THE POWER TO KNOW.

SAS® Micro Analytic Service 2.2: Programming and Administration Guide

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2016. *SAS® Micro Analytic Service 2.2: Programming and Administration Guide*. Cary, NC: SAS Institute Inc.

SAS® Micro Analytic Service 2.2: Programming and Administration Guide

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

January 2017

2.2-P1:masag

Contents

About This Book . . . . . *v*
What's New in SAS Micro Analytic Service 2.2 . . . . . *vii*
Accessibility . . . . . *ix*

Chapter 1 • Introduction to SAS Micro Analytic Service **1**
What Is SAS Micro Analytic Service? 1

Chapter 2 • Concepts **3**
Overview 3
User or Business Context 4
Module Context 4
Revision 4
Architecture 5
Basic Steps for Using SAS Micro Analytic Service 5

Chapter 3 • DS2 Programming for SAS Micro Analytic Service **7**
Overview 7
DS2 Source Code Prerequisites 7
SAS Micro Analytic Service and SAS Foundation 8
I/O 9
Programming Blocks 10
Public and Private Methods and Packages 11
Argument Types Supported in Public Methods 13
DS2 Interface to Python 14

Chapter 4 • Best Practices for DS2 Programming **21**
Overview 21
Global Packages Versus Local Packages 21
Replacing SCAN (and TRANWRD) with DS2 Code 22
Hash Package 25
Character-to-Numeric Conversions 25
Passing Character Values to Methods 25
Performing the Computation Once 26
Moving Invariant Computations Out of Loops 26

Chapter 5 • Python Support in SAS Micro Analytic Service **27**
Introduction 27
Public and Private Methods 29
Example 30
Configuring Python 32
Configuring a SAS Application Server to Support the DS2 Pymas Package 34

Chapter 6 • Administration **39**
SAS Micro Analytic Service Logging 39
Secure DS2 HTTP Package Usage 40
Monitoring 40
Start-up Considerations for Clustered Deployments 43

Chapter 7 • Deployment and Tuning **45**
Pre-installation Steps 45
Deployment 46
Post-installation Steps 47
Cluster Deployment for SAS Micro Analytic Service 48
Tuning SAS Micro Analytic Service 50

Chapter 8 • Backup and Restore **53**
Overview 53
Backup Disk Stores 54
Restore Script 54
Additional Backup Considerations 55
Common Errors and Remediation 57

Chapter 9 • Upgrading, Migrating, and Promotion **59**
Upgrading and Migration 59
Promotion 59

Chapter 10 • SAS Micro Analytic Service REST API **61**
Overview 62
Terminology 63
Client Application Features 64
Security and Authentication 66
Life Cycle 66
Media Types 66
SAS Micro Analytic Service Media Types 68
Resources and Collections 83

Appendix 1 • SAS Micro Analytic Service Return Codes **127**

Appendix 2 • REST Server Error Messages and Resolutions **139**

Appendix 3 • Table Service Driver Reference **143**
DB2 Driver Reference 144
FedSQL Driver Reference 149
Greenplum Driver Reference 152
Netezza Driver Reference 156
ODBC Driver Reference 161
Oracle Reference 168
PostgreSQL Driver Reference 173
SAS Data Set Reference 178
Teradata Reference 182

Recommended Reading **187**
Index **189**

About This Book

Audience

This guide is intended for SAS Micro Analytic Service users who want to author SAS DS2 or Python code as analytical or rules-based decisioning logic in SAS Decision Manager. SAS Decision Manager includes SAS Micro Analytic Service as an execution engine. The guide also explains how SAS Micro Analytic Service executes decisions, and it offers tips, best practices, and restrictions on programming DS2 or Python to run in SAS Micro Analytic Service.

In addition, the guide shows how to configure SAS Micro Analytic Service, and how (as an option) to configure Anaconda Python and SAS Micro Analytic Service to run Python code that is called from DS2.

What's New in SAS Micro Analytic Service 2.2

Overview

SAS Micro Analytic Service 2.2 has the following enhancements:

- Ability to reference modules by name
- Ability to run Python modules in SAS Micro Analytic Service from PROC DS2
- Payload logging
- Simplified pymas DS2 package interface

Reference Modules by Name

Earlier releases of SAS Micro Analytic Services return a globally unique module identifier when a module is published. If the same module were published on two different SAS Micro Analytic Service systems, two different identifiers would be generated. Rather than returning a globally unique identifier, SAS Micro Analytic Service 2.2 returns the module name as the module identifier when a module is published. This allows a module to be referenced by the same name across all systems the module has been published to.

Run Python modules in SAS Micro Analytic Service from PROC DS2

DS2 code that is run using PROC DS2 can use the pymas DS2 package, as long as the SAS environment that is executing PROC DS2 has been configured to include Python and SAS Micro Analytic Service.

The pymas DS2 package enables DS2 methods to publish Python modules and to execute Python functions within those modules.

Payload Logging

Payload logging enables you to capture JSON payload, for both input and output, and log it to a file, so that it can be harvested and analyzed.

Simplified Pymas DS2 Package Interface

The pymas DS2 package enables DS2 methods to publish Python modules and to execute Python functions within those modules. Pymas is easier to use than in previous releases because it references Python modules by name rather than by a globally unique identifier and revision number.

When pymas is used in PROC DS2, the new method appendSrcLine can be used to add lines of Python source code, making it easier to publish Python modules. The new method getSource can be used to return the lines of Python source code as a single string.

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? 1

What Is SAS Micro Analytic Service?

SAS Micro Analytic Service 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 provides hosting for DS2 and Python 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. A web application layer with a REST interface provides easy integration with client applications, and adds persistence and clustering for scalability and high availability.

SAS 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 to author DS2 or Python 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 high-performance execution of transactions. Specific rules and restrictions are detailed in Chapter 3, "DS2 Programming for SAS Micro Analytic Service," on page 7.

Chapter 2

Concepts

Overview 3
User or Business Context 4
Module Context 4
Revision 4
Architecture 5
Basic Steps for Using SAS Micro Analytic Service 5

Overview

DS2 and Python programs that are published to SAS Micro Analytic Service, whether user-written or generated by SAS analytical solutions, are known as *modules*. This term reflects the language-neutral nature of SAS Micro Analytic Service interfaces.

A module is a collection of methods. For DS2, a module represents one DS2 package and its methods. For Python, a module is a collection of Python functions.

Module methods can be used for a wide variety of purposes, including computing scores, processing data, or making business decisions.

SAS Micro Analytic Service uses two internal component types to manage the modules that are published to it. These are the module context and the revision. A third component, the user context, provides isolated execution environments that contain sets of module contexts and revisions. In most cases, SAS Micro Analytic Service automatically manages user and module contexts for the user.

Note: If you plan to write DS2 modules to deploy to SAS Micro Analytic Service, follow the programming guidelines described in Chapter 3, "DS2 Programming for SAS Micro Analytic Service," on page 7. If you plan to write Python modules to deploy to SAS Micro Analytic Service, follow the programming guidelines described in Chapter 5, "Python Support in SAS Micro Analytic Service," on page 27.

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. 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.

In the case of Python, each module represents a collection of related Python functions, and each module method represents one of those functions.

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 or Python module, it creates a revision of that module. Therefore, a module context is a container of revisions. A module context also houses any compiler warning or error messages that were generated from the latest compilation or compilation attempt.

Note: The Micro Analytic Service REST interface supports running only the latest revision of a module.

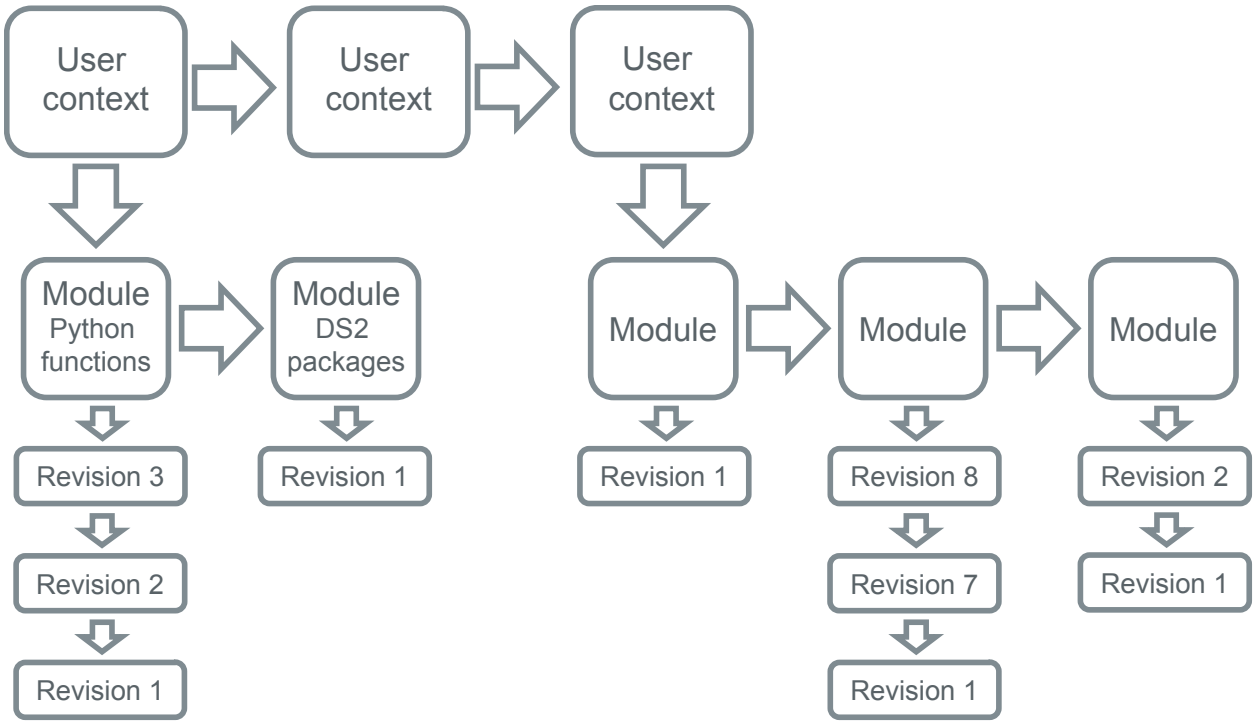
Revision

A revision is a version of a module. Each revision contains source code, an executable code stream (optimized binary executable), and metadata. The metadata describes the methods and method signatures of the module.

Revisions provide several advantages, including the ability to roll back to a previous version of a module.

SAS Micro Analytic Service assigns a revision number to each revision, which is a monotonically increasing integer beginning with 1. A revision is uniquely identified by module name and revision number. When you reference a revision, specifying revision number 0 selects the latest revision.

Figure 2.1 *Component Hierarchy*



Architecture

SAS Micro Analytic Service has a layered architecture:

Core Engine
: The SAS Micro Analytic Service core engine is written in C and is multi-threaded for high performance.

Java Layer
: a thin Java layer communicates with the core engine through the Java Native Interface (JNI). Commands from the REST/JSON interface are passed to the core engine through this Java layer.

REST/JSON
: adds functionality such as persistence and clustering support.

Basic Steps for Using SAS Micro Analytic Service

Using SAS Micro Analytic Service involves five steps. The REST interface automatically handles the first two.

1. Instantiate SAS Micro Analytic Service.
2. Get a user or business context. A user context is a module container, and provides an isolated execution environment.

3. Create one or more module contexts. A module context is a revision container, and represents a DS2 package. A revision has an executable code stream with an entry point for each DS2 package method, source code, and signature metadata.
4. For each module context, create one or more revisions.
5. Execute many times.

Chapter 3

DS2 Programming for SAS Micro Analytic Service

Overview 7
DS2 Source Code Prerequisites 7
SAS Micro Analytic Service and SAS Foundation 8
I/O 9
Programming Blocks 10
Public and Private Methods and Packages 11
Overview 11
Public Method Rules 11
Public Method Example 12
Private Method Example 13
Method Overloading 13
Argument Types Supported in Public Methods 13
Overview 13
Supported DS2 Data Types 14
Unsupported DS2 Data Types 14
DS2 Interface to Python 14

Overview

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

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

DS2 Source Code Prerequisites

The DS2 source code submitted to SAS Micro Analytic Service should begin with the following statement, just above the PACKAGE statement:

```
"ds2_options sas"
```

This statement instructs DS2 to use SAS missing value handling, which helps ensure that your DS2 program behaves the same as if it were run by SAS Foundation. DS2 source code should end with this statement:

```
"endpackage"
```

The code cannot contain DATA statements, PROC statements, or THREAD statements. 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. It is possible to place all of your source code on a single line. However, doing so makes it difficult to use compiler warning and error messages that include line numbers.

Note: DS2 supports only a specific style of comment. Comments start with the characters /*, and they end with the characters */. All characters between the starting and ending characters are part of the comment text. Comments can be nested. When there is ambiguity in determining a token, the compiler always chooses the longest possible sequence of characters that can make up a token.

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 that 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.

Here is an example PROC DS2 step that illustrates the above mechanism:

```
proc ds2;

ds2_options sas;
package myPackage/overwrite=yes;
method copyArray(char(12) in_array[4], in_out char(12) out_array[4]);
    out_array := in_array;
end;
endpackage;
run;

table _null_;
method init();
   dcl package myPackage p();
   dcl char(12) inarr[4];
   dcl char(12) outarr[4];
   inarr[1] = 'one';
   inarr[2] = 'two';
   inarr[3] = 'three';

   p.copyArray(inarr, outarr);
   put outarr[1]=;
   put outarr[2]=;
   put outarr[3]=;
end;
```

```
run;

quit;
```

I/O

SAS Micro Analytic Service supports I/O through the DS2 SQLStmt package. Supported databases include DB2, Greenplum, Netezza, Oracle, Postgres, SQL Server, and Teradata.

Connection strings are used to specify database connection information such as URL, credentials, and options. Only one connection string can be specified per user context. However, connection strings can be federated, allowing multiple databases to be used concurrently.

The SQLStmt package supports the FedSQL dialect. Therefore, the connection string should begin with **DRIVER=SQL;CONOPTS=(**, where **sql** specifies the FedSQL language driver as the managing driver, and one or more target driver connection strings are specified within the CONOPTS= option. The following example illustrates a federated connection string that includes Oracle and PostgreSQL data sources:

```
driver=sql;conopts=((driver=oracle;catalog=acat;uid=scott;
pwd=tiger;path=oraclev11.abc.123.com:1521/ORA11G);
(driver=postgres;catalog=bcat;uid=myid;pwd='mypass';
server=sv.abc.123.com;port=5432;DB=mydb;schema=public))
```

If you use the SAS Micro Analytic Service REST interface, you can enter your connection string in the Configuration Manager plug-in in SAS Management Console. Connection string forms vary from database to database. Most of the data source drivers require some client configuration, such as modifications to the environment variables that enable the driver software to be found and used correctly. You must ensure that the environment has been set up appropriately for the data source drivers that are being used. For more information, see Appendix 3, “Table Service Driver Reference,” on page 143.

Package SQLStmt enables you to specify a connection string in the DS2 code. However, this technique is not recommended. If the connection string is set through Configuration Manager, SAS Micro Analytic Service manages the database connection, detects whether the connection has been lost, and tries to reconnect periodically. If the connection string is set in the DS2 code, the connection is managed by the DS2 run time, which will not recover from lost connections. If connection strings are specified both in the DS2 code and through Configuration Manager or the Java API, SAS Micro Analytic Service overrides the connection string that was set in DS2.

When you are calling package SQLStmt to perform database I/O from a DS2 method, certain types of severe errors can cause DS2 to render the SQLStmt instance, and the DS2 package that called it, unusable. To maximize reliability, SAS Micro Analytic Service detects this condition and recompiles the offending package. This is useful if SQLStmt temporarily encounters fatal errors while performing database I/O. If a recompilation is successful, SAS Micro Analytic Service returns the error code MASDS2FatalRecompiled to indicate that the method failed but the package was successfully recompiled. If the recompilation fails, the error code MASDS2FatalRecompFailed is returned. If a given DS2 package must be recompiled more than 1000 times, SAS Micro Analytic Service removes the module from the system, and returns the error code MASDS2RevisionEjected.

