
       "uri":"/modules",
       "type":"application/vnd.sas.collection"
    },
    {
       "method":"GET",
       "rel":"source",
       "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232/source",
       "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/source",
       "type":"application/vnd.sas.microanalytic.module.source"
    },
    {
       "method":"GET",
       "rel":"steps",
       "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
       "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
       "type":"application/vnd.sas.collection"
    },
    {
       "method":"PUT",
       "rel":"update",
       "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232",
       "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232",
       "type":"application/vnd.sas.microanalytic.module"
    },
    {
       "method":"DELETE",
       "rel":"delete",
       "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232",
       "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232"
    }
 ],
 "description":"Sample module",
 "version":1,
 "scope":"public",
 "warnings":{
    "errorCode":0,
    "message":"Module compiled with warnings.",
    "details":[
       "In declaration of method copy_arrays: parameter out_charN_array is 'in_out';
        therefore, the type size (12) will be ignored.",
       "In declaration of method copy_arrays: parameter out_varchar_array is 'in_out';
        therefore, the type size (512) will be ignored.",
       "In declaration of method copy_charN_array: parameter out_array is 'in_out';
        therefore, the type size (12) will be ignored."
    ],
    "remediation":"",
    "links":[
```

```
            ],
            "version":1,
            "httpStatusCode":0
         },
         "id":"36af8e3c-6a37-4494-a8e0-9cc96ad62232",
         "steps":[
            "copy_arrays",
            "copy_bigint_array",
            "copy_charN_array",
            "copy_float_array",
            "copy_int_array",
            "copy_varchar_array"
         ],
         "properties":[
         ],
         "revision":1,
         "creationTimeStamp":"2015-05-06T22:41:02.000-0400",
         "modifiedTimeStamp":"2015-05-06T22:41:02.000-0400",
         "name":"samplemodule"
      }
   ],
   "limit":5,
   "version":1
}
```


GET returns the following media type representations by setting the Accept: header of the request:

- application/vnd.sas.collection
- application/json

This operation can return the application/vnd.sas.error media type for failure. This media type is returned when the server encounters an error. An example of an error is when a node in a clustered deployment has become out of sync.

The POST method returns a module resource for the module that is loaded in memory by SAS Micro Analytic Service. The module resource that is returned contains links to the compiled and loaded steps.

The POST method requires authentication and has a request URL of POST http://*www.example.com*/SASMicroAnalyticService/rest/modules.

Here is an example of the JSON representation:


```
{
"version": "1",
"description": "Sample module",
"scope" : "public",
"type" : "text/vnd.sas.source.ds2",
"properties" : [],
"code" : "ds2_options sas;\n package sampleModule / overwrite=yes; \n \n
method copy_charN_array(char(12) in_array[4], in_out char(12)  out_array[4]);\n
out_array := in_array;\n end;\n \n method copy_varchar_array(varchar(512) in_array[3],
in_out varchar out_array[3]);\n out_array := in_array;\n end;\n \n
method copy_int_array(int in_array[5], in_out int out_array[5]);\n out_array := in_array;\n
end;\n \n method copy_float_array(double in_array[2], in_out double out_array[2]);\n
out_array := in_array;\n end;\n \n method copy_bigint_array(bigint in_array[1],
in_out bigint out_array[1]);\n out_array := in_array;\n end;\n \n method copy_arrays( char(12)
```

```
in_charN_array[4],\n varchar(512) in_varchar_array[1],\n int in_int_array[5], \n
double in_double_array[2], \n bigint in_bigint_array[1], \n in_out char(12)  out_charN_array[4],\n
in_out varchar(512) out_varchar_array[1],\n in_out int out_int_array[5],\n
in_out double out_double_array[2],\n in_out bigint out_bigint_array[1]);\n \n
copy_charN_array(in_charN_array, out_charN_array);\n copy_int_array(in_int_array,
out_int_array);\n copy_float_array(in_double_array, out_double_array);\n
copy_bigint_array(in_bigint_array, out_bigint_array);\n \n end;\n \n endpackage;\n \n \n"
        }
```

The POST method accepts the following content types, as named by the Content-Type: header:

- application/json
- application/vnd.sas.microanalytic.module.definition+json

Here are the HTTP response codes:

201
Created

400
Bad Request

401
Unauthorized

403
Forbidden

500
Server error.

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

One situation that causes a 403 code to be returned is when the POST is initiated from an untrusted site.

This operation returns the following media type representations by setting the Accept: header of the request:

- application/json
- application/vnd.sas.microanalytic.module+json

This operation returns the application/vnd.sas.error media type for failure. This media type is returned when there is an error creating the module. An example is when the source code contains a syntax error. Another example is when the module name is already taken.

Here is an example of a successfully compiled module with no warnings:

```
{
   "links":[
      {
         "method":"GET",
         "rel":"self",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
          36af8e3c-6a37-4494-a8e0-9cc96ad62232",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232",
         "type":"application/vnd.sas.microanalytic.module"
      },
      {
```

```
      "method":"GET",
      "rel":"up",
      "href":"http://www.example.com/SASMicroAnalyticService/rest/modules",
      "uri":"/modules",
      "type":"application/vnd.sas.collection"
   },
   {
      "method":"GET",
      "rel":"source",
      "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232/source",
      "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/source",
      "type":"application/vnd.sas.microanalytic.module.source"
   },
   {
      "method":"GET",
      "rel":"steps",
      "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
      "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
      "type":"application/vnd.sas.collection"
   },
   {
      "method":"PUT",
      "rel":"update",
      "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232",
      "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232",
      "type":"application/vnd.sas.microanalytic.module"
   },
   {
      "method":"DELETE",
      "rel":"delete",
      "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232",
      "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232"
   }
],
"description":"Sample module",
"version":1,
"scope":"public",
"id":"36af8e3c-6a37-4494-a8e0-9cc96ad62232",
"steps":[
   "copy_arrays",
   "copy_bigint_array",
   "copy_charN_array",
   "copy_float_array",
   "copy_int_array",
   "copy_varchar_array"
],
"properties":[
],
"revision":1,
"creationTimeStamp":"2015-05-06T22:14:17.000-0400",
"modifiedTimeStamp":"2015-05-06T22:14:17.000-0400",
"name":"samplemodule"
```

```
}
```

Here is an example of a successfully compiled module with warnings:


```
{
   "links":[
      {
         "method":"GET",
         "rel":"self",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 36af8e3c-6a37-4494-a8e0-9cc96ad62232",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232",
         "type":"application/vnd.sas.microanalytic.module"
      },
      {
         "method":"GET",
         "rel":"up",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules",
         "uri":"/modules",
         "type":"application/vnd.sas.collection"
      },
      {
         "method":"GET",
         "rel":"source",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 36af8e3c-6a37-4494-a8e0-9cc96ad62232/source",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/source",
         "type":"application/vnd.sas.microanalytic.module.source"
      },
      {
         "method":"GET",
         "rel":"steps",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
         "type":"application/vnd.sas.collection"
      },
      {
         "method":"PUT",
         "rel":"update",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 36af8e3c-6a37-4494-a8e0-9cc96ad62232",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232",
         "type":"application/vnd.sas.microanalytic.module"
      },
      {
         "method":"DELETE",
         "rel":"delete",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 36af8e3c-6a37-4494-a8e0-9cc96ad62232",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232"
      }
   ],
   "description":"Sample module",
   "version":1,
   "scope":"public",
   "warnings":{
```

```
      "errorCode":0,
      "message":"Module compiled with warnings.",
      "details":[
         "In declaration of method copy_arrays: parameter out_charN_array is 'in_out';
          therefore, the type size (12) will be ignored.",
         "In declaration of method copy_arrays: parameter out_varchar_array is 'in_out';
          therefore, the type size (512) will be ignored.",
         "In declaration of method copy_charN_array: parameter out_array is 'in_out';
          therefore, the type size (12) will be ignored."
      ],
      "remediation":"",
      "links":[
      ],
      "version":1,
      "httpStatusCode":0
   },
   "id":"36af8e3c-6a37-4494-a8e0-9cc96ad62232",
   "steps":[
      "copy_arrays",
      "copy_bigint_array",
      "copy_charN_array",
      "copy_float_array",
      "copy_int_array",
      "copy_varchar_array"
   ],
   "properties":[
   ],
   "revision":1,
   "creationTimeStamp":"2015-05-06T22:41:02.000-0400",
   "modifiedTimeStamp":"2015-05-06T22:41:02.000-0400",
   "name":"samplemodule"
}
```


Here is an example of an error response:


```
{
   "errorCode":-30,
   "message":"Invalid source code.  ",
   "details":[
      "Line 1: Parse failed:  int out_int); out_int=3; end;
       >>> endpackages <<< ; package ship_backen",
      "Parse encountered identifier when expecting end of input."
   ],
   "remediation":"",
   "links":[

   ],
   "version":1,
   "httpStatusCode":400
}
```


resource /modules/{moduleId}

The /modules/{*moduleId*} resource is a single compiled module that is loaded in memory by SAS Micro Analytic Service.

The /modules/{*moduleId*} resource has the following methods:

- GET
- PUT
- DELETE

The GET method requires authentication and has a request URL of GET http://*www.example.com*/SASMicroAnalyticService/modules/{*moduleId*}.

Here are the HTTP response codes:

200
OK

401
Unauthorized

404
Not found

500
Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

This operation returns the following media type representations by setting the Accept: header of the request:

- application/json
- application/vnd.sas.microanalytic.module+json

This operation returns the application/vnd.sas.error media type for failure. This media type is returned when the resource cannot be located either because the module ID is incorrect or the module has been deleted.

Here is an example of a JSON response:

```
{
    "links":[
         {
            "method":"GET",
            "rel":"self",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                    45e7118a-c61b-4e59-b5b1-9a415355551f",
            "uri":"/modules/45e7118a-c61b-4e59-b5b1-9a415355551f",
            "type":"application/vnd.sas.microanalytic.module"
         },
         {
            "method":"GET",
            "rel":"up",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules",
            "uri":"/modules",
            "type":"application/vnd.sas.collection"
         },
         {
            "method":"GET",
            "rel":"source",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                    45e7118a-c61b-4e59-b5b1-9a415355551f/source",
            "uri":"/modules/45e7118a-c61b-4e59-b5b1-9a415355551f/source",
```

```
            "type":"application/vnd.sas.microanalytic.module.source"
         },
         {
            "method":"GET",
            "rel":"steps",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                   45e7118a-c61b-4e59-b5b1-9a415355551f/steps",
            "uri":"/modules/45e7118a-c61b-4e59-b5b1-9a415355551f/steps",
            "type":"application/vnd.sas.collection"
         },
         {
            "method":"PUT",
            "rel":"update",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                   45e7118a-c61b-4e59-b5b1-9a415355551f",
            "uri":"/modules/45e7118a-c61b-4e59-b5b1-9a415355551f",
            "type":"application/vnd.sas.microanalytic.module"
         },
         {
            "method":"DELETE",
            "rel":"delete",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                   45e7118a-c61b-4e59-b5b1-9a415355551f",
            "uri":"/modules/45e7118a-c61b-4e59-b5b1-9a415355551f"
         }
   ],
   "version":1,
   "description":"Decision Tree Model",
   "scope":"private",
   "id":"45e7118a-c61b-4e59-b5b1-9a415355551f",
   "steps":[
      "score"
   ],
   "properties":[
   ],
   "creationTimeStamp":"2015-04-13T01:11:44.000-0400",
   "modifiedTimeStamp":"2015-04-13T01:11:44.000-0400",
   "revision":1,
   "name":"tree"
}
```


Here is an example of a JSON error response:


```
{
  "errorCode": 4001,
  "message": "No module with the module id 48B9A582-ADA4-C64D-9759-BBEB8E1DAA8B exists.",
  "details": [],
  "remediation": "",
  "links": [],
  "version": 1,
  "httpStatusCode": 404
}
```


The PUT method updates a module resource for the module that is loaded in memory by SAS Micro Analytic Service. It is an error to change the name of the module in a PUT operation. The module resource that is returned contain links to the compiled and loaded

steps. The latest revision supersedes previous revisions. Previous revisions are not retrievable.

The PUT method requires authentication and has a request URL of PUT http://*www.example.com*/SASMicroAnalyticService/rest/modules/{*moduleId*}.

The PUT method accepts the following media type representations by setting the Content-Type: header of the request:

- application/json
- application/vnd.sas.microanalytic.module.definition+json

Here are the HTTP response codes:

200
OK

400
Bad request

401
Unauthorized

403
Forbidden

404
Not found

500
Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

One situation that causes a 403 code to be returned is when the PUT is initiated from an untrusted site.

Here is an example of the JSON representation:

```
{
"version": "1",
"description": "Expanded sample module",
"scope" : "public",
"type" : "text/vnd.sas.source.ds2",
"properties" : [  {"name" : "connectionString", "value" : "DRIVER=base;"} ],
"code" : "ds2_options sas;\n package sampleModule / overwrite=yes; \n \n
method produce_warnings(char(12) in_string, in_out char(12) out_string);\n
out_string = in_string;\n end;\n \n method copy_char12(char(12) in_string,
in_out char out_string);\n out_string=in_string;\n end;\n \n
method copy_varchar(varchar(32767) in_string, in_out varchar out_string);\n
out_string=in_string;\n end;\n \n method copy_bigint(bigint in_int,
in_out bigint out_int);\n out_int=in_int;\n end;\n \n method copy_float(double in_float,
in_out double out_float);\n out_float=in_float;\n end;\n \n
method copy_int(int in_int, in_out int out_int);\n out_int=in_int;\n end;\n \n
method copy_scalars(char(12) in_char12, varchar(32767) in_varchar, int in_int,\n
bigint in_bigint, double in_float, \n in_out char out_char, in_out char out_char12,\n
in_out varchar out_varchar, in_out int out_int,\n in_out bigint out_bigint,
in_out double out_float);\n \n copy_char12(in_char12, out_char12);\n
copy_varchar(in_varchar, out_varchar);\n copy_bigint(in_bigint, out_bigint);\n
copy_float(in_float, out_float);\n copy_int(in_int, out_int);\n end;\n \n
method copy_charN_array(char(12) in_array[4], in_out char(12)  out_array[4]);\n
```

```
out_array := in_array;\n end;\n \n method copy_varchar_array(varchar(512) in_array[3],
in_out varchar out_array[3]);\n out_array := in_array;\n end;\n \n
method copy_int_array(int in_array[5], in_out int out_array[5]);\n out_array := in_array;\n
end;\n \n method copy_float_array(double in_array[2], in_out double out_array[2]);\n
out_array := in_array;\n end;\n \n method copy_bigint_array(bigint in_array[1],
bigint out_array[1]);\n out_array := in_array;\n end;\n \n method copy_arrays( char(12)
in_charN_array[4],\n varchar(512) in_varchar_array[1],\n int in_int_array[5], \n
double in_double_array[2], \n bigint in_bigint_array[1], \n in_out char(12)
out_charN_array[4],\n in_out varchar(512) out_varchar_array[1],\n in_out int out_int_array[5],\n
in_out double out_double_array[2],\n in_out bigint out_bigint_array[1]);\n \n
copy_charN_array(in_charN_array, out_charN_array);\n copy_int_array(in_int_array,
out_int_array);\n copy_float_array(in_double_array, out_double_array);\n
copy_bigint_array(in_bigint_array, out_bigint_array);\n \n end;\n \n endpackage;\n \n \n"
      }
```

This operation returns the following media type representations by setting the Accept: header of the request:

- application/json
- application/vnd.sas.microanalytic.module+json

This operation returns the application/vnd.sas.error media type when there is an error. For example, this media type is returned when you attempt to change the name of the module, or the source code contains a syntax error. Another example is when the server fails to acquire a resource.

Here is an example of a successfully compiled module response body:

```
{
   "links":[
      {
         "method":"GET",
         "rel":"self",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 36af8e3c-6a37-4494-a8e0-9cc96ad62232",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232",
         "type":"application/vnd.sas.microanalytic.module"
      },
      {
         "method":"GET",
         "rel":"up",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules",
         "uri":"/modules",
         "type":"application/vnd.sas.collection"
      },
      {
         "method":"GET",
         "rel":"source",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 36af8e3c-6a37-4494-a8e0-9cc96ad62232/source",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/source",
         "type":"application/vnd.sas.microanalytic.module.source"
      },
      {
         "method":"GET",
         "rel":"steps",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
```

```
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
         "type":"application/vnd.sas.collection"
      },
      {
         "method":"PUT",
         "rel":"update",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 36af8e3c-6a37-4494-a8e0-9cc96ad62232",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232",
         "type":"application/vnd.sas.microanalytic.module"
      },
      {
         "method":"DELETE",
         "rel":"delete",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 36af8e3c-6a37-4494-a8e0-9cc96ad62232",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232"
      }
   ],
   "description":"Expanded sample module",
   "version":1,
   "scope":"public",
   "id":"36af8e3c-6a37-4494-a8e0-9cc96ad62232",
   "steps":[
      "copy_arrays",
      "copy_bigint",
      "copy_bigint_array",
      "copy_char12",
      "copy_charN_array",
      "copy_float",
      "copy_float_array",
      "copy_int",
      "copy_int_array",
      "copy_scalars",
      "copy_varchar",
      "copy_varchar_array",
      "produce_warnings"
   ],
   "properties":[
      {
         "name" : "connectionString",
         "value" : "DRIVER=base;"
      }
   ],
   "revision":2,
   "creationTimeStamp":"2015-05-06T22:41:02.000-0400",
   "modifiedTimeStamp":"2015-05-07T00:15:47.000-0400",
   "name":"samplemodule"
}
```


Here is an example of a successfully compiled module with a warnings response body:


```
{
   "links":[
      {
         "method":"GET",
         "rel":"self",
```

```
      "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232",
      "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232",
      "type":"application/vnd.sas.microanalytic.module"
   },
   {
      "method":"GET",
      "rel":"up",
      "href":"http://www.example.com/SASMicroAnalyticService/rest/modules",
      "uri":"/modules",
      "type":"application/vnd.sas.collection"
   },
   {
      "method":"GET",
      "rel":"source",
      "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232/source",
      "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/source",
      "type":"application/vnd.sas.microanalytic.module.source"
   },
   {
      "method":"GET",
      "rel":"steps",
      "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
      "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
      "type":"application/vnd.sas.collection"
   },
   {
      "method":"PUT",
      "rel":"update",
      "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232",
      "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232",
      "type":"application/vnd.sas.microanalytic.module"
   },
   {
      "method":"DELETE",
      "rel":"delete",
      "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232",
      "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232"
   }
],
"description":"Expanded sample module",
"version":1,
"scope":"public",
"warnings":{
   "errorCode":0,
   "message":"Module compiled with warnings.",
   "details":[
      "In declaration of method copy_arrays: parameter out_charN_array is 'in_out';
       therefore, the type size (12) will be ignored.",
      "In declaration of method copy_arrays: parameter out_varchar_array is 'in_out';
       therefore, the type size (512) will be ignored.",
      "In declaration of method copy_charN_array: parameter out_array is 'in_out';
```

```
          therefore, the type size (12) will be ignored.",
         "In declaration of method produce_warnings: parameter out_string is 'in_out';
          therefore, the type size (12) will be ignored."
      ],
      "remediation":"",
      "links":[
      ],
      "version":1,
      "httpStatusCode":0
   },
   "id":"36af8e3c-6a37-4494-a8e0-9cc96ad62232",
   "steps":[
      "copy_arrays",
      "copy_bigint",
      "copy_bigint_array",
      "copy_char12",
      "copy_charN_array",
      "copy_float",
      "copy_float_array",
      "copy_int",
      "copy_int_array",
      "copy_scalars",
      "copy_varchar",
      "copy_varchar_array",
      "produce_warnings"
   ],
   "properties":[
     {
         "name" : "connectionString",
         "value" : "DRIVER=base;"
      }
   ],
   "revision":3,
   "creationTimeStamp":"2015-05-06T22:41:02.000-0400",
   "modifiedTimeStamp":"2015-05-07T00:22:19.000-0400",
   "name":"samplemodule"
}
```


Here is an example of an error response body:


```
{
   "errorCode":-33,
   "message":"Module name cannot be changed from a PUT operation.",
   "details":[
   ],
   "remediation":"",
   "links":[
   ],
   "version":1,
   "httpStatusCode":400
}
```


The DELETE method deletes all revisions of a module resource through the module ID.

The DELETE method requires authentication and has a request URL of DELETE http://*www.example.com*/SASMicroAnalyticService/modules/{*moduleId*}.

Here are the HTTP response codes:

204
No content

401
Unauthorized

403
Forbidden

404
Not found

500
Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

One situation that causes a 403 code to be returned is when the PUT is initiated from an untrusted site.

This operation returns the application/vnd.sas.error media type for failure. This media type is returned when the server cannot locate the module either because the module ID is incorrect, the module does not exist anymore, or the module cannot be deleted (for example, when another operation is taking place on this module).

Resource /modules/{moduleId}/source

The /modules/{*moduleId*}/source resource is the source code of the module.

The GET method returns the source code of a module. It requires authentication and has a request URL of GET http://*www.example.com*/SASMicroAnalyticService/modules/{*moduleId*}/source.

Here are the HTTP response codes:

200
OK

401
Unauthorized

404
Not found

500
Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

This operation returns the following media type representations by setting the Accept: header of the request:

- application/json
- application/vnd.sas.microanalytic.module.source+json

This operation returns the application/vnd.sas.error media type for failure. This media type is returned when the server encounters an error. An example of an error is when a node in a clustered deployment has become out of sync.

Here is an example of the JSON response:

```
{
```

```
   "moduleId":"fafbf5d4-01c0-48ea-a3e5-ef36fc3dfb64",
   "source":"ds2_options sas;package methods ;\n method echo_char(char in_string,
in_out char out_string);\n out_string=in_string;\n end;\n method echo_char12_implicit(char(12)
in_string, in_out char out_string);\n out_string=in_string;\n end;\n
method echo_char12_explicit(char(12) in_string, in_out char(12) out_string);\n
out_string=in_string;\n end;\n method echo_varchar_implicit(varchar(32767) in_string,
in_out varchar out_string);\n out_string=in_string;\n end;\n
method echo_varchar_explicit(varchar(32767) in_string, in_out varchar(32767) out_string);\n
out_string=in_string;\n end;\n method echo_bigint(bigint in_int, in_out bigint out_int);\n
out_int=in_int;\n end;\n method echo_float(double in_float, in_out double out_float);\n
out_float=in_float;\n end;\n method echo_int(int in_int, in_out int out_int);\n
out_int=in_int;\n end;\n method echo_scalars(char in_char, char(12) in_char12, varchar(32767)
in_varchar, int in_int,\n bigint in_bigint, double in_float, \n in_out char out_char,
in_out char(12) out_char12,\n in_out varchar out_varchar, in_out int out_int,\n
in_out bigint out_bigint, in_out double out_float);\n out_char = in_char;\n
out_char12 = in_char12;\n out_string=in_string;\n out_int=in_int;\n out_bigint=in_bigint;\n
out_float=in_float;\n end;\n method echo_char1_array(char  in_array[4],
in_out char out_array[4]);\n dcl int count;\n do count = 1 to 4;\n
out_array[count] = in_array[count];\n end;\n end;\n method echo_charN_array(char(12)
in_array[4], in_out char(12)  out_array[4]);\n dcl int count;\n do count = 1 to 4;\n
out_array[count] = in_array[count];\n end;\n end;\n method echo_int_array(int in_array[17],
in_out int out_array[37]);\n dcl int count;\n do count = 1 to 17;\n
out_array[count] = in_array[count];\n end;\n end;\n method echo_float_array(double in_array[2048],
in_out double out_array[2048]);\n dcl int count;\n do count = 1 to 2048;\n
out_array[count] = in_array[count];\n end;\n end;\n method echo_bigint_array(bigint in_array[1],
bigint out_array[1]);\n dcl int count;\n do count = 1 to 1;\n out_array[count] = in_array[count];\n
end;\n end;\n method echo_arrays(char  in_char1_array[4], \n char(12) in_charN_array[4],\n
varchar(512) in_varchar_array[1],\n int in_int_array[17], \n double in_double_array[2048], \n
bigint in_bigint_array[1], \n in_out char out_char1_array[4],\n in_out char(12)
 out_charN_array[4],\n in_out varchar(512) out_varchar_array[1],\n in_out int out_int_array[37],\n
in_out double out_double_array[2048],\n bigint out_bigint_array[1]);\n \n dcl int count;\n \n
do count = 1 to 4;\n out_char1_array[count] = in_char1_array[count];\n end;\n \n do count = 1 to 4;\n
out_charN_array[count] = in_charN_array[count];\n end;\n \n do count = 1 to 1;\n
out_varchar_array[count] = in_varchar_array[count];\n end;\n \n do count = 1 to 17;\n
out_int_array[count] = in_int_array[count];\n end;\n \n do count = 1 to 2048;\n
out_double_array[count] = in_double_array[count];\n end;\n \n do count = 1 to 1;\n
out_bigint_array[count] = in_bigint_array[count];\n end;\n \n end;\n \n endpackage;\n \n ",
   "links":[
      {
         "method":"GET",
         "rel":"self",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 fafbf5d4-01c0-48ea-a3e5-ef36fc3dfb64/source",
         "uri":"/modules/fafbf5d4-01c0-48ea-a3e5-ef36fc3dfb64/source",
         "type":"application/vnd.sas.microanalytic.module.source"
      },
      {
         "method":"GET",
         "rel":"up",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 fafbf5d4-01c0-48ea-a3e5-ef36fc3dfb64",
         "uri":"/modules/fafbf5d4-01c0-48ea-a3e5-ef36fc3dfb64",
         "type":"application/vnd.sas.microanalytic.module"
      }
   ],
   "version":1
```

```
}
```

Here is an example of an error response body:

```
{
  "errorCode": 4001,
  "message": "No module with the module ID a1511cb8-58b3-475a-a4d6-8a5817d936 exists.",
  "details": [],
  "remediation": "",
  "links": [],
  "version": 1,
  "httpStatusCode": 404
}
```

Collection /modules/{moduleId}/steps

The /modules/{*moduleId*}/steps collection is a collection of steps within a specific module that is loaded in memory by SAS Micro Analytic Service.

The /modules/{*moduleId*}/steps collection uses the GET method, which returns a resource collection of steps corresponding to a specific module. It requires authentication, and has a request URL of GET http://*www.example.com*/SASMicroAnalyticService/modules/{*moduleId*}/steps.

Here are the HTTP response codes:

200
: OK

401
: Unauthorized

404
: Not found

500
: Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

Here are the query parameters for /modules/{*moduleId*}/steps:

Name	Type	Description
?start	integer	The starting index of the first item in a page. The index is 0-based. Default is 0.
?limit	integer	The maximum number of steps to return in this page of results. The actual number of returned steps might be less if the collection has been exhausted. The default is 10.
?label	string	Filter by the name of the steps. Each step is checked if its name contains the label.

This operation returns the following media type representations by setting the Accept: header of the request:

- application/json

- application/vnd.sas.collection

This operation returns the application/vnd.sas.error media type for failure. This media type is returned when the server encounters an error. An example of an error is when a node in a clustered deployment has become out of sync.

Here is an example of the JSON response:


```
{
   "links":[
      {
         "method":"GET",
         "rel":"self",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
         "type":"application/vnd.sas.collection"
      },
      {
         "method":"GET",
         "rel":"first",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps?start=0&limit=10",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps?start=0&limit=10",
         "type":"application/vnd.sas.collection"
      },
      {
         "method":"GET",
         "rel":"next",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps?start=10&limit=10",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps?start=10&limit=10",
         "type":"application/vnd.sas.collection"
      },
      {
         "method":"GET",
         "rel":"last",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps?start=3&limit=10",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps?start=3&limit=10",
         "type":"application/vnd.sas.collection"
      },
      {
         "method":"GET",
         "rel":"up",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 36af8e3c-6a37-4494-a8e0-9cc96ad62232",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232",
         "type":"application/vnd.sas.microanalytic.module"
      }
   ],
   "name":"items",
   "accept":"application/vnd.sas.microanalytic.module.step",
   "start":0,
   "count":13,
   "items":[
      {
```

```
"links":[
   {
      "method":"GET",
      "rel":"self",
      "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
             36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_arrays",
      "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_arrays",
      "type":"application/vnd.sas.microanalytic.module.step"
   },
   {
      "method":"GET",
      "rel":"up",
      "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
             36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
      "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
      "type":"application/vnd.sas.collection"
   },
   {
      "method":"POST",
      "rel":"validate",
      "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
             36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_arrays/validations",
      "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_arrays/validations",
      "type":"application/vnd.sas.microanalytic.module.step.input.validity"
   },
   {
      "method":"POST",
      "rel":"execute",
      "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
             36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_arrays",
      "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_arrays",
      "type":"application/vnd.sas.microanalytic.module.step.output"
   }
],
"id":"copy_arrays",
"moduleId":"36af8e3c-6a37-4494-a8e0-9cc96ad62232",
"inputs":[
   {
      "name":"in_charN_array",
      "type":"stringArray",
      "dim":4,
      "size":12
   },
   {
      "name":"in_varchar_array",
      "type":"stringArray",
      "dim":1,
      "size":512
   },
   {
      "name":"in_int_array",
      "type":"integerArray",
      "dim":5,
      "size":0
   },
   {
```

```
            "name":"in_double_array",
            "type":"decimalArray",
            "dim":2,
            "size":0
         },
         {
            "name":"in_bigint_array",
            "type":"bigintArray",
            "dim":1,
            "size":0
         }
      ],
      "outputs":[
         {
            "name":"out_charN_array",
            "type":"stringArray",
            "dim":4,
            "size":12
         },
         {
            "name":"out_varchar_array",
            "type":"stringArray",
            "dim":1,
            "size":512
         },
         {
            "name":"out_int_array",
            "type":"integerArray",
            "dim":5,
            "size":0
         },
         {
            "name":"out_double_array",
            "type":"decimalArray",
            "dim":2,
            "size":0
         },
         {
            "name":"out_bigint_array",
            "type":"bigintArray",
            "dim":1,
            "size":0
         }
      ],
      "description":null,
      "version":1
   },
   {
      "links":[
         {
            "method":"GET",
            "rel":"self",
            "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                    36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_bigint",
            "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_bigint",
            "type":"application/vnd.sas.microanalytic.module.step"
```

```
      },
      {
         "method":"GET",
         "rel":"up",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
         "type":"application/vnd.sas.collection"
      },
      {
         "method":"POST",
         "rel":"validate",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_bigint/validations",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_bigint/validations",
         "type":"application/vnd.sas.microanalytic.module.step.input.validity"
      },
      {
         "method":"POST",
         "rel":"execute",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_bigint",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_bigint",
         "type":"application/vnd.sas.microanalytic.module.step.output"
      }
   ],
   "id":"copy_bigint",
   "moduleId":"36af8e3c-6a37-4494-a8e0-9cc96ad62232",
   "inputs":[
      {
         "name":"in_int",
         "type":"bigint",
         "dim":0,
         "size":0
      }
   ],
   "outputs":[
      {
         "name":"out_int",
         "type":"bigint",
         "dim":0,
         "size":0
      }
   ],
   "description":null,
   "version":1
},
{
   "links":[
      {
         "method":"GET",
         "rel":"self",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_bigint_array",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_bigint_array",
         "type":"application/vnd.sas.microanalytic.module.step"
```

```
      },
      {
         "method":"GET",
         "rel":"up",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
         "type":"application/vnd.sas.collection"
      },
      {
         "method":"POST",
         "rel":"validate",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_bigint_array/validations",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_bigint_array/validations",
         "type":"application/vnd.sas.microanalytic.module.step.input.validity"
      },
      {
         "method":"POST",
         "rel":"execute",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_bigint_array",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_bigint_array",
         "type":"application/vnd.sas.microanalytic.module.step.output"
      }
   ],
   "id":"copy_bigint_array",
   "moduleId":"36af8e3c-6a37-4494-a8e0-9cc96ad62232",
   "inputs":[
      {
         "name":"in_array",
         "type":"bigintArray",
         "dim":1,
         "size":0
      },
      {
         "name":"out_array",
         "type":"bigintArray",
         "dim":1,
         "size":0
      }
   ],
   "outputs":null,
   "description":null,
   "version":1
},
{
   "links":[
      {
         "method":"GET",
         "rel":"self",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_char12",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_char12",
         "type":"application/vnd.sas.microanalytic.module.step"
      },
```

```
      {
         "method":"GET",
         "rel":"up",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
         "type":"application/vnd.sas.collection"
      },
      {
         "method":"POST",
         "rel":"validate",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_char12/validations",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_char12/validations",
         "type":"application/vnd.sas.microanalytic.module.step.input.validity"
      },
      {
         "method":"POST",
         "rel":"execute",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_char12",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_char12",
         "type":"application/vnd.sas.microanalytic.module.step.output"
      }
   ],
   "id":"copy_char12",
   "moduleId":"36af8e3c-6a37-4494-a8e0-9cc96ad62232",
   "inputs":[
      {
         "name":"in_string",
         "type":"string",
         "dim":0,
         "size":12
      }
   ],
   "outputs":[
      {
         "name":"out_string",
         "type":"string",
         "dim":0,
         "size":0
      }
   ],
   "description":null,
   "version":1
},
{
   "links":[
      {
         "method":"GET",
         "rel":"self",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_charN_array",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_charN_array",
         "type":"application/vnd.sas.microanalytic.module.step"
      },
```

```
      {
         "method":"GET",
         "rel":"up",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
         "type":"application/vnd.sas.collection"
      },
      {
         "method":"POST",
         "rel":"validate",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_charN_array/validations",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_charN_array/validations",
         "type":"application/vnd.sas.microanalytic.module.step.input.validity"
      },
      {
         "method":"POST",
         "rel":"execute",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_charN_array",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_charN_array",
         "type":"application/vnd.sas.microanalytic.module.step.output"
      }
   ],
   "id":"copy_charN_array",
   "moduleId":"36af8e3c-6a37-4494-a8e0-9cc96ad62232",
   "inputs":[
      {
         "name":"in_array",
         "type":"stringArray",
         "dim":4,
         "size":12
      }
   ],
   "outputs":[
      {
         "name":"out_array",
         "type":"stringArray",
         "dim":4,
         "size":12
      }
   ],
   "description":null,
   "version":1
},
{
   "links":[
      {
         "method":"GET",
         "rel":"self",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_float",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_float",
         "type":"application/vnd.sas.microanalytic.module.step"
      },
```