Access to SAS data sets is supported. However, since they use file-level locking, they are not suitable for writing from multiple threads. Set appropriate connection options

carefully before reading SAS data sets from multiple threads. Otherwise, a deadlock occurs. For these reasons, the use of a third-party database management system is highly recommended.

Note: If SAS Micro Analytic Service is installed with SAS Decision Manager, SAS Micro Analytic Service must be installed on servers that have the same operating system family as the SAS Decision Manager server tier. For more information, see *SAS Decision Manager Administrator's Guide*. This requirement ensures that appropriate data access components are licensed for use by both SAS Micro Analytic Service and SAS Decision Manager.

For detailed driver reference information, see Appendix 3, "Table Service Driver Reference," on page 143.

SAS Micro Analytic Service enables access to HTTP and HTTPS web services through the DS2 HTTP package, which can execute HTTP requests to, and receive responses from, HTTP and HTTPS web services. Direct file I/O is not supported. As a result, DS2 hash packages cannot be populated from the contents of a file.

For more information about DS2 and FedSQL, see *SAS 9.4 DS2 Language Reference.* and *SAS 9.4 FedSQL Language Reference*.

Programming Blocks

Each DS2 module represents exactly one package, and therefore 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.

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.
- DS2 packages must not be passed as arguments in public methods. The presence of a DS2 package argument results in the method becoming private.
- The VARARRAY statement must not be present in the argument list of a public method. VARARRAY is a DS2 statement, not a data type. The presence of VARARRAY in a methods argument list causes the method to become private.
- For a full list of data types that can be used as public method arguments, see "Supported DS2 Data Types" on page 14.

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 does not support method overloading.

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

- BIGINT
- CHAR(n)
- DOUBLE
- INTEGER
- NCHAR(n)
- NVARCHAR(n)
- VARCHAR(n)

Unsupported DS2 Data Types

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

DS2 Interface to Python

DS2 modules, running in SAS Micro Analytic Service, can publish and execute Python modules.

Note that Python 2.7 or Python 3.4 must be available for SAS Micro Analytic Service to load. If both are available, SAS Micro Analytic Service loads Python 3.4. See Chapter 5, "Python Support in SAS Micro Analytic Service," on page 27, for information about installing Python and configuring the environment variables necessary to allow Python to run embedded in SAS Micro Analytic Service. As is the case when calling any package from DS2, it is recommended that you always check return codes where available, and return any error codes using an output argument from your DS2 method.

To call Python from DS2, use the DS2 package called pymas. Each pymas package instance represents exactly one Python module revision. You can create as many instances as you want, allowing multiple modules to be used.

Here are some operations that a DS2 module would typically perform.

Instantiate the following DS2 package:

```
py = _new_ pymas();
```

Calling publish() compiles your Python module and sets it as the module that is represented by this pymas instance. Subsequent pymas function calls, such as setting values and executing methods, operate on this module. The Python code is passed as a string in the first argument. Pass the name that you want to give to your new Python module in the second argument. publish() returns the revision number that SAS Micro Analytic Service assigned to your new module. You could use this revision number later to execute or delete a specific revision of your module. If you do not specify a revision number, the latest revision is assumed. If your Python code fails to publish (because of syntax errors, for example), then -1 is returned for the revision number.

```
revision = py.publish( pgm, moduleName );
```

In very rare cases, you might need to use a prior revision of a module rather than the latest revision that would be selected by default. Or, rather than publishing a Python module from DS2, you might need to specify a module that was previously published to SAS Micro Analytic Service by an external client. In these rare cases, you can call useModule() instead of publish(). If a module was already associated with your pymas instance before calling useModule(), then useModule() disassociates the current module from the instance before making the specified module current.

```
rc = py.useModule( moduleName, revision);
```

Before calling Python, you must tell the pymas instance which method to execute. This is accomplished by calling useMethod(). In addition to specifying the method (Python function) to call, useMethod() also validates that the method exists within the current module, prepares the pymas instance to receive the input values for the specific method arguments, and prepares to return any output values from the method execution.

```
rc = py.useMethod( methodName );
```

Call the type-specific setter methods to set input values before executing the method. Because these setters store arguments by name, they can be called in any order, and they insert the values in the correct positions:

```
py.setDouble("airflow", sensor_maf);
```

Since the DS2 package instance represents a single revision, the execute() method needs no arguments.

```
rc = py.execute();
```

After execution, call getters to retrieve the results.

```
score = py.getDouble("credit_score");
```

Scalar argument setters are of the form:

```
return_code = set<type>(name, value)
```

Scalar argument getters are of the form:

```
value = get<type>(name)
```

Array argument setters are of the form:

```
rc = set<type>Array(name, array-value)
```

Array argument getters are of the following form.

Note: DS2 passes arrays and output values by reference.

```
get<type>Array(name, array-value, rc)
```

The example below assumes that you have declared your package as py:

```
dcl package pymas py;
dcl int rc;
```

```
dcl bigint result;

rc = py.publish(python_source_code, my_module_name);
py.setString("inString", "A string");

py.execute()

result = py.getLong("outLong");
```

The complete set of DS2 package methods follows, where **rc** is the integer return code, and **py** is the package instance.

Methods for Python module management and execution:

```
rc = py.publish(python_source_code, "module_name");
rc = py.remove();
rc = py.isLoaded(); // returns true is Python is available and false otherwise
revision = py.getRevisionNumber();
rc = py.setTimeZone(time_zone_identifier);
rc = py.execute();
```

Scalar argument setters:

```
 rc = py.setString(argument_name, value);
rc = py.setBool(argument_name, value);
rc = py.setLong(argument_name, value);
rc = py.setInt(argument_name, value);
rc = py.setDouble(argument_name, value);
rc = py.setDateTime(argument_name, value);
rc = py.setDate(argument_name, value);
rc = py.setTime(argument_name, value);
```

Scalar argument getters:

```
string_value   = py.getString(argument_name);
int_value              = py.getBool(argument_name);
long_value             = py.getLong(argument_name);
int_value              = py.getInt(argument_name);
double_value           = py.getDouble(argument_name);
date_time_value        = py.getDateTime(argument_name);
date_value             = py.getDate(argument_name);
time_value             = py.getTime(argument_name);
```

Array argument setters:

```
rc = py.setStringArray(argument_name, string_array);
rc = py.setBoolArray(argument_name, integer_array);
rc = py.setLongArray(argument_name, bigint_array);
rc = py.setIntArray(argument_name, integer_array);
rc = py.setDoubleArray(argument_name, double_array);
rc = py.setDateTimeArray(argument_name, date_time_array);
rc = py.setDateArray(argument_name, date_array);
rc = py.setTimeArray(argument_name, time_array);
```

Array argument getters:

```
py.getStringArray(argument_name, string_array, rc);
py.getBoolArray(argument_name, integer_array, rc);
py.getLongArray(argument_name, bigint_array, rc);
py.getIntArray(argument_name, integer_array, rc);
py.getDoubleArray(argument_name, double_array, rc);
```

```
py.getDateTimeArray(argument_name, date_time_array, rc);
py.getDateArray(argument_name, date_array, rc);
py.getTimeArray(argument_name, time_array, rc);
```

Python 2.*x* uses ASCII as the default encoding. Therefore, you must specify another encoding at the top of the file to use non-ASCII Unicode characters in literals. As a best practice, when using Python 2.*x*, always use the following as the first line of your Python script:

```
# -*- coding: utf-8 -*-
```

Also, in Python 2.*x*, the Unicode literal must be preceded by the letter u. Therefore, literal strings should be written using the following form:

```
u"xxxxx"
```

Note: Python 3.*x* uses UTF-8 as the default encoding, so these issues affect Python 2.*x* only. When using Python 3.*x*, the default encoding can be used, and literals can simply be enclosed in quotation marks.

If you prefer not to insert the linefeed characters yourself, you can add the Python source code line-by-line using the appendSrcLine() method. When the entire Python program has been added, you then call the getSource() method. The getSource() method returns the Python program as one string, inserting linefeed characters between Python source code lines. You can then pass that string to the publish method to publish the Python program in SAS Micro Analytic Service. Here is an example.

```
data tstinput; a = 8;  b = 4; output; a = 10; b = 2; output;
run;

proc ds2;
    ds2_options sas;
    package testpkg /overwrite=yes;
        dcl package pymas py();
        dcl package logger logr('App.TableServices.DS2.Runtime.Log');
        dcl varchar(67108864) character set utf8 pycode;
        dcl int rc revision;

        method testpkg( varchar(2048) modulename, varchar(2048)pyfuncname );
            rc = py.appendSrcLine('# Here is the first Python function:');
            rc = py.appendSrcLine('def domath1(a, b):');
            rc = py.appendSrcLine('  "Output: c, d"');
            rc = py.appendSrcLine('  print("Will compute {0} times {1}".format(a, b))');
            rc = py.appendSrcLine('  c = a * b');
            rc = py.appendSrcLine('  print("domath1 c is {0}".format(c))');
            rc = py.appendSrcLine('  print("domath1 also do {0} div {1}".format(a, b))');
            rc = py.appendSrcLine('  d = a / b');
            rc = py.appendSrcLine('  print("domath1 d is {0}".format(d))');
            rc = py.appendSrcLine('  return c, d');
            rc = py.appendSrcLine('');
            rc = py.appendSrcLine('# Here is the second function:');
            rc = py.appendSrcLine('def domath2(a, b):');
            rc = py.appendSrcLine('  "Output: c, d"');
            rc = py.appendSrcLine('  c,d = domath1( a, b )');
            rc = py.appendSrcLine('  print("domath2: c is {0} and d is {1}".format(c,d))');
            rc = py.appendSrcLine('  return c, d' );
            pycode = py.getSource();
            logr.log( 'I', 'pycode=$s', pycode );
            revision = py.publish( pycode, modulename );
```

```
            if revision lt 1 then
                logr.log( 'E', 'pymas.publish() failed.');
            rc = py.useMethod( pyfuncname );
            if rc then
                logr.log( 'E', 'pymas.useMethod() failed.');
        end;

        method exec( double a, double b, in_out int rc,
                     in_out double c, in_out double d );
            rc = py.setDouble( 'a', a );   if rc then return;
            rc = py.setDouble( 'b', b );   if rc then return;
            rc = py.execute();             if rc then return;
            c = py.getDouble( 'c' );
            d = py.getDouble( 'd' );
        end;
    endpackage;

    data _null_;
        dcl package logger logr( 'App.TableServices.DS2.Runtime.Log' );
        dcl package testpkg t( 'my Python Module Context name', 'domath2' );
        dcl int rc;
        dcl double a b c d;

        method run();
            a = b = c = d = 0.0;
            set tstinput;
            t.exec( a, b, rc, c, d );
            logr.log( 'I', '##### Results: a=$s   b=$s   c=$s   d=$s',
                      a, b, c, d );
            put a= b= c= d=;
        end;
    enddata;
    run;
quit;
```

When using PROC DS2 in a SAS session to create a pymas package instance, you cannot provide the Python program as one big quoted literal string. The reason is that the SAS tokenizer strips out the embedded line-ending characters, causing indentation problems in the Python code. In this situation, the pymas package's appendSrcLine() and getSource() methods can be used to produce a DS2 character variable containing the lines of code concatenated together with embedded linefeed characters separating the lines of Python code. Once you have added each line of your Python code to the pymas package instance using the appendSrcLine() method, you can use the "getSource() method to retrieve the complete program into a DS2 character variable, which can then be provided as the first input argument to the pymas publish() method. Here is an example.

```
ds2_options sas;
    package testpkg /overwrite=yes;
        dcl package pymas py();
        dcl package logger logr('App.tk.MAS');
        dcl varchar(67108864) character set utf8 pycode;
        dcl int rc revision;

        method testpkg( varchar(256) modulename,
                        varchar(256) pyfuncname );
            rc = py.appendSrcLine('# The first Python function:');
```

```
        rc = py.appendSrcLine('def domath1(a, b):');
        rc = py.appendSrcLine('  "Output: c, d"');
        rc = py.appendSrcLine('  c = a * b');
        rc = py.appendSrcLine('  d = a / b');
        rc = py.appendSrcLine('  return c, d');
        rc = py.appendSrcLine('');
        rc = py.appendSrcLine('# Here is the second function:');
        rc = py.appendSrcLine('def domath2(a, b):');
        rc = py.appendSrcLine('  "Output: c, d"');
        rc = py.appendSrcLine('  c,d = domath1( a, b )');
        if rc then logr.log( 'E', 'py.appendSrcLine() failed.');
        rc = py.appendSrcLine('  return c, d' );
        pycode = py.getSource();
        revision = py.publish( pycode, modulename );
        if revision lt 1 then
            logr.log( 'E', 'py.publish() failed.');
        rc = py.useMethod( pyfuncname );
        if rc then logr.log( 'E', 'py.useMethod() failed.');
    end;

    method usefunc( varchar(256) pyfuncname );
        rc = py.useMethod( pyfuncname );
        if rc then logr.log( 'E', 'py.useMethod() failed.');
    end;

    method exec( double a, double b, in_out int rc,
                 in_out double c, in_out double d );
        rc = py.setDouble( 'a', a );   if rc then return;
        rc = py.setDouble( 'b', b );   if rc then return;
        rc = py.execute();             if rc then return;
        c = py.getDouble( 'c' );
        d = py.getDouble( 'd' );
    end;
endpackage;
```

Chapter 4

Best Practices for DS2 Programming

Overview **21**

Global Packages Versus Local Packages **21**
Overview 21
Example of Optimized Code 22
Example of Poorly Optimized Code 22

Replacing SCAN (and TRANWRD) with DS2 Code **22**

Hash Package **25**

Character-to-Numeric Conversions **25**

Passing Character Values to Methods **25**

Performing the Computation Once **26**

Moving Invariant Computations Out of Loops **26**

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.

Example of Optimized Code

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;
```

Example of Poorly Optimized Code

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

Python Support in SAS Micro Analytic Service

Introduction **27**

Public and Private Methods **29**

Example **30**

Configuring Python **32**
Python 2.7 and 3.4 on 64-Bit Windows 32
Python 2.7 and 3.4 on 64-Bit Linux 32
Further Considerations for Configuring Python 33

Configuring a SAS Application Server to Support the DS2 Pymas Package **34**

Introduction

SAS Micro Analytic Service 2.2 supports modules that are written in the Python programming language. A Python module represents a collection of Python functions, and each of the module's methods represents one Python function. Python modules can be published to SAS Micro Analytic Service and called from DS2. (See "DS2 Interface to Python" on page 14.) If your SAS solution supports it, Python modules can be published and called directly.

Here is an example of Python code that can be used by SAS Micro Analytic Service:

```
def func0():
    print('func1')

def func1(arg1, arg2):
    "Output: arg3, arg4"
    func0()
    arg3=arg1 + arg2
    arg4 = arg3 + 1
    return arg3, arg4

def func2(arg1, arg2):
    "Output: arg3"
    func0()
    arg3=arg1 + arg2
    return arg3
```

The parameters listed with the function definition are considered the input arguments for the function. The first line after the function declaration must be a quoted string containing the word "Output:" followed by all of the outputs for the function.

This example has no output. Input arguments are given in the function's argument list. This example has input variables a and b. Outputs of the function must be listed after "Output:" in the quoted string that follows the function definition. The output variables should match the variables listed in the return statement.