```
      {
         "method":"GET",
         "rel":"up",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
         "type":"application/vnd.sas.collection"
      },
      {
         "method":"POST",
         "rel":"validate",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_float/validations",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_float/validations",
         "type":"application/vnd.sas.microanalytic.module.step.input.validity"
      },
      {
         "method":"POST",
         "rel":"execute",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_float",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_float",
         "type":"application/vnd.sas.microanalytic.module.step.output"
      }
   ],
   "id":"copy_float",
   "moduleId":"36af8e3c-6a37-4494-a8e0-9cc96ad62232",
   "inputs":[
      {
         "name":"in_float",
         "type":"decimal",
         "dim":0,
         "size":0
      }
   ],
   "outputs":[
      {
         "name":"out_float",
         "type":"decimal",
         "dim":0,
         "size":0
      }
   ],
   "description":null,
   "version":1
},
{
   "links":[
      {
         "method":"GET",
         "rel":"self",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_float_array",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_float_array",
         "type":"application/vnd.sas.microanalytic.module.step"
      },
```

```
      {
         "method":"GET",
         "rel":"up",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
         "type":"application/vnd.sas.collection"
      },
      {
         "method":"POST",
         "rel":"validate",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_float_array/validations",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_float_array/validations",
         "type":"application/vnd.sas.microanalytic.module.step.input.validity"
      },
      {
         "method":"POST",
         "rel":"execute",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_float_array",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_float_array",
         "type":"application/vnd.sas.microanalytic.module.step.output"
      }
   ],
   "id":"copy_float_array",
   "moduleId":"36af8e3c-6a37-4494-a8e0-9cc96ad62232",
   "inputs":[
      {
         "name":"in_array",
         "type":"decimalArray",
         "dim":2,
         "size":0
      }
   ],
   "outputs":[
      {
         "name":"out_array",
         "type":"decimalArray",
         "dim":2,
         "size":0
      }
   ],
   "description":null,
   "version":1
},
{
   "links":[
      {
         "method":"GET",
         "rel":"self",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_int",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_int",
         "type":"application/vnd.sas.microanalytic.module.step"
      },
```

```
      {
         "method":"GET",
         "rel":"up",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
         "type":"application/vnd.sas.collection"
      },
      {
         "method":"POST",
         "rel":"validate",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_int/validations",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_int/validations",
         "type":"application/vnd.sas.microanalytic.module.step.input.validity"
      },
      {
         "method":"POST",
         "rel":"execute",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_int",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_int",
         "type":"application/vnd.sas.microanalytic.module.step.output"
      }
   ],
   "id":"copy_int",
   "moduleId":"36af8e3c-6a37-4494-a8e0-9cc96ad62232",
   "inputs":[
      {
         "name":"in_int",
         "type":"integer",
         "dim":0,
         "size":0
      }
   ],
   "outputs":[
      {
         "name":"out_int",
         "type":"integer",
         "dim":0,
         "size":0
      }
   ],
   "description":null,
   "version":1
},
{
   "links":[
      {
         "method":"GET",
         "rel":"self",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_int_array",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_int_array",
         "type":"application/vnd.sas.microanalytic.module.step"
      },
```

```
      {
         "method":"GET",
         "rel":"up",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
         "type":"application/vnd.sas.collection"
      },
      {
         "method":"POST",
         "rel":"validate",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_int_array/validations",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_int_array/validations",
         "type":"application/vnd.sas.microanalytic.module.step.input.validity"
      },
      {
         "method":"POST",
         "rel":"execute",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_int_array",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_int_array",
         "type":"application/vnd.sas.microanalytic.module.step.output"
      }
   ],
   "id":"copy_int_array",
   "moduleId":"36af8e3c-6a37-4494-a8e0-9cc96ad62232",
   "inputs":[
      {
         "name":"in_array",
         "type":"integerArray",
         "dim":5,
         "size":0
      }
   ],
   "outputs":[
      {
         "name":"out_array",
         "type":"integerArray",
         "dim":5,
         "size":0
      }
   ],
   "description":null,
   "version":1
},
{
   "links":[
      {
         "method":"GET",
         "rel":"self",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_scalars",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_scalars",
         "type":"application/vnd.sas.microanalytic.module.step"
      },
```

```
    {
       "method":"GET",
       "rel":"up",
       "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
       "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
       "type":"application/vnd.sas.collection"
    },
    {
       "method":"POST",
       "rel":"validate",
       "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_scalars/validations",
       "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_scalars/validations",
       "type":"application/vnd.sas.microanalytic.module.step.input.validity"
    },
    {
       "method":"POST",
       "rel":"execute",
       "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_scalars",
       "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_scalars",
       "type":"application/vnd.sas.microanalytic.module.step.output"
    }
 ],
 "id":"copy_scalars",
 "moduleId":"36af8e3c-6a37-4494-a8e0-9cc96ad62232",
 "inputs":[
    {
       "name":"in_char12",
       "type":"string",
       "dim":0,
       "size":12
    },
    {
       "name":"in_varchar",
       "type":"string",
       "dim":0,
       "size":32767
    },
    {
       "name":"in_int",
       "type":"integer",
       "dim":0,
       "size":0
    },
    {
       "name":"in_bigint",
       "type":"bigint",
       "dim":0,
       "size":0
    },
    {
       "name":"in_float",
       "type":"decimal",
       "dim":0,
```

```
            "size":0
         }
      ],
      "outputs":[
         {
            "name":"out_char",
            "type":"string",
            "dim":0,
            "size":0
         },
         {
            "name":"out_char12",
            "type":"string",
            "dim":0,
            "size":0
         },
         {
            "name":"out_varchar",
            "type":"string",
            "dim":0,
            "size":0
         },
         {
            "name":"out_int",
            "type":"integer",
            "dim":0,
            "size":0
         },
         {
            "name":"out_bigint",
            "type":"bigint",
            "dim":0,
            "size":0
         },
         {
            "name":"out_float",
            "type":"decimal",
            "dim":0,
            "size":0
         }
      ],
      "description":null,
      "version":1
   }
 ],
 "limit":10,
 "version":1
}
```


Here is an example error response:


```
{
  "errorCode": 4001,
  "message": "No module with the module ID a1511cb8-58b3-475a-a4d6-8a5817d936 exists.",
  "details": [],
  "remediation": "",
  "links": [],
```

```
  "version": 1,
  "httpStatusCode": 404
}
```

Resource /modules/{moduleId}/steps/{stepId}

The /modules/{*moduleId*}/steps/{*stepId*} resource is a single step of a compiled module.

The /modules/{*moduleId*}/steps/{*stepId*} collection uses the GET method. It returns detailed information about input and output signatures used to execute a specific step of the module. It requires authentication, and has a request URL of GET http://*www.example.com*/SASMicroAnalyticService/rest/modules/{*moduleId*}/steps/{*stepId*}.

Here are the HTTP response codes:

200
: OK

401
: Unauthorized

404
: Not found

500
: Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

This operation returns the following media type representations by setting the Accept: header of the request:

- application/json
- application/vnd.sas.microanalytic.module.step+json

This operation returns the application/vnd.sas.error media type for failure. This media type is returned when the module cannot be located, either because the module ID is incorrect or the module does not exist anymore. This media type is also returned when the module ID corresponds to an existing module. However, the step ID is incorrect.

Here is an example of the JSON response:

```
{
    "id": "test_all_types",
    "moduleId": "8eee3045-83fa-4725-88ef-471ddb5ac4f9",
    "description": null,
    "inputs": [
      {
        "name": "in_string",
        "type": "string",
        "dim": 0,
        "size": 32767
      },
      {
        "name": "in_bigint",
        "type": "bigint",
        "dim": 0,
        "size": 0
      },
```

```
    {
      "name": "in_int",
      "type": "integer",
      "dim": 0,
      "size": 0
    },
    {
      "name": "in_double ",
      "type": "decimal",
      "dim": 0,
      "size": 0
    }
  ],
  "outputs": [
    {
      "name": "out_string",
      "type": "string",
      "dim": 0,
      "size": 8
    },
    {
      "name": "out_bigint",
      "type": "bigint",
      "dim": 0,
      "size": 0
    },
    {
      "name": "out_int",
      "type": "integer",
      "dim": 0,
      "size": 0
    },
    {
      "name": "out_double",
      "type": "decimal",
      "dim": 0,
      "size": 0
    },
    {
      "name": "string_arr",
      "type": "stringArray",
      "dim": 3,
      "size": 32767
    },
    {
      "name": "bigint_arr",
      "type": "bigIntArray",
      "dim": 3,
      "size": 0
    },
    {
      "name": "int_arr",
      "type": "intArray",
      "dim": 3,
      "size": 0
    },
```

```
      {
        "name": "double_arr",
        "type": "decimalArray",
        "dim": 3,
        "size": 0
      }
  ],
  "links": [
      {
        "method": "GET",
        "rel": "self",
        "href": "http://www.example.com/modules/8eee3045-83fa-4725-88ef-471ddb5ac4f9/
                 steps/test_all_types",
        "uri": "/modules/8eee3045-83fa-4725-88ef-471ddb5ac4f9/steps/test_all_types",
        "type": "application/vnd.sas.microanalytic.module.step"
      },
      {
        "method": "GET",
        "rel": "up",
        "href": "http://www.example.com/modules/8eee3045-83fa-4725-88ef-471ddb5ac4f9/
                 steps/test_all_types",
        "uri": "/modules/8eee3045-83fa-4725-88ef-471ddb5ac4f9/steps",
        "type": "application/vnd.sas.collection"
      },
      {
        "method": "POST",
        "rel": "validate",
        "href": "http://www.example.com/modules/8eee3045-83fa-4725-88ef-471ddb5ac4f9/
                 steps/test_all_types/validations",
        "uri": "/modules/8eee3045-83fa-4725-88ef-471ddb5ac4f9/steps/test_all_types/validations",
        "type": "application/vnd.sas.microanalytic.module.step.input.validity"
      },
      {
        "method": "POST",
        "rel": "execute",
        "href": "http://www.example.com/modules/8eee3045-83fa-4725-88ef-471ddb5ac4f9/
                 steps/test_all_types",
        "uri": "/modules/8eee3045-83fa-4725-88ef-471ddb5ac4f9/steps/test_all_type",
        "type": "application/vnd.sas.microanalytic.module.step.output"
      }
  ],
  "version": 1
}
```


Here is an example of an error response:


```
{
  "errorCode": 4001,
  "message": "No module with the module ID a1511cb8-58b3-475a-a4d6-8a5817d936 exists.",
  "details": [],
  "remediation": "",
  "links": [],
  "version": 1,
  "httpStatusCode": 404
}
```

There are two POST methods. The first POST method validates step inputs. The request body for each POST contains the input values that are used to execute the steps. The input values are validated against the expected input signature of the step. The POST methodrequires authentication, and has a request URL of POST http://*www.example.com*/SASMicroAnalyticService/rest/modules/{*moduleId*}/steps/{*stepId*}/validations.

Here are the HTTP response codes:

200
OK

400
Bad Request

401
Unauthorized

403
Forbidden

404
Not found

500
Server error.

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

One situation that causes a 403 code to be returned is when the POST is initiated from an untrusted site.

Here is an example of the JSON request:

```
{
  "inputs": [
    {
      "name": "in_string",
      "value": "This is a test..."
    },
    {
      "name": "in_bigint",
      "value": 987654321
    },
    {
      "name": "in_int",
      "value": 7654321
    },
    {
      "name": "in_double",
      "value": 0.9997
    }
  ]
}
```

This operation accepts the following media type representations by setting the Content-Type: header of the request:

- application/json
- application/vnd.sas.microanalytic.module.step.input+json

This operation returns the following media type representations by setting the Accept: header of the request:

- application/json
- application/vnd.sas.microanalytic.module.step.input.validity+json

This operation returns the application/vnd.sas.error media type for failure. This media type is returned whenever there is an error in performing the validation, not when the input parameter is invalid.

Here is an example of the JSON response:

```
{
  "moduleId": "052209DE-DF4D-6D44-B469-9094AC95F18E",
  "stepId": "test_all_types",
  "version": 1,
  "results": {},
  "valid": true
}
```

Here is an example response body for an instance when an input value is invalid:

```
{
  "moduleId": "052209DE-DF4D-6D44-B469-9094AC95F18E",
  "stepId": "test_all_types",
  "version": 1,
  "results": {
    "in_integer ": "Integer value expected but found 0.9997."
  },
  "valid": false
}
```

The second POST method executes a step. This method creates the output from executing a step on the provided input values. The request body contains the input values. The response body contains the results as output values. This POST method has a request URL of POST http://*www.example.com*/SASMicroAnalyticService/rest/modules/{*moduleId*}/steps/{*stepId*} .

Here are the HTTP response codes:

200
: OK

400
: Bad Request

401
: Unauthorized

403
: Forbidden

404
: Not found

500
: Server error.

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

One situation that causes a 403 code to be returned is when the POST is initiated from an untrusted site.

Here is an example of the JSON request:

```
{
  "inputs": [
    {
      "name": "in_string",
      "value": "This is a test..."
    },
    {
      "name": "in_bigint",
      "value": 987654321
    },
    {
      "name": "in_int",
      "value": 7654321
    },
    {
      "name": "in_double ",
      "value": 0.9997
    }
  ]
}
```

This operation accepts the following media type representations by setting the Content-Type: header of the request:

- application/json
- application/vnd.sas.microanalytic.module.step.input+json

This operation returns the following media type representations by setting the Accept: header of the request:

- application/json
- application/vnd.sas.microanalytic.module.step.output+json

This operation might return the following media types for failure:

application/vnd.sas.microanalytic.module.step.input.validity+json
: This media type is returned when the input is invalid.

application/vnd.sas.error
: This media type is returned when there is problem executing the step.

Here is an example of the JSON response:

```
{
  "moduleId": "0BCA724F-53D7-3540-8A62-4E2731D69813",
  "stepId": "test_all_types",
  "output": [
    {
      "name": "out_string",
      "value": "This is a test..."
    },
    {
      "name": "out_bigint",
      "value": 987654321
    },
    {
      "name": "out_int",
      "value": 7654321
```

```
    },
    {
      "name": "out_double",
      "value": 0.9997
    },
    {
      "name": "string_arr",
      "value": [
        "This is a test...",
        "This is a test...",
        "This is a test..."
      ]
    },
    {
      "name": "bigint_arr",
      "value": [
        987654321,
        987654321,
        987654321
      ]
    },
    {
      "name": "int_arr",
      "value": [
        7654321,
        7654321,
        7654321
      ]
    },
    {
      "name": "double_arr",
      "value": [
        0.9997,
        0.9997,
        0.9997
      ]
    }
  ],
  "version": 1
}
```

Here is an example response body for the instances when the input is invalid:

```
{
  "moduleId": "0BCA724F-53D7-3540-8A62-4E2731D69813",
  "stepId": "test_all_types",
  "version": 1,
  "results": {
    "in_double ": "Integer value expected but found 0.9997."
  },
  "valid": false
}
```

Here is an example error response:

```
{
   "errorCode":-1958744015,
   "message":"Step ID echo_arrays failed to execute.",
```

```
   "details":[
      "Method not found."
   ],
   "remediation":"",
   "links":[],
   "version":1,
   "httpStatusCode":400
}
```

Chapter 7

Administration

SAS Micro Analytic Service Logging

An optional SAS Micro Analytic Service start-up parameter specifies the location of an XML logging configuration file, which controls the logging levels and the location of the log file or files. SAS Micro Analytic Service uses the SAS 9.4 Logging Facility. For more information, see *SAS 9.4 Logging: Configuration and Programming Reference*. Your SAS solution might provide a default logging configuration file, and that file might include loggers or appenders in addition to those described in this chapter. For example, on UNIX the file might be **`/data1/SAS/config/Lev1/Web/Common/LogConfig/SASMicroAnalyticService-log4sas.xml`**. For more information, see your solution's documentation.

SAS Micro Analytic Service uses two loggers named App.tk.MAS and App.tk.MAS.CodeGen. Code that is hosted by SAS Micro Analytic Service, or the functions that it calls, can use additional loggers.

The logger App.tk.MAS is used for logging all aspects of SAS Micro Analytic Service operation besides code compilation and code generation, which use App.tk.MAS.CodeGen. Normal operations, such as start-up and shutdown, are logged at the INFO level. Detailed information about such operations as compilation start and finish, and others, are logged at the DEBUG level. Warning and error conditions are logged at the WARN or ERROR levels, as appropriate. By default, App.tk.MAS is set to the INFO level.

App.tk.MAS.CodeGen is used for logging compiler-generated messages, such as compilation warnings and errors. Compiler messages can also be retrieved programmatically through the Java and REST interfaces. (See getCompilationMessages in "Method Descriptions" on page 33.) Your SAS solution might report compilation messages automatically. Because these messages are available programmatically, and to prevent compiler messages from cluttering the log, App.tk.MAS.CodeGen is set to the FATAL logging level by default.

Secure DS2 HTTP Package Usage

The DS2 HTTP package supports HTTP and HTTPS endpoints. The configuration of SAS Micro Analytic Service defines the SSLCALISTLOC environment variable, which specifies the location of the digital certificates for trusted certificate authorities.

The SSLCALISTLOC environment variable is defined in a host-specific configuration script that is located in the application server's bin directory. For example, a UNIX platform **SAS-configuration-directory/*LevN*/Web/WebAppServer/SASServer13_1/bin/setenv.sh** defines SSLCALISTLOC with a value of SSLCALISTLOC=$JRE_HOME/../../../SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.pem. For more information about SSLCALISTLOC, see *Encryption in SAS 9.4*.

When an HTTP endpoint requires client authentication, it responds to the client with its list of supported authentication mechanisms. The DS2 HTTP package currently supports two of the three most common authentication mechanisms. It supports Basic and Negotiate, but does not support the Digest mechanism. Because Basic authentication in itself does not provide any credential confidentiality, it should be used only when the data is being encrypted through TLS. The DS2 HTTP package does not provide an interface allowing the user to specify credentials, other than including them in the URL. An example is http://username:password@example.com/. The Negotiate mechanism supports Kerberos and, when it is used on Windows, NTLM is also supported. For more information, see “Using the HTTP Package” in *SAS 9.4 DS2 Language Reference*.

Monitoring

Monitoring SAS Micro Analytic Service

SAS Micro Analytic Service provides several logs to help you with monitoring. One of these is the web server error log located at **SAS/config/*LevN*/Web/WebServer/logs**. These logs have a filename format of error_yyyy-mm-dd.number.log. In them you can find connection errors between the web server and the tcServer.

The tcServer log is called SAS/config/*LevN*/Web/WebAppServer/SASServer13_X/logs/server.log. To determine whether the tcServer has started, look for a message similar to the following:

```
2015-06-03 16:43:13,176 INFO  (main) [org.apache.catalina.startup.Catalina]
Server startup in 36647 ms.
```

The catalina.out file captures the output to the console. The content is identical to the entries that are logged in the REST service log file. Whether information should be sent to console is controlled by the Log4j configuration file of the REST service.

The gemfire.log file in **SAS/config/*LevN*/Web/WebAppServer/SASServer13_X/logs** logs the activity of GemFire, which is a third-party distributed data management platform. When the tcServer does not start up, check gemfire.log to see whether GemFire is waiting for data availability. Look for a log entry in a form that is similar to the following:

```
[info 2015/06/04 15:44:09.187 EDT  <localhost-startStop-1> tid=0x15]
Region /sas_gemfire_region_surrogatekeytomodulereplica initialized with data
```