```
def calcATimesB(a, b):
  "Output: "
  print ("Function with no output variables.")
  c = a * b
  print ("Result is: ", c, ", but is not returned")
  return None
```

Note: Input and output argument names live in a single namespace. Therefore, they cannot be the same. This means that in_out arguments are not supported. This is true for all module types in SAS Micro Analytic Service. This is not an issue in Python, as a new variable can be assigned the value of an input argument and then safely added to the output list. If the "Output:" line is missing, the function is not exposed as a callable function through SAS Micro Analytic Service. However, that function can be called internally. As a result, any function without the "Output:" line is a private function. The func0() function is an example of such a private function. SAS Micro Analytic Service parses the code to create a dictionary of the methods and their signatures.

The following data types are supported between SAS Micro Analytic Service and Python.

SAS Micro Analytic Service Type	SAS Micro Analytic Service Functions	Python Type	Python Example
String	• sfSymGetString() • sfSymGetStringArray() • sfSymSetString() • sfSymSetStringArray()	Unicode string	outStr = unicode('abcdef')
BigInt	• sfSymGetBigInt() • sfSymGetBigIntArray() • sfSymSetBigInt() • sfSymSetBigIntArray()	Long	outLong = 10
Double	• sfSymGetDouble() • sfSymGetDoubleArray() • sfSymSetDouble() • sfSymSetDoubleArray()	Float	outFloat = 10.10

SAS Micro Analytic Service Type	SAS Micro Analytic Service Functions	Python Type	Python Example
Boolean	• sfSymGetBool() • sfSymGetBoolArray() • sfSymSetBool() • sfSymSetBoolArray()	Boolean	outBool = True
Datetime	• sfSymGetDatetime() • sfSymGetDatetimeArray() • sfSymSetDatetime() • sfSymSetDatetimeArray()	Datetime	outDatetime = datetime.datetime(2013, 12, 22, 11, 30, 59)
Date	• sfSymGetDate() • sfSymGetDateArray() • sfSymSetDate() • sfSymSetDateArray()	Date	outDate = datetime.date(2013, 12, 22)
Time	• sfSymGetTime() • sfSymGetTimeArray() • sfSymSetTime() • sfSymSetTimeArray()	Time	outTime = datetime.time(11, 30, 59)

Public and Private Methods

Private and public methods are SAS Micro Analytic Service concepts, rather than Python features. Any method having the "Output:" doc string is considered a public method. If a method does not have the "Output:" doc string, then it is considered a private method. SAS Micro Analytic Service can host public and private Python methods, where a method is a Python function. Although a private method cannot be called directly, it can be called by another method (public or private). Private methods are useful as utility functions. 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 methods.

Python modules can be published containing all public methods, or a mixture of public and private methods. Both public and private methods can call other functions that either

exist within the module internally or in external Python packages, including third-party libraries.

All public functions returning more than one output argument must return a tuple containing all of the output arguments. This can be done by returning all of the arguments separated by commas. When returning zero arguments from a public function you are still required to include the "Output:" doc string to indicate a public function. It should simply be "Output:", with no output arguments listed. You can omit the return statement, return "None", or return an empty tuple.

An example of returning an empty tuple is **`return ()`**. An example of returning "None" is **`return None`**. One output argument can be returned as is. It is not required to be returned within a tuple. Here is an example: **`return a`**.

Therefore, it could be **`return a,b,c or return (a,b,c)`**.

Note: Order does matter. Therefore, the order in the return statement must match the order in the "Output:" line. A best practice is to cut and paste from one to the other.

Example

The following simple example illustrates how to pass parameter data as input to and as output from a public Python function.

```
#
#    Name: scalarsTest.py
# Purpose: Test Python program for scalar types
#
# Inputs (name)         (type)
#       inString        String
#       inBool          Boolean
#       inLong          Long
#       inDouble        Double
#       inDateTime      DateTime
#       inDate          Date
#       inTime          Time
#
# Outputs (name)        (type)
#       outString       String
#       outBool         Boolean
#       outLong         Long
#       outDouble       Double
#       outDateTime     DateTime
#       outDate         Date
#       outTime         Time

# import the datetime module to perform datetime operations
import datetime

def scalarsTest(inString, inBool, inLong, inDouble, inDateTime, inDate, inTime):
    "Output: outString, outBool, outLong, outDouble, outDateTime, outDate, outTime"

    if inString == None:
        outString = None
    else:
```

```
    # convert the casing of the string input
    outString = inString.swapcase()
print ("\n inString=", inString, " outString=", outString)

if inBool == None:
    outBool = None
else:
    # reverse value of boolean
    outBool = not inBool
print ("\n inBool=", inBool, " outBool=", outBool)

if inLong == None:
    outLong = None
else:
    # add 10 to long
    outLong = inLong + 10
print ("\n inLong=", inLong, " outLong=", outLong)

if inDouble == None:
    outDouble = None
else:
    # add 10.1 to the double
    outDouble = inDouble + 10.1
print ("\n inDouble=", inDouble, " outDouble=", outDouble)

if inDateTime == None:
    outDateTime = None
else:
    # add a day to the datetime object
    outDateTime = inDateTime + datetime.timedelta(days=1)
print ("\n inDateTime=", inDateTime, " outDateTime=", outDateTime)

if inDate == None:
    outDate = None
else:
    # add a week to the date object
    outDate = inDate + datetime.timedelta(weeks=1)
print ("\n inDate=", inDate, " outDate=", outDate)

if inTime == None:
    outTime = None
else:
    # add 30 minutes to the time object
    dt = datetime.datetime.combine(datetime.date.today(), inTime)
    dt = dt + datetime.timedelta(minutes=30)
    outTime = dt.time()
print ("\n inTime=", inTime, " outTime=", outTime)

# return all of our outputs
return outString, outBool, outLong, outDouble, outDateTime, outDate, outTime
```

Configuring Python

Python 2.7 and 3.4 on 64-Bit Windows

1. Run Anaconda for Windows, Python 3.5, Windows 64-bit Graphical Installer from https://www.continuum.io/downloads.

   Note: During the installation process, you are prompted for the destination folder. These instructions assume that your Anaconda Python installation folder is C:\Anaconda3.

2. Create a Python environment by entering the following at a Windows command prompt (note that there are two hyphens before **name**). Provide the Python version that you installed. In the following example, Python 3.4 is used.

   ```
   conda create --name python34 python=3.4
   ```

3. Activate the environment (providing the appropriate Python version).

   ```
   activate python34
   ```

4. Before adding any other Python packages to your new, activated environment, first install the nomkl package.

   ```
   conda install nomkl
   ```

   Note: Intel Math Kernel Library (MKL) is incompatible with the SAS kernel. Installing nomkl instructs Anaconda to select a non-MKL dependent package version whenever Python packages are added to your environment. If you have a pre-existing environment that has MKL-dependent packages, follow the instructions at www.continuum.io for removing MKL and replacing the packages that have MKL dependencies before using SAS Micro Analytic Service.

5. This step is for Python 3.4 only.

   Set environment variable PYTHONHOME, according to the location where Python is installed. Here is an example:

   ```
   set PYTHONHOME=C:\anaconda3
   ```

   Note: When enabling the SAS Web Application Server to use Python, disable the service and start the SAS Web Application Server in the foreground from a DOS command shell in which you have activated your Python environment. For example, after you have activated your Python environment in a DOS command shell as described in above, change your shell's present working directory to the server's bin directory and enter tcruntime-ctl run. You can stop the server using Control-C. The tcruntime-ctl.bat script is located in the ***SAS-configuration-directory*/Lev*N*/Web/WebAppServer/*SASServer13_N*/bin** directory.

Python 2.7 and 3.4 on 64-Bit Linux

1. Download Anaconda for Linux, Python 3.5, and Linux 64-bit installer from https://www.continuum.io/downloads.

2. After downloading the installer, enter the following code in a terminal window. Provide the Python version that you installed. In the following example, Python 3.5 is used.

```
bash Anaconda3-2.5.0-Linux-x86_64.sh
```

Answer yes to the question, **Do you wish the installer to prepend the Anaconda3 install location to PATH in your .bashrc?** These instructions assume that you used the location **`/users/myuserid/anaconda3`**.

3. Create a Python 3.4 environment by entering the following code (note that there are two hyphens before **`name`**). Provide the appropriate Python version.

```
bash
conda create --name python34 python=3.4
```

4. Before adding any other Python packages to your new, activated environment, first install the nomkl package.

```
conda install nomkl
```

Note: Intel Math Kernel Library (MKL) is incompatible with the SAS kernel. Installing nomkl instructs Anaconda to select a non-MKL dependent package version whenever Python packages are added to your environment. If you have a pre-existing environment that has MKL-dependent packages, follow the instructions at www.continuum.io for removing MKL and replacing the packages that have MKL dependencies before using SAS Micro Analytic Service.

5. Activate the environment (provide the appropriate Python version).

```
source activate python34
```

6. Prepend the Python environment's **`lib`** directory to the LD_LIBRARY_PATH environment variable. Provide the Python version that you installed. In the following example, Python 3.4 is used.

```
LD_LIBRARY_PATH=/users/myuserid/anaconda3/envs/python34/lib:${LD_LIBRARY_PATH}
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH%:}
```

Note: Regarding 64-bit Linux, the **`conda create`** and **`source activate`** commands must be run from a bash or zsh shell.

If you encounter a message at run-time, such as "Could not find platform independent libraries" and "Consider setting $PYTHONHOME to <**`prefix`**>[:<**`exec_prefix`**>]", then set the PYTHONHOME environment variable. Here is an example:

```
export PYTHONHOME=$CONDA_PREFIX
```

Further Considerations for Configuring Python

The Anaconda documentation states that Python 3.4 can be run from an Anaconda 2.7 installation by creating and activating a Python 3.4 environment. However, you cannot do this with embedded Python. Therefore, it is recommended that you use the Python 3.5 installer for both Python 2.7 and 3.4.

When starting SAS Web Application Server, do so from a shell in which you have activated Python, thus enabling the process to use Python. For example, when starting the server using a script such as tcruntime-ctl.sh, do so from the shell in which you activated Python, as described above.

A rich set of Python packages is available, covering a wide variety of computing needs. You might want to add some of these packages to your Python environment.

Note: Make sure nomkl has been installed as the first package, as instructed in the previous sections.

When you add packages to an Anaconda environment, the packages are placed in **`<your-environment-path>/lib/python3.4/site-packages`**. In order to use

the Python scripts that these packages require, add their locations to the PYTHONPATH environment variable.

If your Python script imports your own .py files, you also must add their location to PYTHONPATH. An example location might be .(dot).

Some packages include a lib directory, which also needs to be added to PYTHONPATH.

Finally, you must add **<*your-environment-path*>/lib/python3.4** to PYTHONPATH.

Anaconda sets the environment variable CONDA_PREFIX when you activate an environment and sets it to the location where Anaconda stores any new Python packages that you install (for example, the site-packages folder).

Here is an example of the locations that you might set for PYTHONPATH, after adding packages to your Python 3.4 environment for 64-bit Linux:

```
export SITE_PACKAGES=$CONDA_PREFIX/lib/python3.4/site-packages
export PYTHONPATH=.:$CONDA_PREFIX/lib/python3.4
export PYTHONPATH=$PYTHONPATH:$SITE_PACKAGES
export PYTHONPATH=$PYTHONPATH:$SITE_PACKAGES/numpy
export PYTHONPATH=$PYTHONPATH:$SITE_PACKAGES/numpy/lib
```

Configuring a SAS Application Server to Support the DS2 Pymas Package

The SAS DS2 pymas package provides interfaces that enable users to publish and execute Python code using the SAS Micro Analytic Service. This section describes how SAS Micro Analytic Service can be configured in a SAS Application Server, enabling you to test DS2 pymas package usage using PROC DS2 running in a SAS session. Examples of a SAS session are a workspace server that has been launched by SAS Studio or SAS Decision Manager.

Two user modifications are needed when configuring a SAS Application Server to support the use of the SAS DS2 pymas package in a SAS server, such as the workspace server. Here is a brief example:

1. Copy the SAS Micro Analytic Service shared libraries from ***SASHome*/SASFoundation** on the middle tier to the appropriate directory under SAS Foundation, on the server tier.
2. Add environment settings to the server's corresponding _usermods.sh(.bat) file provided in the server's configuration directory.

Because the _usermods files are sourced within each of the server wrapper scripts, the server inherits any logic or environment. SAS preserves _usermods files during software updates, unlike the server wrapper scripts, which SAS overwrites. For this reason, editing the wrapper scripts is discouraged.

One place that SAS Micro Analytic Service is installed in the middle tier is a platform-specific location within ***SASHome*/SASFoundation**. Here is an example on UNIX platforms:

- ***SASHome*/SASFoundation/<*release*>/sasexe**

A specific instance on a UNIX platform can look like this:

- **/install/SASServer/SASHome/SASFoundation/9.4/sasexe**

Here is a list of the shared libraries that must be copied from that directory on a UNIX platform:

- libtksf.so
- pymas.so
- tkmaspy2.so
- tkmaspy3.so
- t1j8en.so

The library name t1j8en.so is the English translation of the SAS Micro Analytic Service message file. Copy any other SAS Micro Analytic Service language files matching t1j8??.so as well.

On Windows, the list is almost the same as UNIX except that the file extensions are .dll instead of .so, and libtksf.so is tksf.dll:

- tksf.dll
- pymas.dll
- tkmaspy2.dll
- tkmaspy3.dll
- t1j8en.dll

On Windows, those libraries are located at **core\sasext**, instead of sasexe. Here is an example:

- **C:\Program Files\SASHome\SASFoundation\9.4\core\sasext**

With respect to environment setting changes needed in the _usermodes.sh(.bat) file, the environment must be updated to enable the server to find Python and any Python modules needed by your Python code. When configuring the workspace server on UNIX, you are updating the following environment **sas-configuration-directory/<*LevN*>/SASApp/WorkspaceServer/WorkspaceServer_usermods.sh**:

First make a backup copy. Here is an example:

```
cp --preserve=timestamps WorkspaceServer_usermods.sh \
          WorkspaceServer_usermods.orig.sh
```

Then, make the following changes to WorkspaceServer_usermods.sh. This example assumes that Anaconda Python is installed and configured on the server machine, and that at least one Python environment has been created.

1. Update the name of the shell in the first line from sh to bash. Here is an example:

   ```
   #!/bin/bash -p
   ```

   Note: The remainder of the updates are the same commands that you use to activate and configure your Anaconda Python environment normally.

2. Source the Anaconda activate script to activate the desired environment. Here is an example:

   ```
   source /anaconda3/bin/activate python27
   ```

   Note: A dot (.) is an alias for the source command.

3. Add ${CONDA_PREFIX}/lib to the LD_LIBRARY_PATH environment variable. Here is an example:

   ```
   LD_LIBRARY_PATH=${CONDA_PREFIX}/lib:${LD_LIBRARY_PATH}
   ```

```
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH%:}
```

4. If the server has multiple Python installations that are conflicting, you could possibly encounter an error, such as `Python DateTime API failed to load`. If so, set the PYTHONHOME to your Python home directory. Here is an example:

```
PYTHONHOME=/anaconda3
export PYTHONHOME
```

For more information see, "Further Considerations for Configuring Python" on page 33.

On Windows, when configuring the workspace server, you are updating WorkspaceServer_usermods.bat. Add the command that you used to activate the Anaconda Python environment that you created previously. Here is an example:

```
c:\Anaconda3\Scripts\activate python27
```

If you have other environment settings needed for your Anaconda Python environment, add those commands as well.

You can test your configuration by submitting a PROC DS2 program to make sure it can successfully use the DS2 pymas package. Here is an example:

```
%macro chkrc; if rc then put rc=; %mend;
%macro addln( line ); rc = py.appendSrcLine( &line ); %chkrc; %mend;