```
from /10.xx.xxx.yy:/data1/SAS/config/Lev1/Web/WebAppServer/SASServer13_1/logs
created at timestamp 1433364173611 version 0 diskStoreId 19892c25-b655-4ae7-96ed-c978dde636d2
is waiting for the data previously hosted at
[/10.xx.xxx.xx:/data/SAS/config/Lev1/Web/WebAppServer/SASServer13_1/logs created
at timestamp 1433364164520 version 0 diskStoreId 20d2f45e-876f-4cc1-84b0-ccf6920da3e8]
to be available
```

The wait will eventually time out, and SAS Micro Analytic Service will not start correctly. This is most likely to happen in a clustered environment. For more information, see "Cluster Deployment for SAS Micro Analytic Service" on page 124.

The REST service log file is located at **SAS/config/*LevN*/Web/Logs/SASServer13_1/SASMicroAnalyticService1.2.log**. The current day log entries are in that file. The first log entry that occurs after midnight causes the previous day's log file to roll over to another file with the format SASMicroAnalyticService1.2.log.yyyy-mm-dd. SASMicroAnalyticService1.2.log is created fresh with the first log entry. The service logs are at INFO level. Therefore, they capture start-up entries, module creation, update and deletion boundary entries, as well as errors from all operations. When there is an error, and more information must be captured to identify the cause of the error, update the REST service's Log4j configuration file to set logging level to DEBUG, and restart the service.

Log entries are tagged with an INFO, WARN, or ERROR keyword. When the REST service is started properly, there is no entry with the ERROR keyword added to the log file. When a web service request is processed successfully, the HTTP status returned is either 200, 201 or 204, depending on the context. If the HTTP status returned is 4XX (such as 400, 401, 404) or 5XX (such as 503), an error message is included in the HTTP response body. In addition, one or more ERROR entries are in the log file.

A related log file in the same directory is the SAS Micro Analytic Service log. The filename has the format SASMicroAnalyticService1.2MAS.log.yyyy-mm-dd.pid. Pid is the process ID of the JVM process that hosts SAS Micro Analytic Service. Each time the REST service restarts, a new log file is used and then the log file rolls over to another file at midnight. The SAS Micro Analytic Service log file can capture compilation errors of modules, as well as any anomaly that is encountered by the SAS Micro Analytic Service.

The application's Log4j configuration file is in the directory **SAS/config/*LevN*/Web/Common/LogConfig**. The configuration file for the REST service log is SASMicroAnalyticService-log4j.xml. The configuration file for SAS Micro Analytic Service is SASMicroAnalyticService-log4sas.xml.

Monitoring SAS Micro Analytic Service Using SAS Environment Manager

Overview

SAS Environment Manager provides several pieces of monitoring functionality that can be used to help understand SAS Micro Analytic Service usage, check service availability, and set custom alerts.

Initialize SAS Environment Manager

To initialize SAS Environment Manager:

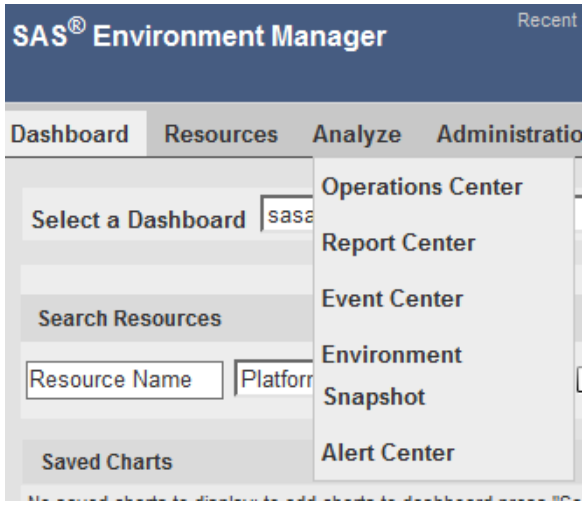
1. Open the file /config/*LevN*/Web/SASEnvironmentManager/emi-framework/ConfigureFiles/Kits/WebServer/WebServer.properties.
2. Make sure that kitenabled is set to TRUE.

3. Follow the instructions found inside the file /config/*LevN*/Web/SASEnvironmentManager/emi-framework/SAS_Environment_Manager_Service_Architecture_Quickstart.pdf.

Access a Report

To access reports in SAS Environment Manager:

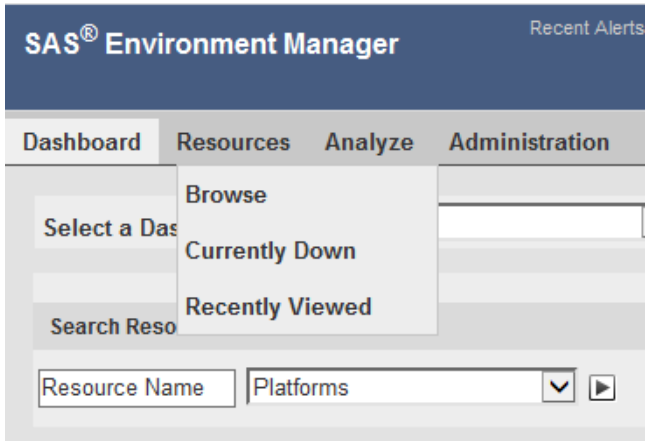
1. Open SAS Environment Manager inside a browser (SAS Environment Manager default port is 7080).
2. Select **Report Center** from the **Analyze** drop-down menu.



3. Navigate to **Stored Processes** ⇨ **Products** ⇨ **SAS Environment Manager** ⇨ **Kits** ⇨ **Web Server**. Click **HTTP Web Server return codes**.
4. To see all of the TKMAS HTTP requests with response codes, navigate to **Classification Variables** and move **clientsrc** from **Available** to **Selected**.
5. Under **Tabulate Report**, click **Subsets**. Set the **Where clause to filter SAS Environment Manager Data Mart table** to clientsubsrc = 'SASMicroAnalyticService'.
6. Click **Run** to see the report.

Monitor SAS Micro Analytic Service Downtime

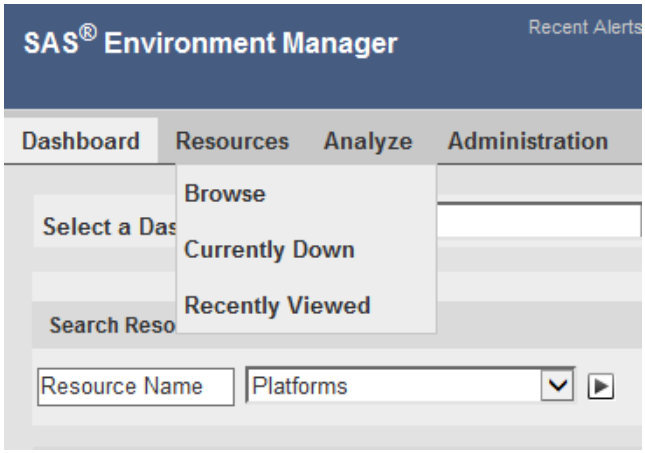
To monitor SAS Micro Analytic Service downtime, select **Currently Down** from the **Resources** drop-down menu. This provides you with a list of all of the resources that are currently down.



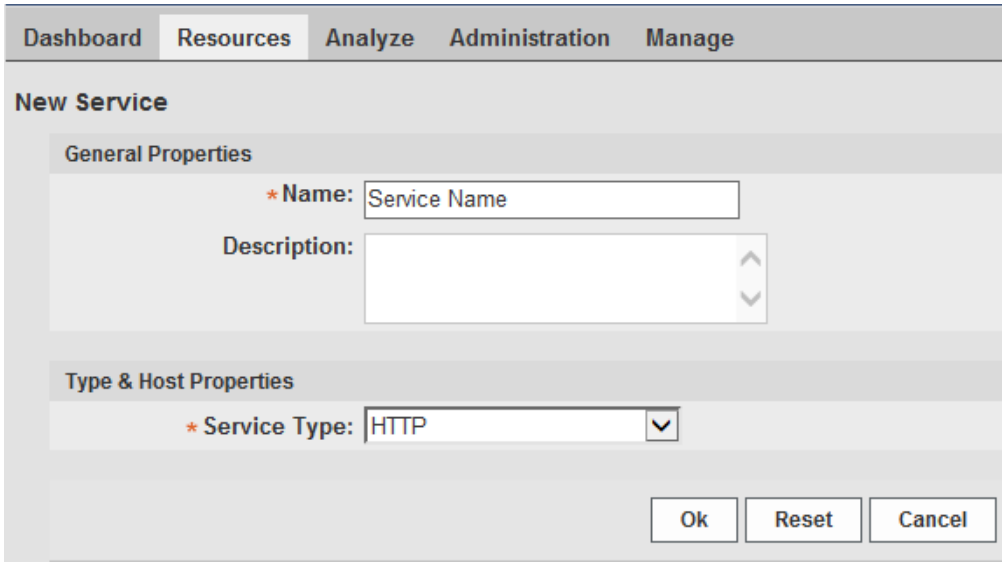
Set Alerts

To set up custom alerts for SAS Micro Analytic Service servers:

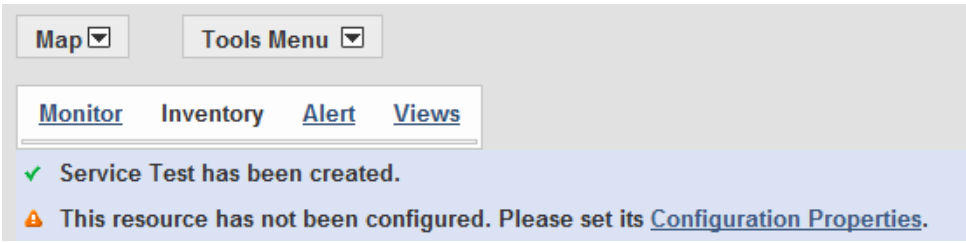
1. Select **Browse** from the **Resources** drop-down menu.



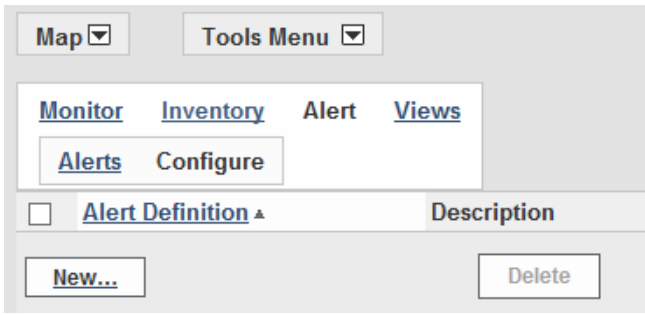
2. On the **Platforms** tab, click the platform where SAS Micro Analytic Service is installed.
3. Select **New Platform Service** from the **Tools** Menu.
4. Enter a name for the new service, and select **HTTP** from the **Service Type** drop-down menu. Click **OK**.



5. You should receive two messages on the service window. The first should tell you that your service has been created. The second should ask you to set the configuration properties. Click **Configuration Properties** in the second message.



6. Under **Configuration Properties**, set the following:
   a. Set the **port** field. The default is 7980.
   b. Set the **hostname** field to the location where SAS Micro Analytic Service is installed.
   c. Set the **path** field to /SASMicroAnalyticService.
   d. Select **GET** from the **method** drop-down menu.
   e. Click **OK**.
7. Click **Alert** and then **Configure**.
8. Click **New**.



9. Provide the information about the New Alert Definition window. Click **OK**.

When the condition that is specified for the alert is satisfied, an alert should be visible on the top banner of SAS Environment Manager.

Chapter 8

Deployment and Tuning

Deploying SAS Micro Analytic Service

The full SAS Micro Analytic Service software stack, including the REST, Java, and C interfaces, and the core C engine, is deployed as a SAS web application in SAS Web Application Server. SAS web applications can be clustered and tuned for performance and high availability. For information about how to tune the SAS Micro Analytic Service web application for optimum performance, see *SAS 9.4 Web Applications Tuning for Performance and Scalability*.

After deployment, validate the web service URL for the SAS Micro Analytic Service REST API by following the instructions found in SAS/config/LevN/documents/Instructions.html. Follow the steps found in the topic about validation. If the service is deployed correctly, the following JSON object is returned:

```
{"version":1,"links":[{"method":"GET","rel":"modules",
"href":"http://www.example.com/SASMicroAnalyticService/rest/modules","uri":"/modules"},
{"method":"POST","rel":"createModule","href":
"http://www.example.com/SASMicroAnalyticService/rest/modules","uri":"/modules"}]}
```

To grant access to the service to a user, add that user as a member of the Decision Manager Users group.

1. In SAS Management Console, expand **Environment Manager**.
2. Right-click **User Manager**, and click **New** ⇨ **User**.
3. On the **General** tab, enter the name and any other optional information.
4. On the **Groups and Roles** tab, find the **Decision Manager Users** group from the **Available Groups and Roles** list and add it to the **Member of** list.
5. On the **Accounts** tab, click **New**.

6. In the New Login Properties dialog box, you must complete at least the **User ID** field. Click **OK**.
7. Click **OK** in the New User Properties dialog box.

Cluster Deployment for SAS Micro Analytic Service

In a cluster deployment, the web server runs on only one node, and it serves as the balancer. The URL to the service sends the request to the web server. The web server dispatches the requests in round-robin to the nodes in the cluster, unless a different policy is specified in the web server configuration.

The metadata server for a middle-tier node is specified during deployment. The same metadata server that is referenced by the middle tier can be referenced by a middle-tier node. When that is the case, user management data and application properties that are set on the middle tier are applicable automatically to the middle-tier node. If different metadata servers are referenced by the middle tier and the middle-tier nodes, then any user and application management data changes should be made in both metadata servers.

By contrast with the middle tier, the Instructions.html file for the middle-tier node includes neither a web service URL, nor a section on validating steps for the web service. The web server directs requests to the middle-tier node based on the specified load-balancing policy in its configuration.

If a user wants to use the same node to serve a group of requests, this can be achieved by including the same route information in the HTTP request for that group of requests. The cluster is enabled for a sticky session by default. When a service request is made, the header section of the HTTP response includes a Set-Cookie header, such as the following:

```
Set-Cookie: c74b1b873e98ef08505dee685863e7b2_Cluster13=EC5213E970F0655
8E63F145001F64CEC.c74b1b873e98ef08505dee685863e7b2_SASServer13_1;
Path=/SASMicroAnalyticService/; HttpOnly
```

The first item is a variable=value construct. The variable is a session ID. The value is a route.

To use the same node to serve a group of requests, extract the route information from the first request of the group. From the second request to the last request, set the cookie header with the sessionID and route value, similar to the following example:

```
EC5213E970F06558E63F145001F64CEC.c74b1b873e98ef08
505dee685863e7b2_SASServer13_1
```

Using the same node to serve a group of requests can be useful because it avoids introducing errors by a delay in replicating content from one cluster node to another.

For example, the cluster consists of two nodes, Node 1 and Node 2. You want to deploy two modules, A and B. Also, B depends on A. Suppose A is a very big module and takes more than 20 seconds to compile. If A is deployed on Node 1, it must be replicated to Node 2 and then compiled on Node 2, before it is available on Node 2. If B is deployed to Node 2 before A is ready there, there is an error. To avoid this type of error, set the cookie to tell the web server to use Node 1 to deploy B.

Clustering relies on GemFire, a third-party distributed data management platform. GemFire persists data to files that are stored in **`SAS/config/LevN/Web/`**

`WebAppServer/SASServer13_X/logs`. The filenames contain the masgemfire sub-string. Those files should be left alone. Also, make sure that enough disk space is allocated to the **`SAS/config/`*`LevN`*`/Web/WebAppServer/SASServer13_X/logs`** directory so that the cache files grow.

CAUTION:

These files should not be truncated or deleted regardless of their size. Sometimes they might appear to be zero bytes. GemFire also uses the word BACKUP in some of the filenames. Deleting or truncating these files deletes the modules repository.

In a typical deployment, a middle-tier node uses the middle tier's GemFire locator. A locator is used in the peer-to-peer cache to discover other processes. If the whole cluster must be restarted, the commands to start the middle tier and middle-tier node should be submitted immediately one after another. The order does not matter.

Tuning SAS Micro Analytic Service

Adjust Thread Pool Size

Tasks in SAS Micro Analytic Service, such as revision compilations and method executions, are performed by special worker threads, which are part of the SAS threaded kernel architecture. These worker threads are maintained in a thread pool. The size of the thread pool to use is provided to SAS Micro Analytic Service as a start-up parameter. By default, the thread pool size is set to 4. Optimum performance is usually achieved by setting the thread pool size about equal to the number of cores in the hosting server. However, the optimum setting might vary depending on the characteristics of the programs that are run by SAS Micro Analytic Service.

To change the worker thread pool size:

1. In SAS Management Console, expand **Application Management**.
2. Expand **SAS Application Infrastructure**.
3. Right-click **SAS Micro Analytic Service 1.2**.
4. Select **Properties**.
5. Click the **Advanced** tab.
6. Unlock masintf.tk.threads in the **Property Name** column.
7. Change the value. To tell SAS Micro Analytic Service to automatically set the worker thread pool size equal to the number of server cores, enter 0 (zero) for the value.
8. Click **OK**.

Adjust Serial or Parallel Content Creation

The POST operation on the modules collection and the PUT and DELETE operations on a module are serialized by default. They are taken in the order of arrival to the REST server's processing queue, one after another is done. To adjust this setting to allow them to be done in parallel:

1. In SAS Management Console, expand **Application Management**.

2. Expand **SAS Application Infrastructure**.
3. Right-click **SAS Micro Analytic Service 1.2**.
4. Select **Properties**.
5. Click the **Advanced** tab.
6. Unlock masintfc.tk.serializecontentcreation in the **Property Name** column.
7. Change the value. The choices are true and false. The default value is True.
8. Click OK.

Adjust DS2 Module Compilation Mode

The REST server always inserts a DS2 option in front of a DS2 module to force it to be compiled in SAS mode. You can stop this behavior by changing a property:

1. In SAS Management Console, expand **Application Management**.
2. Expand **SAS Application Infrastructure**.
3. Right-click **SAS Micro Analytic Service 1.2**.
4. Select **Properties**.
5. Click the **Advanced** tab.
6. Unlock masintfc.tk.sasmode in the **Property Name** column.
7. Change the value. The choices are true and false. The default value is true.
8. Click OK.

Increase Module ExecutionThroughput of the REST Interface

The module execution throughput of the SAS Micro Analytic Service REST interface can be increased. However, those making connections to the REST server to execute micro analytics must always be authorized and authenticated by some other means, such as a private network. If this is the case, you can edit the JVM option that starts the REST server to include the argument

```
-Dsas.mas.access.mode=private
```

As a result, the authentication is not required to execute micro analytics. Authentication is still required for other operations.

As a result of specifying this option, the CPU cycles and sockets that are used for authentication are available for other uses, such as executing micro analytics.

The place to edit the JVM option is host specific:

- Linux - **SAS/config/*LevN*/Web/WebAppServer/SASServer13_X/bin/setenv.sh**
- Windows - **SAS\Config*LevN*\Web\WebAppServer\SASServer13_X\conf\wrapper.conf**

Prevent HTTP Error Messages

To prevent HTTP error messages, make sure that the web server is located on a separate host machine from the web application server. When the web server and web application server are located on the same machine, they both compete to use the ephemeral ports on the system. Separating them reduces the contention for this finite resource.

Appendix 1

SAS Micro Analytic Service Return Codes

SAS Micro Analytic Service core component, tkmas, supports the following return codes. Depending on logging settings, an associated message, listed below, might be logged. When a message is logged, any substitution parameters (indicated by %s for string and %d for number) are filled in. The other SAS Micro Analytic Service interface layers, such as the Java interface and the REST interface, might log additional messages that are not listed below.

Return Code	#define Symbol	Message or Description
-1958744063	MASBadArgs	Invalid arguments.
-1958744062	MASInternalError	Internal error.
-1958744061	MASFailure	SAS Micro Analytic Service encountered a failure.
-1958744060	MASFail	%s encountered a failure.
-1958744059	MASUnexFail	%s encountered an unexpected failure.
-1958744058	MASUnexInternal	%s encountered an unexpected internal failure.
-1958744057	MASUnexFailIn	%s encountered an unexpected failure in %s.
-1958744056	MASFailIn	%s encountered a failure in %s.
-1958744055	MASFailWithText	%s encountered a failure in %s: %s.
-1958744054	MASSFGCBLock	Failed to obtain the SFGCB lock.
-1958744053	MASExeLock	Failed to obtain the .exe lock.
-1958744052	MASLockCreate	Failed to create the %s lock.
-1958744051	MASEventCreate	Failed to create the %s event for thread %d.
-1958744050	MASThreadCreate	Failed to create SAS Micro Analytic Service worker thread %d of %d.
-1958744049	MASCPUCount	Failed to determine the number of CPUs. Setting the number of worker threads to %d.

Return Code	#define Symbol	Message or Description
-1958744048	MASThreadCount	The number of threads requested, %d, exceeds the limit. The maximum allowable threads = %d times the number of CPUs = %d.
-1958744047	MASThreadPoolSize	Worker thread pool size set to: %d.
-1958744046	MASInitAlready	SAS Micro Analytic Service was already initialized.
-1958744045	MASInitFailed	SAS Micro Analytic Service failed to initialize.
-1958744044	MASNotLicensed	SAS Micro Analytic Service is not licensed.
-1958744043	MASLicSvcInitFailed	License service failed to initialize.
-1958744042	MASNotInitialized	SAS Micro Analytic Service is not initialized.
-1958744041	MASTermFailed	SAS Micro Analytic Service failed to terminate successfully.
-1958744040	MASArgTrunc	The maximum size of parameter %d in the %s call is not large enough, and the value has been truncated at %d characters.
-1958744039	MASCompStatus	Compiler encountered status 0x%X.
-1958744038	MASUnsupportedType	Unsupported type.
-1958744037	MASUnknownType	Unknown type.
-1958744036	MASNoSuchPackage	Package not found.
-1958744035	MASNoSuchMethod	Method not found.
-1958744034	MASNoSuchRevision	Revision not found.
-1958744033	MASRevisionGet	Failed to get revision.
-1958744032	MASNoSuchModule	Module not found.
-1958744031	MASNoSuchUserContext	User context not found.
-1958744030	MASModuleCtxtCreate	Failed to create module context.
-1958744029	MASUserCtxtCreate	Failed to create user context.
-1958744028	MASArgTypeMismatch	Argument type mismatch.
-1958744027	MASArgCoutMismatch	Argument count mismatch.

Return Code	#define Symbol	Message or Description
-1958744026	MASClientCodegenError	Code generation error.
-1958744025	MASDS2CompileError	DS2 compilation error.
-1958744024	MASDS2RuntimeError	DS2 run-time error.
-1958744023	MASTKGNoEntryPoint	Code generation did not find an entry point.
-1958744022	MASTKGGenericError	Code generation generic error.
-1958744021	MASInvalidRequest	Invalid request.
-1958744020	MASMissingEntryPoints	Missing entry points.
-1958744019	MASUnassignedInput	Unassigned input.
-1958744018	MASInternalOnly	Internal only.
-1958744017	MASOnlyValidForDS2	Valid only for DS2 code.
-1958744016	MASOnlyValidForC	Valid only for C code.
-1958744015	MASExecutionException	Exception occurred during execution.
-1958744014	MASCompilationException	Exception occurred during compilation.
-1958744013	MASDS2ThreadUnsupported	DS2 thread unsupported.
-1958744012	MASTKEDSError	DS2 error.
-1958744011	MASUnrecognizedLanguage	Unrecognized language.
-1958744010	MASUnspecifiedDataType	Unspecified data type.
-1958744009	MASTKThreadingError	Threading error.
-1958744008	MASFatalProgRepoLost	Program repository lost.
-1958744007	MASSaveToRepo	Failed to save to repository.
-1958744006	MASLog4SASCfgFailed	Logging configuration failed.
-1958744005	MASDS2CompileStart	User context '%s' compiling module '%s' on thread %d.
-1958744004	MASDS2CompileFinish	User context '%s' module '%s' thread %d compilation succeeded.
-1958744003	MASDS2CompileFailed	User context '%s' module '%s' thread %d new revision failed, RC = %d.

Return Code	#define Symbol	Message or Description
-1958744002	MASStartup	*** SAS Micro Analytic Service Started ***
-1958744001	MASShutdown	*** Micro Analytic Service Shutting Down ***
-1958744000	MASAsyncException	SAS Micro Analytic Service received async exception code %d.
-1958743999	MASAsyncInitFailed	SAS Micro Analytic Service failed to install async exception handler.
-1958743998	MASShutdownJNI	SAS Micro Analytic Service calling JVM System.exit(0).
-1958743997	MASExecDeletePending	Attempt to execute method %s while deletion pending for module context %s revision %d.
-1958743996	MASMTXDeletePending	Attempt to add module context &s while deletion pending for user context %s.
-1958743995	MASRevDeletePending	Attempt to create revision while deletion pending for module context %s.
-1958743994	MASRevDelDeletePending	Attempt to delete revision while deletion pending for module context %s.
-1958743993	MASRevDelRefCount	Pending delete called for module context %s with ref count %d.
-1958743992	MASRevDelRefCountError	Delete called for module context %s with ref count %d.
-1958743991	MASMTXDelete	Garbage collection is deleting module context %s.
-1958743990	MASCTXDeletePending	Attempt to delete user context %s while being deleted by another thread.
-1958743989	MASCTXGetCDTDelPending	Attempt to retrieve creation time from user context %s while deletion pending.
-1958743988	MASCTXGetMDTDelPending	Attempt to retrieve modified time from user context %s while deletion pending.
-1958743987	MASMTXGetCDTDelPending	Attempt to retrieve creation time from module context %s while deletion pending.
-1958743986	MASMTXGetMDTDelPending	Attempt to retrieve modified time from module context %s while deletion pending.
-1958743985	MASMTXGetRevDelPending	Attempt to retrieve highest revision from module context %s while deletion pending.

Return Code	#define Symbol	Message or Description
-1958743984	MASMTXGetIUODelPending	Attempt to retrieve internal use flag from module context %s while deletion pending.
-1958743983	MASRevGetCDTDelPending	Attempt to retrieve revision %d creation time from module context %s while deletion pending.
-1958743982	MASMTXGetMsgDelPending	Attempt to retrieve compilation messages from module context %s while deletion pending.
-1958743981	MASMTXRegDeletePending	Attempt to register name while deletion pending for module context %s.
-1958743980	MASMTXLangDelPending	Attempt to retrieve language of module context %s while deletion pending.
-1958743979	MASMTXGetDispDelPending	Attempt to retrieve display name from module context %s while deletion pending.
-1958743978	MASMTXGetCSrcDelPending	Attempt to retrieve C source code from module context %s revision %d while deletion pending.
-1958743977	MASCTXGetPkgsDelPending	Attempt to retrieve packages from user context %s while deletion pending.
-1958743976	MASMTXGetMthsDelPending	Attempt to retrieve methods from module context %s while deletion pending.
-1958743975	MASNoSuchEntryPoint	Entry point not found.
-1958743974	MASMTXGetSigDelPending	Attempt to retrieve method %s signature from module context %s while deletion pending.
-1958743973	MASCTXLdOOTBDelPending	Private load out-of-the-box packages for user context %s while deletion pending.
-1958743972	MASCTXRegIntDelPending	Attempt to publish internal package %s to user context %s while deletion pending.
-1958743971	MASCTXRemIntDelPending	Attempt to remove internal package %s from user context %s while deletion pending.
-1958743970	MASCreateGCAFailed	Attempt to create garbage collection control structures failed.
-1958743969	MASGarbageCollection	Garbage collection interval.
-1958743968	MASGarbageCollectionDel	Garbage collection found assets ready to delete.

Return Code	#define Symbol	Message or Description
-1958743967	MASGCException	Exception occurred during garbage collection run.
-1958743966	MASProgRepoUpdateError	Error obtaining exclusive lock to update DS2 program repository.
-1958743965	MASCTXDelete	Garbage collection is deleting user context %s.
-1958743964	MASRevDelete	Garbage collection is deleting module context %s revision %d.
-1958743963	MASDS2Fatal	Module context %s revision %d generated fatal runtime exception. Deleting revision.
-1958743962	MASGarbageCollectionTerm	Garbage collection is freeing control assets during shutdown.
-1958743961	MASShutdownHang	Worker thread did not interrupt after %d seconds during shutdown.

Appendix 2

REST Server Error Messages and Resolutions

The following table contains SAS Micro Analytic Service REST server error messages, as well as possible causes and remedies.

Error Messages	Cause and Remedy
Another operation on this module is going on.	Wait a while, find out what has changed on the module, and then decide whether it is appropriate to retry your operation. If the problem persists even though you are sure there is not another simultaneous operation on the module, restart the server to refresh its state.
API version 2 is not supported.	The cause is that one or more fields in the module definition object are incorrect or repeated. Correct the errors identified.
Bad Request encountered. Check the format and syntax of the source.	Check the SAS Micro Analytic Service log file for additional details as there can be multiple causes for this error. If the cause is not that an incorrect source was used when updating a module, a restart of the server might be necessary to refresh its state. It might also be necessary to reduce the level of concurrent module update.
Code is missing or assigned the null value.	The cause is that one or more fields in the module definition object is incorrect or repeated. Correct the errors that are identified.
Data type does not match the signature.	Correct the input parameters according to the step's input signature.
Error creating object for HTTP response body.	If you submitted a POST, PUT, or DELETE operation to change the module collection, use the appropriate GET operation to check whether the operation produced the effect that is desired. If the desired effect is not produced, check the SAS Micro Analytic Service log for error messages. (Errors are logged as well as returned through the response body.) If you submitted a POST operation to validate the inputs of a step, execute a step or another GET operation. It is safe to repeat the operation.

Error Messages	Cause and Remedy
Information about the steps in this public module is not available.	The cause of this error is too many simultaneous module creations or updates. Reduce the amount of concurrency.
Information about the steps in this public module is not available because module was compiled successfully before but failed recompilation this time.	The likely cause is that a dependent module is no longer available to recompile a module after the server restarts. Create the dependent module again.
Invalid source code.	The cause is either one or more compilation errors.
Label cannot be used together with start and limit.	Use either the label parameter or start and limit parameters in the GET operation on the modules or steps collections.
Metadata update failure.	Restart the server to go through the metadata correction procedure. Follow this with a GET operation on the module affected to see whether the module was created, updated, or deleted properly.
Module compilation failed with errors.	The cause is one or more compilation errors during re-compilation of a previously compiled module. This might be due to too many simultaneous module creations or updates. Reduce the amount of concurrency. A restart of the server might be necessary to go through the metadata correction procedure.
Module context was not created.	There can be multiple causes. A restart of the server might be necessary to refresh its state.
Module name cannot be changed from a PUT operation.	Use the same module name as the previous revision.
Module name cannot be determined.	If the source is DS2 code, the package does not have a name. Add a name to the package.
Module name **XYZ** is already taken.	Delete the existing module using that name, or choose a different module name when creating a module. If the error persists, this might be a symptom of incorrect metadata. A restart of the server might be necessary to go through the metadata correction procedure.
Module named **XYZ** already exists.	Delete the existing module using that name, or choose a different module name when creating a module. If the problem persists, restart the server to clear its state.
Module type **XYZ** is not valid. Valid value is text/vnd.sas.source.ds2.	The cause is one or more fields in the module definition object are incorrect or repeated. Correct the errors identified.

Error Messages	Cause and Remedy
No module with the module ID **XYZ** exists.	Verify that the module ID is correct. If the module ID is correct, the module might have been deleted. In that case, create the module again and use the new ID that is assigned to it. In the case of a clustered deployment, the module was never replicated to all peers and the load balancer sends your request to one of those peer nodes. Check the SAS Micro Analytic Service log to confirm that. A restart is necessary to go through the metadata correction procedure.
Private module named **XYZ** was not removed successfully.	This error can be left uncorrected, if **XYZ** does not pose a problem in the other operations of the server. Otherwise, restart the server to clear its state.
Scope is missing or assigned the null value.	The cause is that one or more fields in the module definition object is incorrect or repeated. Correct the errors identified.
Scope **XYZ** is not valid. Valid scopes are public and private.	The cause is that one or more fields in the module definition object are incorrect or repeated. Correct the errors identified.
Server encountered an internal error.	There can be multiple causes. Check the SAS Micro Analytic Service log for error messages. If the cause is compilation related, and the errors are on a dependent module, make sure that the dependent module exists. It can also be caused by too many simultaneous module creations or updates. In that case, reduce the amount of simultaneous module creations or updates. For other causes, a restart of the server might be necessary to refresh its state.
Server is not initialized properly.	There can be multiple causes. Check the SAS Micro Analytic Service log for more information. Correct the component that prevents the service from initializing properly.
Step ID **XYZ** failed to execute.	See "SAS Micro Analytic Service Return Codes" on page 129 for the meaning of the result code. Also, verify that the module ID is correct and verify the existence of the module by doing a GET operation on the module.
Step ID **XYZ** is not visible.	The step is a member of a private module and its information is hidden from you. Furthermore, you cannot execute this step. If you need to see the signature of this step, you can get the source of the module as an alternative.
The **XYZ** member is repeated.	The cause is that one or more fields in the module definition object are incorrect or repeated. Correct the errors identified.

Error Messages	Cause and Remedy
The **XYZ** property expects a string value but TYPE value is provided.	The value of a property should be a string. Change the value to a string by quoting the value in double quotation marks.
The **XYZ** property is not supported.	The only property that is allowed in the API is connectionString. Remove the property definition from the array.
There is more than one DS2 package in the code.	Provide only one DS2 package in a module definition.
This node is out of sync with the rest of the cluster.	The likely cause is network delay in replicating data from one node to its cluster peers. Another operation on the module on the node that has the up-to-date metadata might cause a correction of the module on the peer nodes. If that does not work, restart the cluster node to go through the metadata correction procedure.
Total number of input parameters (**number**) does not match the number of parameters required by input signature (**number**).	Correct the input parameters according to the step's input signature.
Type is missing or assigned the null value.	The cause is that one or more fields in the module definition object is incorrect or repeated. Correct the errors identified.
User context was not created.	There can be multiple causes. A restart of the server might be necessary to refresh its state.
Version is missing or assigned the null value.	The cause is that one or more fields in the module definition object are incorrect or repeated. Correct the errors identified.

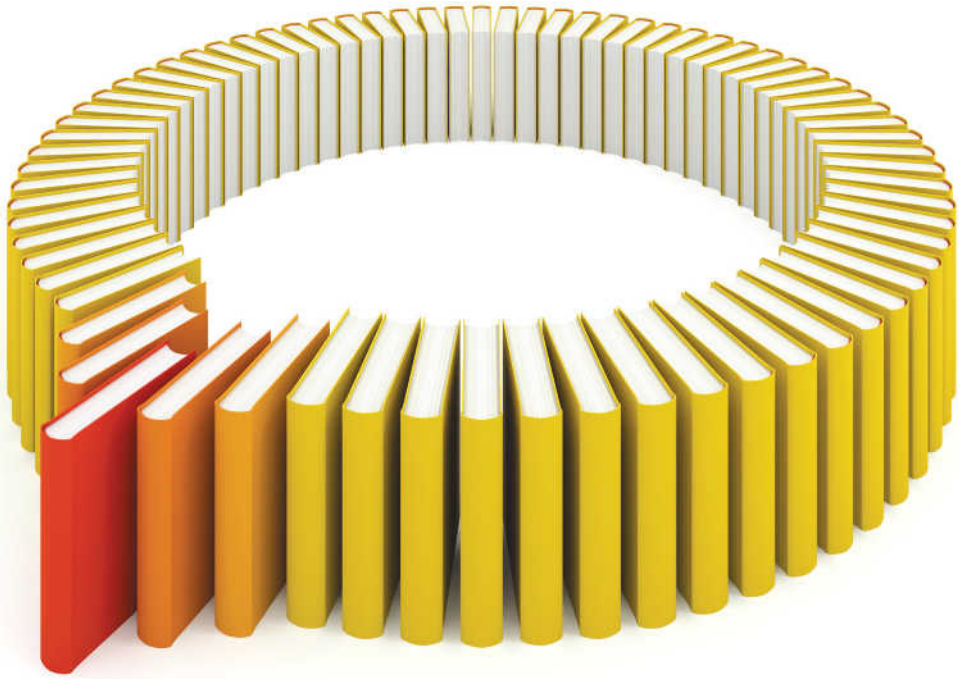
Recommended Reading

- *SAS 9.4 DS2 Language Reference*
- *SAS 9.4 Logging: Configuration and Programming Reference*
- *SAS 9.4 Web Applications Tuning for Performance and Scalability*
- *Encryption in SAS 9.4*
- *SAS 9.4 Intelligence Platform Middle-Tier Administration Guide*

For a complete list of SAS publications, go to sas.com/store/books. If you have questions about which titles you need, please contact a SAS Representative:

SAS Books
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-0025
Fax: 1-919-677-4444
Email: sasbook@sas.com
Web address: sas.com/store/books

Index



Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

support.sas.com/bookstore
for additional books and resources.