/* Input data for the test.*/
data tstinput; a = 8;  b = 4; output; a = 10; b = 2; output;
run;

proc ds2;
    ds2_options sas;
    package testpkg /overwrite=yes;
        dcl package pymas py();
        dcl package logger logr('App.tk.MAS');
        dcl varchar(67108864) character set utf8 pycode;
        dcl int rc revision;

        method testpkg( varchar(256) modulename,
                        varchar(256) pyfuncname );
            %addln( '# The first Python function:' )
            %addln( 'def domath1(a, b):' )
            %addln( '  "Output: c, d"' )
            %addln( '  c = a * b' )
            %addln( '  d = a / b' )
            %addln( '  return c, d' )
            %addln( '' )
            %addln( '# Here is the second function:' )
            %addln( 'def domath2(a, b):' )
            %addln( '  "Output: c, d"' )
            %addln( '  c,d = domath1( a, b )' )
            if rc then logr.log( 'E', 'py.appendSrcLine() failed.');
            rc = py.appendSrcLine('  return c, d' );
            pycode = py.getSource();
            revision = py.publish( pycode, modulename );
            if revision lt 1 then
                logr.log( 'E', 'py.publish() failed.');
            rc = py.useMethod( pyfuncname );
            if rc then logr.log( 'E', 'py.useMethod() failed.');
        end;
```

```
        method usefunc( varchar(256) pyfuncname );
            rc = py.useMethod( pyfuncname );
            if rc then logr.log( 'E', 'py.useMethod() failed.');
        end;

        method exec( double a, double b, in_out int rc,
                     in_out double c, in_out double d );
            rc = py.setDouble( 'a', a );   if rc then return;
            rc = py.setDouble( 'b', b );   if rc then return;
            rc = py.execute();             if rc then return;
            c = py.getDouble( 'c' );
            d = py.getDouble( 'd' );
        end;
    endpackage;

    data _null_;
        dcl package logger logr( 'App.tk.MAS' );
        dcl package testpkg t( 'my Py Module Ctxt name', 'domath1' );
        dcl int rc;
        dcl double a b c d;

        method run();
            a = b = c = d = 0.0;
            set tstinput;
            t.exec( a, b, rc, c, d );
            logr.log( 'I', '##### Results: a=$s   b=$s   c=$s   d=$s',
                      a, b, c, d );
        end;

        method term();
            t.usefunc( 'domath2' );
            a = 6; b = 3;
            t.exec( a, b, rc, c, d );
            logr.log( 'I', '##### Results: a=$s   b=$s   c=$s   d=$s',
                      a, b, c, d );
        end;

    enddata;
    run;
quit;
```

Chapter 6

Administration

SAS Micro Analytic Service Logging **39**

Secure DS2 HTTP Package Usage **40**

Monitoring **40**
Monitoring SAS Micro Analytic Service 40
Monitoring SAS Micro Analytic Service Using SAS Environment Manager 42

Start-up Considerations for Clustered Deployments **43**

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 Logging: Configuration and Programming Reference*. Your SAS solution provides a default logging configuration file, and that file specifies loggers and appenders in addition to those described in this chapter.

For example, on UNIX the logging configuration file might be /data1/<**SAS-configuration-directory**>/Lev1/Web/Common/LogConfig/SASMicroAnalyticService-log4sas.xml. For more information, see your solution's documentation.

SAS Micro Analytic Service uses three loggers named App.tk.MAS, App.tk.MAS.Python, 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 start-up, shut-down, and method execution events. App.tk.MAS.CodeGen is used for code compilation and generation logging events, such as compiler warnings and errors. App.tk.MAS.Python is used for logging that is related to Python. Normal operations, such as start-up and shut-down, are logged at the INFO level. Detailed information about operations such as compilation start and finish is 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 ERROR level.

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.

In order to see the data source connection string information that has been logged, set both the App.tk.MAS and Audit.Table.Connection loggers' level to debug.

When diagnosing DS2 problems, it is important to note that the App.TableServices.DS2.Runtime.* and App.TableServices.DS2.Configuration.* loggers do not inherit configuration from their ancestors. They must be configured explicitly, if you want to capture logging events directed to those loggers. It is recommended that you configure them only when diagnosing a DS2 problem since the additional logging traffic affects performance. For more information about those DS2 loggers, see the "DS2 Loggers" section of *DS2 Language Reference*.

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-configuration-directory/*LevN*/Web/WebServer/logs**. These logs have a filename format of error_yyyy-mm-dd.number.log. In them you can find any connection errors between the web server and the tcServer.

The tcServer log is called **SAS-configuration-directory/*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-configuration-directory/*LevN*/Web/WebAppServer/SASServer13_X/logs** logs the activity of GemFire, which is a third-party distributed data management platform. In the event that 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-configuration-directory/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-configuration-directory/Lev1/Web/WebAppServer/SASServer13_1/logs created
at timestamp 1433364164520 version 0 diskStoreId 20d2f45e-876f-4cc1-84b0-ccf6920da3e8]
to be available
```

In a clustered environment, GemFire communicates with the other nodes in the cluster to determine which has the most recent cache. The sas.servers script starts the SASServer13_<**n**> nodes in succession. If starting them in succession is taking too long and encountering time-outs, then first make sure SASServer_1 has initialized, and then start all of the SASServer13 nodes manually. Slightly staggering manual invocations of the SASServer13_<**n**> servers is preferred when sas.server's successive invocation is encountering time-outs. For information on how to prevent this condition, see "Cluster Deployment for SAS Micro Analytic Service" on page 48.

The REST service log file is located at **SAS-configuration-directory/*LevN*/Web/Logs/SASServer13_1/SASMicroAnalyticService2.1.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 SASMicroAnalyticService2.1.log.yyyy-mm-dd. SASMicroAnalyticService2.1.log is created fresh with the first log entry of the day. The service logs are at INFO level. Therefore, they capture start-up entries, module creation, update and deletion boundary entries, as well as any errors from all operations. When there is an error, and more information must be captured to identify the specific cause of the error, update the REST service's Log4j configuration file to set the 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 SASMicroAnalyticService2.1MAS.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 previous 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 SAS Micro Analytic Service.

The SAS Micro Analytic Service Log4j configuration files are located in the directory **SAS-configuration-directory/*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.

Start-up Considerations for Clustered Deployments

SAS Micro Analytic Service is often deployed on a clustered application server to provide high availability and load balancing. Each node of the cluster can host one or more instances of SAS Micro Analytic Service as a web application.

Since SAS Micro Analytic Service uses a GemFire cache for communication between cluster nodes as well as persistence, there are some implications for scripts to start up or shut down member instances. Upon starting up a GemFire cache, cluster members negotiate and determine the member that has the latest persisted copy of the cache. In order to do this, all members must be available.

Therefore, any start-up script must start all cluster members *in parallel* and not *in sequence*. For the Windows platform, the instances are often created as Windows services, which allows for parallel start-up. UNIX-based deployments often use a shell script to restart the instances. Be careful to avoid unneeded dependencies.

If the script starts the members in sequence (that is, if it starts a member only after the preceding instance has started successfully), GemFire waits and delays start-up. Eventually, nodes do start up. However, they might use an incorrect state of the cache.

Chapter 7

Deployment and Tuning

Pre-installation Steps **45**

Deployment **46**
Deploying SAS Micro Analytic Service 46
Adding Whitelist Websites to SAS Micro Analytic Service 47

Post-installation Steps **47**

Cluster Deployment for SAS Micro Analytic Service **48**
Deploying Clusters 48
License Files for Clusters 49

Tuning SAS Micro Analytic Service **50**
Adjust Thread Pool Size 50
Adjust Serial or Parallel Content Creation 50
Adjust DS2 Module Compilation Mode 51
Adjust Session Time-out Value 51
Increase Module ExecutionThroughput of the REST Interface 51
Prevent HTTP Error Messages 51
Creating and Updating Database Connection Strings 52

Pre-installation Steps

Complete the table below and then complete the pre-installation steps before running the SAS Deployment Wizard to install and configure SAS Micro Analytic Service 2.2.

During configuration, you are prompted for the location of your SAS installation data (SID) file. The SID file can be found in the `sid_files` directory of the SAS Software Depot or media. Copy the SID file to a permanent location that can be accessed from all middle-tier machines that run instances of SAS Micro Analytic Service. The license location should be entered as the fully qualified path to the SID file, including the filename of the SID file. For more information, see "License Files for Clusters" on page 49.

SAS Micro Analytic Service 2.2 supports database access. During configuration, you are prompted for database information, depending on the database type that you selected. SAS Micro Analytic Service supports SAS, DB2, Greenplum, Netezza, Oracle, PostgreSQL, Microsoft SQL Server, and Teradata. You can choose not to specify a database for data access by selecting **No Database Data Access**. For more information, see "I/O" on page 9.

The table below lists the information that you must obtain and have available before running the SAS Deployment Wizard.

Description	Default Value	Actual Value
Database Type	No database data access.	
Database Host	*Machine Host Name*	
Database Port	The default port for the database type.	
Database User ID		
Database Password		
The following information is needed if SAS is chosen for data access:		
SAS Library Name		
Path to SAS Data		
The following information is needed if Oracle is chosen for data access:		
Oracle Default Schema Name		
The following information is needed if SQL Server is chosen for data access:		
SQL Server ODBC Data Source Name		
Greenplum, PostgreSQL, DB2, Teradata, or Netezza		
Database Name		

Deployment

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 optimized 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*.

Adding Whitelist Websites to SAS Micro Analytic Service

For information about adding websites that link directly to SAS Micro Analytic Service, see the "Whitelist of Websites and Methods Allowed to Link to SAS Web Applications" section of *SAS 9.4 Intelligence Platform Middle-Tier Administration Guide*.

Post-installation Steps

Open SAS/config/LevN/documents/Instructions.html and follow the steps found in the topic on validation.

Note: Steps 1 and 2 are needed only if you are configuring the middle-tier on an AIX machine.

1. Create the following symbolic links for every middle-tier machine where the SAS Micro Analytic Service REST API has been deployed:

   - libdflic-1.4.so to libdflic-1.4.a
   - libdfssys-1.3.so to libdfssys-1.3.a

   Here is an example:

   ```
   cd <LevConfig>/Web/WebAppServer/SASServer13_#/sas_webapps/
   sas.microanalyticserver.war/WEB-INF/lib/loadlib
   In -s libdflic-1.4.a libdflic-1.4.so
   In -s libdfssys-1.3.a libdfssys-1.3.so
   ```

2. Restart the SASServer13 nodes on every middle-tier machine where the SAS Micro Analytic Service REST API is deployed.

3. Validate the web service URL for the SAS Micro Analytic Service REST API. 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"}]}
   ```

4. When you have completed the validation steps that are located in instructions.html, grant access to the service to a user and add that user as a member of the Decision Manager Users group.

   a. In SAS Management Console, expand **Environment Manager**.

   b. Right-click **User Manager** and click **New** ⇨ **User**.

   c. On the **General** tab, enter the user name and any other optional information.

   d. 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.

   e. On the **Accounts** tab, click **New**.

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

   g. Click **OK** in the New User Properties dialog box.

Note: If connecting an administrative console such as JConsole or JVisualVM causes the server to terminate, add -Xrs to the JVM options and restart the server.

- On UNIX platforms, -Xrs can be added to the JVM_OPTS variable in the setenv.sh file located in ***SAS-configuration-directory/LevN/Web/WebAppServer/SASServer13_N/bin/ directory***.
- On Windows platforms, -Xrs can be added to setenv.bat, located in the directory mentioned above, if you invoke your server using the tcruntime-ctl.bat script. If you invoke the server as a service, add it to the wrapper.conf file located at ***SAS-configuration-dirctory/LevN/Web/WebAppServer/SASServer13_N/conf/wrapper.conf***.

Cluster Deployment for SAS Micro Analytic Service

Deploying Clusters

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. By default, the web server dispatches requests in round-robin to the nodes in the cluster. However, load-balancing policies might be different policy is specified during the web server configuration.

The SAS metadata server for each middle-tier node is specified during deployment. The same metadata server that is referenced by the middle tier can be referenced by the middle-tier nodes. 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 nodes. If different metadata servers are referenced by the middle tier and the middle-tier nodes, any user or 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 middle-tier nodes 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 not be changed. Also, make sure that sufficient 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 the file size 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 nodes should be submitted immediately one after another. The order does not matter.

Note: The GemFire locator must be started cleanly before the other nodes are started. The other nodes should then be stagger started, to reduce the load on the GemFire locator. In addition, it is important to periodically back up the GemFire persistence storage for production systems.

License Files for Clusters

If you publish modules that perform database I/O, your license (SID) file must include licensing for the database access solutions that you intend to use. SID files are not automatically distributed to cluster nodes. When clustering, choose one of these approaches:

- Place your license file on a shared disk and enter the path to it when prompted by the SAS Deployment Wizard.
- Copy the license file to each cluster node, and enter the relative path to the license file when prompted by the SAS Deployment Wizard.

If you choose the second option, you must copy the updated license file to each cluster node whenever your SAS software licenses are renewed or modified. Therefore, placing the license file in a shared directory is recommended.

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 equal to the number of cores in the hosting server. Optimum performance is usually achieved using this number. However, the optimum setting might vary depending on the characteristics of the modules that you publish to 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 2.2**.
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 logical processors, enter 0 for the value.

   For example, specifying 0 on a system that has one Intel quad-core, hyper-threaded processor results in a thread pool size of 8, given that there are two logical processors per core when hyper-threading is on.
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, and are processed in the order of arrival. To allow these operations to be processed in parallel:

1. In SAS Management Console, expand **Application Management**.
2. Expand **SAS Application Infrastructure**.
3. Right-click **SAS Micro Analytic Service 2.2**.
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 inserts a DS2 option in front of each DS2 module to instruct it to use SAS missing value behavior. Although it is not recommended, you can configure the system to use ANSI missing value behavior for DS2 modules. For ANSI behavior, enter **`False`** in step 7 below.

1. In SAS Management Console, expand **Application Management**.
2. Expand **SAS Application Infrastructure**.
3. Right-click **SAS Micro Analytic Service 2.2**.
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**.

Adjust Session Time-out Value

To shorten the amount of time the web server holds on to memory that is used in fulfilling a request, adjust the session time-out value. This allows for a more frequent and shorter garbage collection interval instead of fewer and longer garbage collection intervals that might reduce the responsiveness of the REST server.

Increase Module ExecutionThroughput of the REST Interface

Execution performance can be increased by disabling authentication within the SAS Micro Analytic Service REST server. 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, REST server 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 compete to use the ephemeral ports on the system. Separating them reduces contention for this finite resource.

Creating and Updating Database Connection Strings

To create or update a connection string:

1. In SAS Management Console, expand **Application Management** ⇨ **Configuration Management** ⇨ **SAS Application Infrastructure**.
2. Right-click **SAS Micro Analytic Service 2.2** and select **Properties**.
3. On the **Advanced** tab, update the **masintfc.db.connectionstring** property's value.
4. Click **OK**.

The connection string can contain a federation of multiple connection strings, to enable access to multiple databases. For more information about federated connection strings, see "I/O" on page 9.

Chapter 8

Backup and Restore

Overview **53**

Backup Disk Stores **54**

Restore Script **54**

Additional Backup Considerations **55**
Backup Considerations for 64-Bit Windows 55
Additional Backup Considerations for 64-Bit HP-UX Itanium 56
Additional Backup Steps in a Clustered Environment 56

Common Errors and Remediation **57**

Overview

SAS Micro Analytic Service uses GemFire cache to persist information about modules deployed in it. Like any other persistent storage of content, this store must be backed up regularly, in accordance your organizational policies. In the case of a hardware failure, the content of the cache can be restored from the last good backup to minimize downtime. If backups are not available, all modules have to be redeployed.

GemFire provides an online backup facility where all nodes of the cluster must be operational during the backup process. To restore from a backup, all nodes of the cluster must be shut down.

This chapter describes the backup and restore procedure as it applies to SAS Micro Analytic Service.

The GemFire persistence folder for a SAS Micro Analytic Service system is found in ***`SAS-Configuration-Directory/LevN/Web/WebAppServer/SASServer13_n/logs`***, where *LevN* is the SAS configuration level directory, and *13_n* denotes the application server processes. This is where GemFire holds files that contain an image of the GemFire shared cache.

Before beginning the backup process, see “Additional Backup Considerations” on page 55.

Backup Disk Stores

To begin the backup process, navigate to the GemFire bin directory.

Note: The current directory must be in the path. All tc Server instances hosting the SAS Micro Analytic Service must be running in order to run the backup command.

1. If you have disabled auto-compaction, run manual compaction:

   Note: This step is necessary only for clustered environments.

   ```
   gemfire compact-all-disk-stores
   ```

2. Run the backup command, providing your backup directory location. The following example stores the backed-up files under the **gemfireBackupFilesDirectory**.

```
SAS-Configuration-Directory/LevN/Web/gemfire/bin>gemfire backup /gemfireBackupFilesDirectory
```

   The tool reports on the success of the operation. If the operation is successful, a message similar to the following is generated:

```
SAS-Configuration-Directory/LevN/Web/gemfire/bin>gemfire backup /gemfireBackupFilesDirectory
Connecting to distributed system: locators=<ServerName>[26340]
The following disk stores were backed up:
DiskStore at <ServerName> SAS-Configuration-Directory/LevN/Web/WebAppServer/SASServer13_1/logs
Backup successful.
```

   If the operation does not succeed at backing up all known members, a message similar to the following is generated:

```
SAS-Configuration-Directory/LevN/Web/gemfire/bin>gemfire backup /gemfireBackupFilesDirectory
Connecting to distributed system: mcast=/239.192.81.1:10334
ERROR: Operation "backup" failed because: There are no members in the distributed system.
```

3. To ensure that the backup can be recovered, validate the backed-up files. Run the validate-disk-store command on the backed-up files, for each disk store. Use the full directory path to where the GemFire backup was stored (for example, **/*gemfireBackupFilesDirectory*/<*date*>/<*ServerName*>_v31_13729_16281/diskstores/masgemfire/dir0**).

   Run the validate-disk-store command as follows:

```
SAS-Configuration-Directory/LevN/Web/gemfire/bin>gemfire validate-disk-store masgemfire/
   gemfireBackupFilesDirectory/<date>/<SeverName>_v31_13729_16281/
   diskstores/masgemfire/dir0
```

   Repeat these steps for all disk stores of all members.

Restore Script

The restore script copies files back to their original locations. The backup process creates a folder that is named with the date and time of the backup, as indicated in the above example. This folder can contain one or more subfolders, each corresponding to a folder containing GemFire persistence files. Each such folder contains a restore script

called restore.sh or restore.bat (for example,***gemfireBackupFilesDirectory/<date>/<ServerName>***_v31_13729_16281/restore.bat /).

Note: For a cluster, the GemFire persistence folders can be on different nodes. This has the following implications:

- In order to restore, the restore script must have access to the backup folders as well as the GemFire persistence folders. Therefore, it is recommended that you create the backup in a shared folder that is accessible from every node of the cluster. Then, run the script on the nodes that contain the GemFire persistence folder.
- You might need to modify the restore script since the paths to the GemFire persistence folder can be different on different nodes of the cluster. Because the restore script copies files from the backup folder to the GemFire persistence folder, it can be easily modified to correct the path. You can also copy the files directly, without using the restore script.

Here are best practices for running the restore script:

- Restore your disk stores when your members are offline, and the system is down.
- Read the restore scripts to see where they place the files. Make sure that the destination locations are ready. The restore scripts do not copy over files with the same names. Therefore, delete all files prefixed with BACKUPmasgemfire in the ***SASServer13_n/logs*** folder, after stopping any SASServer13 processes, but before running the restore script (for example, */gemfireBackupFilesDirectory/<date>/<ServerName>*_v31_13729_16281/restore.sh).
- Run the restore scripts. Run each script on the host where the backup originated, as shown in the step above.

The restore process copies disk store files for all stores containing persistent region data back to their original location.

Additional Backup Considerations

Backup Considerations for 64-Bit Windows

The provided Windows distribution of GemFire does not contain the gemfire.properties file. The GemFire script also does not allow the use of -J switches to supply JVM arguments. To run the GemFire backup command, you must extract the locator and license information from the wrapper.conf file for the locator and supply them to the GemFire command line script as JVM arguments. To do this:

1. Locate the GemFire folder under ***SAS-Configuration-Directory*\LevN\Web\gemfire**.
2. Find the instance folder and locate the wrapper.conf file in it. The instance folder is located at ***SAS-Configuration-Directory*\LevN\Web\gemfire\instances**. It is commonly called ins_41415.
3. Locate the following lines containing the parameter values from the wrapper.conf file:
   - set.GEMFIRE_LOCATORS=<ServerName>[41415]
   - set.USE_IPV4_STACK=false
   - set.USE_IPV6_ADDRESS=false

- wrapper.java.additional.2=-Dgemfire.mcast-port=0
- wrapper.java.additional.3=-Dgemfire.license-application-cache=6M0C3-4VW9H-M8J40-0D52F-DTM0H
- wrapper.java.additional.4=-Dgemfire.locators=%GEMFIRE_LOCATORS%
- wrapper.java.additional.5=-Djava.net.preferIPv4Stack=%USE_IPV4_STACK%
- wrapper.java.additional.6=-Djava.net.preferIPv6Addresses=%USE_IPV6_ADDRESS%

4. When you use the above construct, your command line should look like the following:

```
set JAVA_ARGS=-Djava.net.preferIPv4Stack=false
   -Djava.net.preferIPv6Addresses=false -Dgemfire.mcast-port=0
   -Dgemfire.locators=<ServerName>[41415]
   -Dgemfire.license-application-cache=6M0C3-4VW9H-M8J40-0D52F-DTM0H
```

   Substitute the appropriate values for the arguments based on the contents of wrapper.conf. This line must be run before running the GemFire script, so that the utility can find the locator.

Additional Backup Considerations for 64-Bit HP-UX Itanium

This distribution does not contain the gemfire.properties file. Instead, the scripts in the locator instance can be used to define the appropriate values. They can also be used as shell variables, and as JVM arguments.

1. Locate the GemFire folder under ***SAS-Configuration-Directory/LevN*/Web/gemfire**.
2. Find the instance folder and locate the wrapper.conf file in it. The instance folder is located at ***SAS-Configuration-Directory/LevN*/Web/gemfire/instances**. It is commonly called ins_41415.
3. Run the gemfire-locator.sh script, so that it defines the appropriate values as variables in the current shell, as follows:

```
. gemfire-locator.sh
```

   Note: There is a space between . and **gemfire**.

4. Run the GemFire script in the ***SAS-Configuration-Directory/LevN*/Web/gemfire/bin** folder, using the following arguments:

```
-J-Dgemfire.mcast-port=0 -J-Djava.net.preferIPv4Stack=$USE_IPV4_STACK
-J-Djava.net.preferIPv6Addresses=$USE_IPv6_ADDRESS
-J-Dgemfire.locators=$LOCATORS
-J-Dgemfire.license-application-cache=$GEMFIRE_LICENCE_KEY backup
/localdata/config/Lev1/gbkh6i_1
```

Additional Backup Steps in a Clustered Environment

Make sure that all cluster members are running. Before backing up a cluster, run the GemFire utility using the compact-all-disk-stores command. The backup process creates multiple folders containing content from the different cluster members in the destination folder. Use the restore script in each such folder to restore the folder to the appropriate cluster member.

Common Errors and Remediation

The backup process must be run while all the cluster nodes are running. If the backup process is run while some of the nodes are down, you might see error messages, such as `ERROR: Operation "backup" failed because: There are no members in the distributed system`.

The backup process uses GemFire configuration information that has been set up by the SAS installer. There is considerable variation across different platforms regarding how and where this information is stored. Make sure that you are following the correct instructions for invoking the GemFire backup process for your platform. Otherwise, you might see messages, such as `ERROR: Operation "backup" failed because: There are no members in the distributed system`.

The restore process must be run when all of the cluster nodes are shut down. In some operating systems, you might receive errors about locked files.

Chapter 9

Upgrading, Migrating, and Promotion

Upgrading and Migration 59
Promotion 59

Upgrading and Migration

Upgrading refers to updating SAS software and the associated metadata and configuration. For SAS Micro Analytic Service, it specifically means updating the software from SAS Micro Analytic Service 1.2 or 1.3 to 2.2. It is possible to upgrade the software and run on the same hardware deployment.

By contrast, *migration* is a process in which your SAS content and configuration from an earlier SAS release is upgraded to run in a later SAS release. When performed successfully, migration attempts to preserve as much of your current content and configuration as possible, reduce the number of manual migration tasks, and minimize system downtime.

The modules that are deployed in SAS Micro Analytic Service are held in the GemFire cache persistence. Consequently, any upgrade or migration process must be preceded by backing up the stores as explained in Chapter 8, “Backup and Restore,” on page 53.

When the software is upgraded, it is possible to continue to use the same GemFire storage files. If these files have been deleted, they can be restored from a last known good backup.

If the source and target environments are different, in terms of hardware, topology, platform, and so on, all the modules that are deployed must be redeployed. For specific REST API calls to retrieve and repost modules, see “Promotion” on page 59.

Promotion

Promotion is the movement of selected content from a source system to an already configured target system. Sometimes called *partial promotion*, promotion of metadata content is typically used to support movement across development, test, and production environments. It is possible to have multiple such environments depending on the workflow and policies of the organization.

For SAS Micro Analytic Service, *content* refers to the modules that have been deployed.

The source code for the modules can be retrieved from a source system using the REST API call:

```
GET http:/host:port/SASMicroAnalyticService/rest/modules/{moduleId}/source
```

This returns a JSON object that contains the source code of the module.

The same source code can be posted to the target system using the REST API call:

```
POST http:/host:port/SASMicroAnalyticService/rest/modules
```

In SAS Micro Analytic Service 2.2, the module ID is based on the DS2 package name and is not a generated GUID. This ensures that the modules created from the same package have the same module ID.

Chapter 10

SAS Micro Analytic Service REST API

Overview **62**

Terminology **63**
Micro Analytic Service 63
Micro Analytic Module 63
Micro Analytic Step 63
Package 63
Method 63
Signature 63
Input Signature 63
Output Signature 63
Module 63
Module ID 64
Module Name 64
Step 64
Step ID 64
Source Code 64

Client Application Features **64**
Post Load or Create Modules 64
Get Input or Output Step Signatures 64
Post Validate Input Variables 65
Post Execute Modules 65
Put Update Modules 65
Delete Modules 65
Payload Logging 65

Security and Authentication **66**

Life Cycle **66**

Media Types **66**
Externally Defined Media Types 66

SAS Micro Analytic Service Media Types **68**
application/vnd.sas.microanalytic.module 68
application/vnd.sas.microanalytic.module.definition 72
application/vnd.sas.microanalytic.module.source 73
application/vnd.sas.microanalytic.module.step 75
application/vnd.sas.microanalytic.module.step.input 79
application/vnd.sas.microanalytic.module.step.input.validity 80
application/vnd.sas.microanalytic.module.step.output 81

Resources and Collections **83**

Resource / 83
Collection /modules 84
Resource /modules/{moduleId} 96
Resource /modules/{moduleId}/source 104
Collection /modules/{moduleId}/steps 106
Resource /modules/{moduleId}/steps/{stepId} 119

Overview

The SAS Micro Analytic Service REST API provides an interface for web client applications to compile and execute micro analytic modules as steps that provide near real-time analytic capabilities. The REST API supports DS2 modules, including SAS Enterprise Miner models, SAS Business Rules Manager rule flows, and user-written packages. User-written DS2 packages can also publish Python modules and execute Python functions.

The API provides the following POST methods:

Create module
: publishes module 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 step signature
: 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.

Terminology

Micro Analytic Service

A small footprint, near real-time or machine-embedded, execution service providing the ability to embed SAS analytics and business logic 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. In the DS2 language, 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 steps. In the DS2 language, a module is defined as a DS2 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 module, this corresponds to the DS2 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 must 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 module, 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 might contain 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.

Payload Logging

Payload logging enables you to capture the JSON payload, for both input and output, and log it to a file, so that it can be harvested and analyzed.

The data is captured in text files. The files are managed using log4j and can be configured to roll over daily or when the file reaches a particular size. The files are created in the same location as the system log file. The name of the file is SASMicroAnalyticServiceMessages<*version_number*>.log. Every node in the cluster produces its own file. It is possible to change the location by modifying the log4j configuration ID. It is recommended that the file be local to the cluster node, to minimize impact on system performance.

For every invocation of the REST service, a single line that contains the timestamp is logged. The timestamp is followed by a payload JSON object that contains both the request and the response representations. The timestamp is a fixed size. The payload object is not.

Payload logging can be turned on by defining the system property sas.mas.message.audit and setting its value to TRACE. This change can be made in the wrapper.conf file (Windows) or setenv.sh (UNIX) as shown below:

- In setenv.sh (UNIX): **`-Dsas.mas.message.audit=TRACE`**
- In wrapper.conf (Windows): **`wrapper.java.additional.50=-Dsas.mas.message.audit=TRACE`**

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 re-create 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
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

Relationship	HTTP Method	Description
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
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.

Name	Type	Description
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.
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 68. 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.
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 hold additional metadata about the module. 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. 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 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.
type	string	The source code type. The only valid value is text/vnd.sas.source.ds2.

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
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
      }
   ]
}
```


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.

Here is the link relation for the application/vnd.sas.microanalytic.module.step.output media type.

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

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.
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.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.

Here is the link relation for the application/vnd.sas.microanalytic.module.step.output media type.

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

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.
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.
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.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":"next",
         "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.

Note: A module name is reserved during the four minutes while the REST server is creating the module. This prevents name collision in a clustered deployment. Normally, if the module fails to be created, possibly because of incorrect syntax, the

name reservation is released immediately. If the name reservation is not released immediately, you must wait for the reservation to expire before using that name.

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
      }
   ]      },
{
   "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
      }
   ]
},
{
   "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,
      "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
         }
      ]       },
   {
      "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
         }
      ]
   },
   {
      "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
      }
   ]
},
{
   "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
      }
   ]
},
{
   "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
      }
   ]
},
{
   "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
         }
      ]
   },
   {
      "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
         }
      ]
   }
],
"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",
    "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 method requires 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
}
```

Appendix 1

SAS Micro Analytic Service Return Codes

The SAS Micro Analytic Service core component, tkmas, supports the following return codes. Depending on logging settings, an associated message 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 run-time exception. Deleting revision.
-1958743962	MASGarbageCollectionTerm	Garbage collection is freeing control assets during shut-down.
-1958743961	MASShutdownHang	Worker thread did not interrupt after %d seconds during shutdown.
-1958743960	MASGCInvalidIntervalHigh	Specifies that the garbage collection interval is above the maximum. Setting to default value.
-1958743959	MASGCInvalidIntervalLow	Specifies that the garbage collection interval is below the minimum. Setting to default value.
-1958743958	MASGCInvalidGraceHigh	Specifies that the grace period is above the maximum. Setting to default value.
-1958743957	MASGCInvalidGraceLow	Specifies that the grace period is below the minimum. Setting to default value.
-1958743956	MASGCMissingInterval	Garbage collection interval is not specified. Setting to default value.
-1958743955	MASGCMissingGracePeriod	Grace period is not specified. Setting to default value.
-1958743954	MASModuleStats	Check the log for module statistics.
-1958743953	MASInvalidDS2Connection	Attempt to create TKTS driver connection failed.
-1958743952	MASDS2FatalRecompiled	DS2 package fatal error. Auto-recompile succeeded.
-1958743951	MASDS2FatalRecompFailed	DS2 package fatal error. Transaction failed. Recompile failed. Ejecting revision.
-1958743950	MASDS2RevisionEjected	DS2 package fatal error. Max retry exceeded. Ejecting revision. Correct and republish.

Return Code	#define Symbol	Message or Description
-1958743949	MASDBConnLost	Connection to the database lost. Check the log for details.
-1958743948	MASDBConnReestablished	Lost connection reestablished for user context.
-1958743947	MASDBConnRetryLimit	Maximum connection retry attempts exceeded for user context.
-1958743946	MASDBConnDoesNotExist	Attempt to execute SQLStmt, when no connection exists.
-1958743945	MASDBConnRetryThreadErr	Error while creating database connection retry thread.
-1958743944	MASDBConnRetryAttempt	Connection retry attempt unsuccessful.
-1958743943	MASNameRegisterFailed	Unable to register tkmas in the TK named registry. DS2 programs that call Python scripts will not function.
-1958743942	MASDS2PythonNameRequired	AS DS2 Python constructor missing Python module name.
-1958743941	MASDS2PythonCreateError	Unable to create SAS Micro Analytic Service DS2 Python package.
-1958743940	MASDS2PythonInitError	Unable to initialize support for SAS Micro Analytic Service DS2 Python package.
-1958743939	MASUnsupportedFunction	Unsupported function.
-1958743938	MASDS2NotInitialized	Attempt to perform action on uninitialized SAS Micro Analytic Service DS2 Python package.
-1958743937	MASDS2PythonParmError	SAS Micro Analytic Service DS2 Python package parameter mismatch.
-1958743936	MASDS2PythonArgNameReqd	SAS Micro Analytic Service DS2 Python missing argument name.
-1958743935	MASDS2PythonArgValueReqd	AS DS2 Python missing argument value.
-1958743934	MASDS2PythonArgInvalid	SAS Micro Analytic Service DS2 Python invalid argument value.
-1958743933	MASDS2PythonThreadError	Illegal operation: DS2 callback into SAS Micro Analytic Service received an unrecognized thread.
-1958743932	MASPythonCompileEx	Exception thrown while initializing Python or compiling Python script.

Return Code	#define Symbol	Message or Description
-1958743931	MASDS2InvalidMaxRecomp	Invalid maximum DS2 recompile count given. Setting to default value.
-1958743930	MASDBInvalidIntervalHigh	Specified DBMS connection retry interval is above the maximum. Setting to default value.
-1958743929	MASDBInvalidIntervalLow	Specified DBMS connection retry interval is below the minimum. Setting to default value.
-1958743928	MASDBInvalidMaxRetry	Invalid setting for maximum DBMS reconnection attempts. Setting to default value.
-1958743927	MASDBCreateConnErr	SAS Micro Analytic Service failed to create a connection.
-1958743926	MASDBCreateConn	SAS Micro Analytic Service created a connection.
-1958743925	MASGCCanBeDeleted	Garbage collection is checking module context for deletion pending.
-1958743924	MASRepoLockRemovePriv	Locking program repository to remove internal package.
-1958743923	MASRepoUnlockRemovePriv	Released program repository lock after removing internal package.
-1958743922	MASRepoLockRemoveRev	Locking program repository to remove module context.
-1958743921	MASRepoUnlockRemoveRev	Released program repository lock, after removing module context.
-1958743920	MASRepoLockCreate	Creating a lock for user context.
-1958743919	MASRepoLockDestroy	Destroying a lock for user context.
-1958743918	MASRepoLockPackageComp	Locking program repository during compilation of package.
-1958743917	MASRepoUnlockPackageComp	Released program repository lock after compilation of package.
-1958743916	MASRepoUnlockCompCrash	Released program repository lock due to DS2 compiler crash while compiling package.
-1958743915	MASRepoLockPackageSave	Locking program repository to save package after successful compilation.
-1958743914	MASRepoUnlockPackageSave	Released program repository after saving package.

Return Code	#define Symbol	Message or Description
-1958743913	MASRepoLockPackagePriv	Locking program repository to save internal package.
-1958743912	MASRepoUnlockPackagePriv	Released program repository after saving internal package.
-1958743911	MASPythonNotLoaded	Python extension not loaded. Python must be installed in order to execute Python within SAS Micro Analytic Service.
-1958743910	MASTKTSConnHndlFail	Failed to create a table services connection handle.
-1958743909	MASDBDisconnected	SAS Micro Analytic Service disconnected database from user context.
-1958743908	MASDBDisconnect	SAS Micro Analytic Service encountered a failure when attempting to disconnect the database from the user context.
-1958743907	MASPercentS	Internal error. Check the SAS Micro Analytic Service Core log.
-1958743906	MASPythonCompileErr	Error compiling the Python script for the module.
-1958743905	MASDS2MissingArray	A missing array argument is not supported with DS2.
-1958743904	MASDS2EmptyArray	An empty array argument is not supported with DS2.
-1958743903	MASDS2ArrayReplaced	Missing or insufficiently sized DS2 array argument has been replaced with new array of size %d.
-1958743902	MASDS2OutputTransError	Error %d when converting char string of length %d to TKChar string.
-1958743901	MASDS2InputTransError	Error %d when converting TKChar string of length %d to char string.
-1958743900	MASDS2PythonOutputTrans	Error %d when converting Python char string of length %d to TKChar string.
-1958743899	MASDS2PythonInputTrans	Error %d when converting TKChar string of length %d to char string for Python.
-1958743898	MASDBCr8ConnNoSub	SAS Micro Analytic Service created a default data source connection.
-1958743897	MASDBCr8ConnErrNoSub	SAS Micro Analytic Service failed to create a default data source connection.

Return Code	#define Symbol	Message or Description
1958743896	MASDBDisconnNoSub	SAS Micro Analytic Service disconnected from the default data source.
1958743895	MASDBDisconnErrNoSub	SAS Micro Analytic Service encountered a failure when attempting to disconnect from the default data source.
1958743894	MASDS2ScanError	Out of memory or malformed DS2 encountered while scanning the package %s source code prior to dictionary generation.
1958743893	MASDS2ParseError	Out of memory or malformed DS2 encountered while parsing the package %s method %s during dictionary generation.
1958743892	MASMTXGetDictDelPending	Attempt to retrieve the dictionary from module context %s revision %d while deletion pending.
1958743892	MASCFuncProtoNotSupp	Part of the C function prototype is not supported.
1958743890	MASDupModuleName	Module name %s already exists. Module name must be unique within the user context.
1958743889	MASDupDS2Package	The DS2 package name %s is already bound to module %s. Separate modules cannot represent the same DS2 package.
1958743888	MASIndexOutOfRangeSet	The index is out of range while setting an argument. Argument %d specified when number of arguments is %d.
1958743887	MASIndexOutOfRangeGet	The index is out of range while retrieving an argument. Argument %d specified when number of arguments is %d.
1958743886	MASIntTypeExpected	The argument %d in method %Us should be an integral type used to specify the length of the previous argument, which is an array.
1958743885	MASOutArgExpected	The argument %d in method %Us should be an output argument. All input arguments must precede output arguments.
1958743884	MASDS2pymas	DS2 pymas package encountered a failure.
1958743883	MASDS2pymasFailIn	DS2 pymas package encountered a failure in %Us.
1958743882	MASDS2pymasPubUTF8	DS2 pymas package failed to publish module %Us.

Return Code	#define Symbol	Message or Description
1958743881	MASDS2pymasPubTK	DS2 pymas package failed to publish module %s.
1958743880	MASDS2pymasUsed	The DS2 pymas package's use method has already been called on this package instance. Create a separate pymas instances for each method that is used.
1958743879	MASThrdPoolSizeDiff	SAS Micro Analytic Service has already been initialized with a worker thread pool size of %d.
1958743878	MASSymbolTableCreateFailed	SAS Micro Analytic Service failed to create a symbol table.
1958743877	MASMethodExecutionFailed	SAS Micro Analytic Service failed to execute a method.

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 127 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.

Appendix 3

Table Service Driver Reference

DB2 Driver Reference **144**
Understanding the Table Services Driver for DB2 144
Data Service Connection Options for DB2 145
DB2 Wire Protocol Driver Usage Notes 148

FedSQL Driver Reference **149**
Overview 149
Connection Options 149

Greenplum Driver Reference **152**
Understanding the Table Services Driver for Greenplum 152
Data Service Connection Options for Greenplum 152
Greenplum Wire Protocol Driver Usage Notes 156

Netezza Driver Reference **156**
Understanding the Table Services Driver for Netezza 156
Data Service Connection Options for Netezza 157

ODBC Driver Reference **161**
About ODBC 161
Understanding the Table Services Driver for ODBC 161
Data Service Connection Options for ODBC 161
Wire Protocol Driver Usage Notes 167

Oracle Reference **168**
Understanding the Table Services Driver for Oracle 168
Data Service Connection Options for Oracle 168
Oracle Wire Protocol Driver Usage Notes 173

PostgreSQL Driver Reference **173**
Understanding the SAS Federation Server Driver for PostgreSQL 173
Data Service Connection Options for PostgreSQL 174

SAS Data Set Reference **178**
Overview 178
Understanding the Driver for Base SAS 178
Data Service Connection Options for SAS Data Sets 178

Teradata Reference **182**
Understanding the Table Services Driver for Teradata 182
Data Service Connection Options for Teradata 182

DB2 Driver Reference

Understanding the Table Services Driver for DB2

The table services driver for DB2 (driver for DB2) enables table services to read and update legacy DB2 tables. In addition, the driver creates DB2 tables that can be accessed by both table services and the DB2 database management system (DBMS).

The driver for DB2 supports most of the FedSQL functionality. The driver also enables an application to submit native DB2 SQL statements.

The table services driver for DB2 is a remote driver, which means that it connects to a server process in order to access data. The process might be running on the same machine as the table services driver, or it might be running on another machine in the network.

The table services driver for DB2 uses shared libraries that are referenced as shared objects in UNIX. You must add the location of the shared libraries to one of the system environment variables and, if necessary, specify the DB2 version that you have installed. Before setting the environment variables, as shown in the examples below, you must also set the following environment variables:

- The INSTHOME environment variable must be set to your DB2 home directory.
- The DB2DIR environment variable should also be set to the value of INSTHOME.
- The DB2INSTANCE environment variable should be set to the DB2 instance that was configured by the administrator.

```
AIX
Bourne Shell
$ LIBPATH=$INSTHOME/lib:$LIBPATH
$ export LIBPATH
C Shell
$ setenv LIBPATH $INSTHOME/lib:$LIBPATH
HP-UX and HP-UX for the Itanium Processor
  Family Architecture
Bourne Shell
$ SHLIB_PATH=$INSTHOME/lib:$SHLIB_PATH
$ export SHLIB_PATH
C Shell
$ setenv SHLIB_PATH $INSTHOME/lib:$SHLIB_PATH
Linux for Intel Architecture, Linux for x64, Solaris,
  and Solaris for x64
Bourne Shell
$LD_LIBRARY_PATH=$INSTHOME/lib:$LD_LIBRARY_PATH
$ export LD_LIBRARY_PATH
C Shell
$ setenv LD_LIBRARY_PATH $INSTHOME/lib:$LD_LIBRARY_PATH
```

Data Service Connection Options for DB2

Overview

The data service connection arguments for DB2 include connection options and advanced options.

Note: When performing connections through DSNs or connection strings, the FedSQL language processor automatically quotes SQL identifiers that do not meet the regular naming convention as defined in *SAS FedSQL Reference Guide*.

Connection Options

Connection options are used to establish a connection to a data source. Specify one or more connection options. Here is an example:

```
driver=sql;conopts=(driver=db2;uid=myuid;
pwd=Blue31;conopts=(DSN=MYDSN);CATALOG=TSSQL)
```

The table services driver for DB2 supports the following connection options for DB2 data sources.

Option	Description
CATALOG	**CATALOG=*catalog-identifer*;** Specifies an arbitrary identifier for an SQL catalog, which groups logically related schemas. Any identifier is valid (for example, **catalog=DB2**). You must specify a catalog. For the DB2 database, this is a logical catalog name to use as an SQL catalog identifier. *Note:* The FedSQL language processor automatically quotes SQL identifiers that do not meet the regular naming convention as defined in *SAS FedSQL Reference Guide*.
DATABASE|DB	**DATABASE=*database-specification*;** Specifies the name of the DB2 database (for example, **database=sample, DB=sample**). *Note:* You must specify a database name.
DRIVER	**DRIVER=DB2;** Identifies the DB2 data source to which you want to connect. *Note:* You must specify the driver.

Advanced Connection Options

The table services driver for DB2 supports the following advanced connection options for DB2 data sources.

Option	Description
CLIENT_ENCODING	**CLIENT_ENCODING=*encoding-value*** Used to specify the encoding of the DB2CODEPAGE to the DB2 driver. When using this option, you must also set the DB2CODEPAGE environment variable on the client. When the encoding of the DB2 client layer (stored in DBCODEPAGE) is different from the encoding value of the DB2 operating system value, the DB2 client layer attempts to convert incoming data to the DB2 encoding value that is stored in DB2CODEPAGE. To prevent the client layer from converting data incorrectly, you must first determine the correct value for DB2CODEPAGE and then set the CLIENT_ENCODING= option to match the corresponding encoding value in DB2CODEPAGE. For example, suppose you are storing Japanese characters in a DB2 database, and the client machine where the DB2 driver is executing is a Windows machine that is running CP1252 encoding. When the application tries to extract the data into the table services driver, the DB2 client layer attempts to convert these Japanese characters into Latin1 representation, which does not contain Japanese characters. As a result, a garbage character appears in order to indicate a failure in transcoding. To resolve this situation, you must first set the DB2CODEPAGE environment variable value to 1208 (the IBM code page value that matches UTF-8 encoding). That enables you to specify that the DB2 client layer send the data to the application in UTF-8 instead of converting it into Latin1. In addition, you must specify the corresponding encoding value of DB2CODEPAGE because the table services driver for DB2 cannot derive this information from a DB2 session. For this particular Windows case, set the CLIENT_ENCODING= option to the UTF-8 encoding in order to match the DB2CODEPAGE value (1208) and also to specify the DB2CODEPAGE value to the DB2 driver. However, changing the value of DB2CODEPAGE affects all applications that run on that machine. You should reset the value to the usual DB2CODEPAGE value, which was derived when the database was created. *Note:* Setting the DB2CODEPAGE value or the CLIENT_ENCODING= value incorrectly can cause unpredictable results. You should set these values only when a situation such as the example above occurs. *Note:* You can specify any valid encoding value for CLIENT_ENCODING=option.
CT_PRESERVE	**CT_PRESERVE=STRICT | SAFE | FORCE | FORCE_COL_SIZE** Enables users to control how data types are mapped. Note that data type mapping is disabled when CT_PRESERVE is set to STRICT. If the requested type does not exist on the target database, an error is returned. Here are the options: • **STRICT** The requested type must exist in the target database. No type promotion occurs. If the type does not exist, an error is returned. • **SAFE** Target data types are upscaled only if they do not result in a loss of precision or scale. When character encodings are changed, the new column size is recalculated to ensure that all characters can be stored in the new encoding. • **FORCE** This is the default for all drivers. The best corresponding target data type is chosen, even if it could potentially result in a loss of precision or scale. When character encodings are changed, the new column size is recalculated to ensure that all characters can be stored in the new encoding. • **FORCE_COL_SIZE** This option is the same as FORCE, except that the column size for the new encoding is the same as the original encoding. This option can be used to avoid column size creep. However, the resulting column might be too large or too small for the target data.

Option	Description
DEFAULT_ATTR	**DEFAULT_ATTR=(*attr=value;...*)** Used to specify connection handle or statement handle attributes that are supported for initial connect-time configuration, where **attr=value** corresponds to any of the following options: • **CURSORS=*n*** - Connection handle option. This option controls the driver's use of client-side, result set cursors. The possible values are 0, 1, or 2. 0 Causes the driver to use client-side static cursor emulation if a scrollable cursor is requested but the database server cannot provide one. 1 Causes the driver to always use client-side static cursor emulation if a scrollable cursor is requested. The database server's native cursor is not used. 2 (Default) Causes the driver to never use client-side static cursor emulation if a scrollable cursor is requested. The database server's native cursor is used if available. Otherwise, the cursor is forward-only. Example: **DEFAULT_ATTR=(CURSORS=2)** • **USE_EVP=*n*** - Statement handle option. This option optimizes the driver for large result sets. The possible values are 0 (OFF) or 1 (ON), which is the default. Example: **DEFAULT_ATTR=(USE_EVP=0)** • **XCODE_WARN=*n*** - Statement handle option. Used to warn about possible character transcoding errors that occur during row input or output operations. Possible values are 0 (returns an error), 1 (returns a warning), or 2 (ignore transaction errors). 0 is the default. Example: **DEFAULT_ATTR=(XCODE_WARN=1)**
DRIVER_TRACE	**DRIVER_TRACE='API | SQL | ALL';** Requests tracing information, which logs transaction records to an external file that can be used for debugging purposes. The driver writes a record of each command that is sent to the database to the trace log based on the specified tracing level, which determines the type of tracing information. Here are the tracing levels: • **API** Specifies that API method calls be sent to the trace log. This option is most useful if you are having a problem and need to send a trace log to SAS Technical Support for troubleshooting. • **SQL** Specifies that SQL statements that are sent to the database management system (DBMS) be sent to the trace log. Tracing information is DBMS specific, but most table services drivers log SQL statements such as SELECT and COMMIT. • **ALL** Activates all trace levels. • **DRIVER** Specifies that driver-specific information be sent to the trace log. **Default:** Tracing is not activated. *Note:* If you activate tracing, you must also specify the location of the trace log with DRIVER_TRACEFILE=. Note that DRIVER_TRACEFILE= is resolved against the TRACEFILEPATH set in ALTER SERVER. TRACEFILEPATH is relative to the server's content root location. (Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=. **Interaction:** You can specify one trace level, or you can concatenate more than one by including the | (OR) symbol. For example, **driver_trace='api|sql'** generates tracing information for API calls and SQL statements.

Option	Description
DRIVER_TRACEFILE	**DRIVER_TRACEFILE=*'filename'***; Used to specify the name of the text file for the trace log. Include the filename and extension in single or double quotation marks (for example, **driver_tracefile='\mytrace.log'**). **Default:** The default TRACEFILE location applies to a relative filename, and it is placed relative to TRACEFILEPATH. **Requirement:** DRIVER_TRACEFILE is required when activating tracing using DRIVER_TRACE. **Interaction:** (Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=.
DRIVER_TRACEOPTIONS	**DRIVER_TRACEOPTIONS=APPEND | THREADSTAMP | TIMESTAMP;** Specifies options in order to control formatting and other properties for the trace log: • **APPEND** Adds trace information to the end of an existing trace log. The contents of the file are not overwritten. • **TIMESTAMP** Prepends each line of the trace log with a time stamp. • **THREADSTAMP** Prepends each line of the trace log with a thread identification. **Default:** The trace log is overwritten with no thread identification or time stamp.
PASSWORD	**PWD=*password*** Specifies the password for DB2.
UID	**UID=*user-id*;** Specifies the DB2 login user ID.

DB2 Wire Protocol Driver Usage Notes

There are a number of third-party wire protocol ODBC drivers that communicate directly with a database server, without having to communicate through a client library. When you configure the ODBC drivers on Windows or UNIX, you can set certain options. SAS runs best when these options are selected. Some, but not all, are selected by default.

Windows	The options are located on the **Advanced** or **Performance** tabs in the ODBC Administrator.
UNIX	The options are available when configuring data sources using the ODBC Administrator tool. Values can also be set by editing the **odbc.ini** file in which their data sources are defined.

Note: A DSN configuration that uses a wire protocol driver with the catalog option selected returns only the schemas that have associated tables or views. To list all existing schemas, create a DSN without selecting the catalog option.

When configuring an ODBC DSN using the DB2 Wire Protocol driver, set the following advanced option:

- **Application Using Threads**

FedSQL Driver Reference

Overview

The FedSQL language driver supports the FedSQL dialect, as documented in *SAS FedSQL Language Reference Guide*. When loaded, the FedSQL driver parses SQL requests, and then sends the parsed query to the appropriate data source driver to determine whether the functionality can be handled by the data service. The FedSQL driver includes an SQL processor that supports the FedSQL dialect. The main emphasis of the FedSQL driver is to support federation of data sources. For example, if an SQL submission is requesting data from DB2 to be joined with data from Oracle, the SQL processor requests the data from the data sources and then performs the join. The FedSQL driver supports the FedSQL dialect regardless of the data source it comes from. For example, if the SQL request is from a single data source that does not support a particular SQL function, the FedSQL processor guarantees implementation of the request.

Connection Options

- CONOPTS=((connection string 1);(connection string 2); ... (connection string <*n*>)) - Specifies one or more data source connection strings. For example, the following illustrates a federated connection string including Oracle, Teradata, Netezza, and Base SAS data sources:

```
driver=sql;conopts=((driver=oracle;catalog=acat;uid=myuid;
pwd=myPass9;path=oraclev11.abc.123.com:1521/ORA11G);
(driver=teradata;catalog=bcat;uid=model;
pwd='{sas002}C5DDFFF91B5D31DFFFCE9FFF';
server=terasoar;database=model);(driver=netezza;uid=myuid;
pwd=myPass2;server=mysrvr;database=testdb;catalog=(ccat={TEST}));
(driver=base;catalog=dcat;schema=(name=dblib;primarypath=/u/mypath/mydir)))
```

- DEFAULT_CATALOG=*catalog-name* - Used to specify the name of the catalog to set as the current catalog upon connecting. This option is useful for SQL Server connections and federated connections.
- DEFAULT_ATTR=(*attr=value*;...) - Used to specify connection handle or statement handle attributes supported for initial connect-time configuration., where **`attr=value`** corresponds to any of the following options:

SQL_CURSORS=*n*

FedSQL connection handle option. This option controls the driver's use of client-side, result set cursors. The possible values are 0, 1, or 2.

- A value of 0 causes the driver to use client-side static cursor emulation if a scrollable cursor is requested but the database server cannot provide one.
- A value of 1 causes the driver to always use client-side static cursor emulation if a scrollable cursor is requested. The database server's native cursor is never used.
- A value of 2 (default) causes the driver to never use client-side static cursor emulation if a scrollable cursor is requested. The database server's native cursor is used if available, otherwise the cursor is forward only.

```
DEFAULT_ATTR=(SQL_CURSORS=2)
```

SQL_AC_BEHAVIOR=*n*

FedSQL connection handle option. Specifies whether FedSQL should use transactions when processing complex operations (for example, **"CREATE TABLE xxx AS SELECT yyy FROM zzz"** or a multi-row delete statement that requires multiple operations to delete the underlying rows). Possible values are 0 (default), 1, and 2.

- A value of **0** (default) means that no transactions are attempted under-the-covers and operations such as emulated UPDATE, DELETE, or INSERT are not guaranteed to be atomic.
- A value of **1** means that FedSQL tries to use transactions to better support the correct behavior when AUTOCOMMIT is set to ON (where individual operations like UPDATE, DELETE, and INSERT should be atomic).
- A value of **2** means that transactions are required. This option fails if the underlying drivers do not support transactions.

```
DEFAULT_ATTR=(SQL_AC_BEHAVIOR=0)
```

SQL_MAX_COL_SIZE=*n*

FedSQL statement handle option. Enables a user to specify the size of the **varchar** or **varbinary** that is used for potentially truncated long data when direct bind is not possible.

- The default value is 32767.
- The limit for this size is 1 MG. If the value exceeds 1 MG, FedSQL resets the value and returns an **Option value changed** warning.

```
DEFAULT_ATTR=(SQL_MAX_COL_SIZE=1048576)
```

SQL_PUSHDOWN=*n*

FedSQL statement handle option. This option tells FedSQL if and when it should try to push down SQL to the underlying driver. The values are 8, 2, or 0 (default).

- A value of 8: (PLAN_FORCE_PUSHDOWN_SQL) - Complete statement pushdown is required. If that is not possible, the INSERT, UPDATE, DELETE, or CREATE TABLE AS statement fails.
- A value of 2: (PLAN_DISABLE_PUSHDOWN_SQL) - Specifies that the INSERT, UPDATE, DELETE, or CREATE TABLE AS statement not be pushed down to the underlying driver.
- A value of 0 (default): Specifies that the FedSQL processor determine whether the INSERT, UPDATE, DELETE, or CREATE TABLE AS statement should be pushed down to the underlying driver.

```
DEFAULT_ATTR=(SQL_PUSHDOWN=0)
```

SQL_STMT_MEM_LIMIT=*n*

FedSQL statement handle option. Used to control the amount of memory that is available to FedSQL to answer SQL requests.

- (*n*) is treated as an integer and is specified in bytes.
- The following example allows 200 MB of memory:

```
DEFAULT_ATTR=(SQL_STMT_MEM_LIMIT=209715200)
```

SQL_TXN_EXCEPTIONS=*n*

FedSQL connection handle option. Supports dynamic connections regardless of the specified transaction isolation. Possible values are 0 or 2 (default).

- Specify a value of 0 to disable support for dynamic connections.
- Specify a value of 2 to enable support for dynamic connections.

```
DEFAULT_ATTR=(SQL_TXN_EXCEPTIONS=2)
```

SQL_USE_EVP=*n*

FedSQL statement handle option. This option optimizes the driver for large result sets. The possible values are 0 or 1 (default) and are used as follows:

- Specify 0 to turn optimization OFF.
- Specify 1 to enable optimization (ON).

```
DEFAULT_ATTR=(SQL_USE_EVP=0)
```

SQL_VDC_DISABLE=*n*

FedSQL statement handle option. This option is used to allow or disallow use of cached data for a statement. The possible values are 0 (default) or 1 and are used as follows:

- Specify a value of 0 to enable cached data.
- Specify a value of 1 to disable cached data.

```
DEFAULT_ATTR=(SQL_VDC_DISABLE=1)
```

SQL_XCODE_WARN=*n*

FedSQL statement handle option. Used to warn when there is an error while transcoding data during row input or output operations. Possible values are 0 (default), 1, or 2 and are used as follows:

- Specify 0 to return an error if data cannot be transcoded.
- Specify 1 to return a warning if data cannot be transcoded.
- Specify 2 to ignore transcoding errors.

```
DEFAULT_ATTR=(SQL_XCODE_WARN=1)
```

Greenplum Driver Reference

Understanding the Table Services Driver for Greenplum

The table services driver (driver for Greenplum) enables table services to read and update Greenplum tables. In addition, the driver creates Greenplum tables that can be accessed by both table services and Greenplum.

The driver for Greenplum supports most of the FedSQL functionality. The driver also enables an application to submit native Greenplum SQL statements.

The table services driver for Greenplum is a remote driver, which means that it connects to a server process in order to access data. The process might be running on the same machine as the table services, or it might be running on another machine in the network.

The table services driver for Greenplum uses shared libraries that are referenced as shared objects in UNIX. You must add the location of the shared libraries to one of the system environment variables, and set any other environment variables required by the Greenplum client libraries. The following Korn shell commands provide an example:

```
export ODBCHOME=/dbi/odbc/gpl94m3
export ODBCINI=/dbi/odbc/gpl94m3/odbc.ini
export ODBCINST=/dbi/odbc/gpl94m3/odbcinst.ini
export GPHOME_LOADERS=/dbi/greenplum/4.2.6/gpfdist
export GPLOAD_HOST=mynode.abc.123.com
export GPLOAD_HOME=/tmp
LD_LIBRARY_PATH=/dbi/odbc/gpl94m3/lib:${LD_LIBRARY_PATH}
LD_LIBRARY_PATH=${LD_LIBRARY_PATH%:}
export LD_LIBRARY_PATH
```

Data Service Connection Options for Greenplum

Connection Options

Connection options are used to establish a connection to a data source. Specify one or more connection options when defining a data service. Here is an example:

```
driver=sql;conopts=(driver=greenplum;uid=myuid;
pwd=MyPasswd;server=greenlight;port=5432;
database=sample;catalog=acat)
```

The driver for Greenplum supports the following connection options.

Option	Description
CATALOG	**CATALOG=*catalog-identifier*;** Specifies an arbitrary identifier for an SQL catalog, which groups logically related schemas. Any identifier is valid (for example, **catalog=gps_test**). You must specify a catalog. For the Greenplum database, this is a logical catalog name to use as an SQL catalog identifier. *Note:* SAS Federation Server automatically quotes SQL identifiers that do not meet the regular naming convention as defined in *SAS FedSQL Reference Guide*.

Option	Description
DATABASE	**DATABASE=*database_name*;** Identifies the database to which you want to connect, which resides on the server that was previously specified by the SERVER option.
DRIVER	**DRIVER=GREENPLUM**; Specifies the data service for the Greenplum database to which you want to connect. You must specify a driver.
DSN	**DSN=*data_source_identifer*;** Identifies the data source name to which you want to connect.
SERVER	**SERVER=*server_name*;** Identifies the name of the server where the Greenplum database resides.

Advanced Connection Options

The driver for Greenplum supports the following advanced connection options.

Option	Description
ALLOW_UNQUOTED_NAMES	**ALLOW_UNQUOTED_NAMES=NO**|**YES**; Specifies whether to enclose table and column names in quotation marks. Tables and columns are quoted when this option is set to NO (default). If the option is set to YES, the driver will not automatically add quotation marks to table and column names if they are not specified. This allows Greenplum tables and columns to be created in the default lowercase.
CLIENT_ENCODING	**CLIENT_ENCODING=cei**; Specifies an encoding, different from the default, to use on the client.
CT_PRESERVE	**CT_PRESERVE = STRICT** | **SAFE** | **FORCE** | **FORCE_COL_SIZE** Enables users to control how data types are mapped. Note that data type mapping is disabled when CT_PRESERVE is set to STRICT. If the requested type does not exist on the target database, an error is returned. Here are the options: • **STRICT** The requested type must exist in the target database. No type promotion occurs. If the type does not exist, an error is returned. • **SAFE** Target data types are upscaled only if they do not result in a loss of precision or scale. When character encodings are changed, the new column size is recalculated to ensure all characters can be stored in the new encoding. • **FORCE** This is the default for all drivers. The best corresponding target data type is chosen, even if it could potentially result in a loss of precision or scale. When character encodings are changed, the new column size is recalculated to ensure that all characters can be stored in the new encoding. • **FORCE_COL_SIZE** This option is the same as FORCE, except that the column size for the new encoding is the same as the original encoding. This option can be used to avoid column size creep. However, the resulting column might be too large or too small for the target data.

Option	Description
DEFAULT_ATTR	DEFAULT_ATTR=(*attr=value*;...) Used to specify connection handle or statement handle attributes supported for initial connect-time configuration, where **attr=value** corresponds to any of the following options: • **CURSORS=*n*-** Connection handle option. This option controls the driver's use of client-side, result set cursors. The possible values are 0, 1, or 2. 0 Causes the driver to use client-side static cursor emulation if a scrollable cursor is requested but the database server cannot provide one. 1 Causes the driver to always use client-side static cursor emulation if a scrollable cursor is requested. The database server's native cursor is not used. 2 (Default) Causes the driver to never use client-side static cursor emulation if a scrollable cursor is requested. The database server's native cursor is used if available. Otherwise, the cursor is forward-only. Example: **DEFAULT_ATTR=(CURSORS=2)** • **USE_EVP=*n*** - Statement handle option. This option optimizes the driver for large result sets. The possible values are 0 (OFF) or 1 (ON), which is the default. Example: **DEFAULT_ATTR=(USE_EVP=0)** • **XCODE_WARN=*n*** - Statement handle option. Used to warn about possible character transcoding errors that occur during row input or output operations. Possible values are 0 (returns an error), 1 (returns a warning), or 2 (ignore transaction errors). 0 is the default. Example: **DEFAULT_ATTR=(XCODE_WARN=1)**
DRIVER_TRACE	**DRIVER_TRACE='API | SQL | ALL';** Requests tracing information, which logs transaction records to an external file that can be used for debugging purposes. The SAS Federation Server driver writes a record of each command that is sent to the trace log based on the specified tracing level, which determines the type of tracing information. Here are the tracing levels: • **ALL** Activates all trace levels. • **API** Specifies that API method calls be sent to the trace log. This option is most useful if you are having a problem and need to send a trace log to SAS Technical Support for troubleshooting. • **DRIVER** Specifies that driver-specific information be sent to the trace log. • **SQL** Specifies that SQL statements that are sent to the database management system (DBMS) be sent to the trace log. Tracing information is DBMS specific, but most table services drivers log SQL statements such as SELECT and COMMIT. **Default:** Tracing is not activated. *Note:* If you activate tracing, you must also specify the location of the trace log with DRIVER_TRACEFILE=. Note that DRIVER_TRACEFILE= is resolved against the TRACEFILEPATH set in ALTER SERVER. TRACEFILEPATH is relative to the server's content root location. (Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=. **Interaction:** You can specify one trace level, or you can concatenate more than one by including the | (OR) symbol. For example, **driver_trace='api|sql'** generates tracing information for API calls and SQL statements.

Option	Description
DRIVER_TRACEFILE	**DRIVER_TRACEFILE=*'filename'*;** Used to specify the name of the text file for the trace log. Include the filename and extension in single or double quotation marks (for example, **driver_tracefile='\mytrace.log'**). **Default:** The default TRACEFILE location applies to a relative filename, and it is placed relative to TRACEFILEPATH. **Requirement:** DRIVER_TRACEFILE is required when activating tracing using DRIVER_TRACE. **Interaction:** (Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=.
DRIVER_TRACEOPTIONS	**DRIVER_TRACEOPTIONS=APPEND** | **THREADSTAMP** | **TIMESTAMP;** Specifies options in order to control formatting and other properties for the trace log: • **APPEND** Adds trace information to the end of an existing trace log. The contents of the file are not overwritten. • **THREADSTAMP** Prepends each line of the trace log with a thread identification. • **TIMESTAMP** Prepends each line of the trace log with a time stamp. **Default:** The trace log is overwritten with no thread identification or time stamp.
MAX_BINARY_LEN	**MAX_BINARY_LEN=*value*;** Specifies a value to limit the length of long binary fields (LONG VARBINARY). As opposed to other databases, Greenplum does not have a size limit for long binary fields.
MAX_CHAR_LEN	**MAX_CHAR_LEN=*value*;** Specifies a value to limit the length of character fields (CHAR and VARCHAR). As opposed to other databases, Greenplum does not have a size limit for character fields.
MAX_TEXT_LEN	**MAX_TEXT_LEN=*value*;** Specifies a value to limit the length of long character fields (LONG VARCHAR). As opposed to other databases, Greenplum does not have a size limit for long character fields.
NUM BYTES PER CHAR	**NUMBYTESPERCHAR=*value*;** Specifies the default number of bytes per character.
PASSWORD	**PASSWORD=*password*;** Specifies a password for the ID passed through the USER= option. The alias is PWD=. *Note:* You must specify the PASSWORD= option.
SCHEMA	**SCHEMA=*value*;** Specifies the default schema for the connection. If the option is not specified, the schema (or list of schemas) is determined based on the value of the schema search path defined on the database server.
STRIP_BLANKS	**STRIP_BLANKS=*value*;** Specifies whether to strip blanks from character fields.

Option	Description
USER	**USER=*user-id*;** Specifies a Greenplum user ID. If the ID contains blanks or national characters, enclose it in quotation marks. The alias is UID=. *Note:* You must specify the USER= option.

Greenplum Wire Protocol Driver Usage Notes

There are a number of wire protocol ODBC drivers that communicate directly with a database server, without having to communicate through a client library. When you configure the ODBC drivers on Windows or UNIX, you can set certain options. SAS runs best when these options are selected. Some, but not all, are selected by default.

Windows	The options are located on the **Advanced** or **Performance** tabs in the ODBC Administrator.
UNIX	The options are available when configuring data sources using the ODBC Administrator tool. Values can also be set by editing the **odbc.ini** file in which their data sources are defined.

Note: A DSN configuration that uses a wire protocol driver with the catalog option selected returns only the schemas that have associated tables or views. To list all existing schemas, create a DSN without selecting the catalog option.

When configuring an ODBC DSN using the Greenplum Wire Protocol driver, select the following advanced options:

- **Application Using Threads**
- **Enable SQLDescribeParam**
- **Fetch TSFS as Time**
- **Fetch TSWTZ as Timestamp**

Netezza Driver Reference

Understanding the Table Services Driver for Netezza

The table services driver for Netezza (driver for Netezza) enables table services to read and update legacy Netezza tables. In addition, the driver creates Netezza tables that can be accessed by both table services and Netezza.

The driver for Netezza supports most of the FedSQL functionality. The driver also enables an application to submit native Netezza SQL statements.

The driver for Netezza is a remote driver, which means that it connects to a server process in order to access data. The process might run on the same machine as table services, or it might run on another machine in the network.

The table services driver for Netezza uses shared libraries that are referenced as shared objects in UNIX. You must add the location of the shared libraries to one of the system environment variables, and set any other environment variables required by the Netezza client libraries. The following Korn shell commands provide an example:

```
LD_LIBRARY_PATH=/dbi/netezza/7.0.4/lib64:${LD_LIBRARY_PATH}
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH%:}
export ODBCINI=/env/netezza/odbc.ini
export NZ_ODBC_INI_PATH=/env/netezza
```

Data Service Connection Options for Netezza

Overview

To access data that is hosted on table services, a client must submit a connection string, which defines how to connect to the data. The data service connection arguments for Netezza include connection options and advanced options.

Connection Options

Connection options are used to establish a connection to a data source. Specify one or more connection options when defining a data service. Here is an example:

```
driver=sql;conopts=(driver=netezza;uid=myid2;
pwd=mypwd2;server=mysrvr;database=mydb;
catalog=(bcat={TEST}))
```

The driver for Netezza supports the following connection options.

Option	Description
CATALOG	**CATALOG=*catalog-identifier*;** Specifies an arbitrary identifier for an SQL catalog, which groups logically related schemas. Any identifier is valid. *Note:* Table services automatically quotes SQL identifiers that do not meet the regular naming convention as defined in *SAS FedSQL Reference Guide*.
DATABASE	**DATABASE=*database–name*;** Identifies the database to which you want to connect, which resides on the server previously specified through the SERVER option.
DRIVER	**DRIVER=*NETEZZA*;** Specifies the data service for the Netezza database to which you want to connect. *Note:* You must specify the driver.
CONOPTS	**CONOPTS=(*ODBC–compliant database connection string*);** Specifies an ODBC-compliant database connection string using ODBC-style syntax. These options, combined with the ODBC_DSN option, must specify a complete connection string to the data source. If you include a DSN= or FILEDSN= specification within the CONOPTS= option, do not use the ODBC_DSN= connection option. However, you can specify the ODBC database-specific connection options by using CONOPTS=. Then you can specify an ODBC DSN that contains other connection information by using the ODBC_DSN= connection option.

Option	Description
DSN	**DSN=*data_source_identifer;*** Identifies the data source name to which you want to connect.
SERVER	**SERVER=*server_name;*** Identifies the name of the server where the Netezza database resides.
PORT	**PORT=*port_number*** Identifies the listen port of the server where the Netezza database resides.

Advanced Connection Options

The driver for Netezza supports the following advanced connection options.

Option	Description
CLIENT_ENCODING	**CLIENT_ENCODING=*cei*** Used to specify encoding for the client.
CT_PRESERVE	**CT_PRESERVE=STRICT** | **SAFE** | **FORCE** | **FORCE_COL_SIZE** Enables users to control how data types are mapped. Note that data type mapping is disabled when CT_PRESERVE is set to STRICT. If the requested type does not exist on the target database, an error is returned. Here are the options: • **STRICT** The requested type must exist in the target database. No type promotion occurs. If the type does not exist, an error is returned. • **SAFE** Target data types are upscaled only if they do not result in a loss of precision or scale. When character encodings are changed, the new column size is recalculated to ensure that all characters can be stored in the new encoding. • **FORCE** This is the default for all drivers. The best corresponding target data type is chosen, even if it could potentially result in a loss of precision or scale. When character encodings are changed, the new column size is recalculated to ensure that all characters can be stored in the new encoding. • **FORCE_COL_SIZE** This option is the same as FORCE, except that the column size for the new encoding is the same as the original encoding. This option can be used to avoid column size creep. However, the resulting column might be too large or too small for the target data.

Option	Description
DEFAULT_ATTR	**DEFAULT_ATTR=(*attr=value*;...)** Used to specify connection handle or statement handle attributes that are supported for initial connect-time configuration, where **attr=value** corresponds to any of the following options: • **CURSORS=*n***- Connection handle option. This option controls the driver's use of client-side, result set cursors. The possible values are 0, 1, or 2. 0 Causes the driver to use client-side static cursor emulation if a scrollable cursor is requested but the database server cannot provide one. 1 Causes the driver to always use client-side static cursor emulation if a scrollable cursor is requested. The database server's native cursor is not used. 2 (Default) Causes the driver to never use client-side static cursor emulation if a scrollable cursor is requested. The database server's native cursor is used if available. Otherwise, the cursor is forward-only. Example: **DEFAULT_ATTR=(CURSORS=2)** • **USE_EVP=*n*** - Statement handle option. This option optimizes the driver for large result sets. The possible values are 0 (OFF) or 1 (ON), which is the default. Example: **DEFAULT_ATTR=(USE_EVP=0)** • **XCODE_WARN=*n*** - Statement handle option. Used to warn about possible character transcoding errors that occur during row input or output operations. Possible values are 0 (returns an error), 1 (returns a warning), or 2 (ignore transaction errors). 0 is the default. Example: **DEFAULT_ATTR=(XCODE_WARN=1)**
DRIVER_TRACE	**DRIVER_TRACE='API | SQL | ALL';** Requests tracing information, which logs transaction records to an external file that can be used for debugging purposes. The driver writes a record of each command that is sent to the database to the trace log based on the specified tracing level, which determines the type of tracing information. Here are the tracing levels: • **ALL** Activates all trace levels. • **API** Specifies that API method calls be sent to the trace log. This option is most useful if you are having a problem and need to send a trace log to SAS Technical Support for troubleshooting. • **DRIVER** Specifies that driver-specific information be sent to the trace log. • **SQL** Specifies that SQL statements that are sent to the database management system (DBMS) be sent to the trace log. Tracing information is DBMS specific, but most table services drivers log SQL statements such as SELECT and COMMIT. **Default:** Tracing is not activated. *Note:* If you activate tracing, you must also specify the location of the trace log with DRIVER_TRACEFILE=. Note that DRIVER_TRACEFILE= is resolved against the TRACEFILEPATH set in ALTER SERVER. TRACEFILEPATH is relative to the server's content root location. (Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=. **Interaction:** You can specify one trace level, or you can concatenate more than one by including the | (OR) symbol. For example, **driver_trace='api|sql'** generates tracing information for API calls and SQL statements.

Option	Description
DRIVER_TRACEFILE	**DRIVER_TRACEFILE=*'filename'***; Used to specify the name of the text file for the trace log. Include the filename and extension in single or double quotation marks (for example, **driver_tracefile='\mytrace.log'**). **Default:** The default TRACEFILE location applies to a relative filename, and it is placed relative to TRACEFILEPATH. **Requirement:** DRIVER_TRACEFILE is required when activating tracing using DRIVER_TRACE. **Interaction:** (Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=.
DRIVER_TRACEOPTIONS	**DRIVER_TRACEOPTIONS=APPEND | THREADSTAMP | TIMESTAMP;** Specifies options in order to control formatting and other properties for the trace log: • **APPEND** Adds trace information to the end of an existing trace log. The contents of the file are not overwritten. • **THREADSTAMP** Prepends each line of the trace log with a thread identification. • **TIMESTAMP** Prepends each line of the trace log with a time stamp. **Default:** The trace log is overwritten with no thread identification or time stamp.
USER	**USER=*"user-id";*** Specifies a Netezza user ID. If the ID contains blanks or national characters, enclose it in quotation marks. Alias: UID. *Note:* You must specify the USER option.
PASSWORD	**PASSWORD=*password;*** Specifies a password for the ID passed through the USER= option. Alias: PWD. *Note:* You must specify the PASSWORD option with USER.
STRIP_BLANKS	**STRIP_BLANKS=YES|NO;** Specifies whether to strip blanks from character fields.
READONLY	**READONLY=YES|NO;** Specifies whether to connect to the Netezza database in Read-Only mode. The default is NO. Alias: READ_ONLY
SHOWSYSTEMTABLES	**SHOWSYSTEMTABLES=YES|NO;** Specifies whether tables are included in the available table list. If set to YES or TRUE, system tables are included in the available table list. The default setting is NO. Alias: SST
NUMBERBYTESPERCHARACTER	**NUMBYTESPERCHAR=*value;*** Specifies the default number of bytes per character.

ODBC Driver Reference

About ODBC

This section provides functionality details and guidelines for the open database connectivity (ODBC) databases that are supported by the table services driver for ODBC (driver for ODBC).

ODBC standards provide a common interface to a variety of databases, including dBASE, Microsoft Access, Oracle, Paradox, and Microsoft SQL Server databases. Specifically, ODBC standards define APIs that enable an application to access a database if both the application and the database conform to the specification. ODBC also provides a mechanism to enable dynamic selection of a database that an application is accessing. As a result, users can select databases other than those that are specified by the application developer.

Understanding the Table Services Driver for ODBC

The driver for ODBC enables table services to read and update legacy ODBC database tables. In addition, the driver creates tables that can be accessed by both table services and an ODBC database.

The driver for ODBC supports most of the FedSQL functionality. The driver also enables an application to submit native database-specific SQL statements.

The driver for ODBC is a remote driver, which means that it connects to a server process in order to access data. The process might be running on the same machine as table services, or it might be running on another machine in the network.

Data Service Connection Options for ODBC

Overview

To access data that is hosted on table services, a client must submit a connection string, which defines how to connect to the data. The data service connection arguments for an ODBC-compliant database include connection options and advanced connection options.

To configure ODBC data sources, you might have to edit the .odbc.ini file in your home directory. Some ODBC driver vendors allow system administrators to maintain a centralized copy, by setting the environment variable ODBCINI. For specific configuration information, see your vendor documentation. The table services driver for ODBC uses shared libraries that are referenced as shared objects in UNIX. You must add the location of the shared libraries to one of the system environment variables, so that drivers for ODBC are loaded dynamically at run time. You must also set the ODBCHOME environment variable to your ODBC home directory before setting the environment variables, as shown in the following example.

```
export ODBCHOME=/dbi/odbc/dd7.1.4
 export ODBCINI=/ODBC/odbc_714_MASTER.ini
 LD_LIBRARY_PATH=/dbi/odbc/dd7.1.4/lib:${LD_LIBRARY_PATH}
 export LD_LIBRARY_PATH=${LD_LIBRARY_PATH%:}
```

Connection Options

Connection options are used to establish a connection to a data source. Specify one or more connection options when defining a data service. Here is an example:

```
driver=sql;conopts=(driver=odbc;
catalog=acat;conopts=(dsn=ODBCPgresDD;pwd=Tester2))
```

The driver for ODBC supports the following connection options.

Option	Description
CATALOG	**CATALOG=*catalog-identifier*;** Specifies an arbitrary identifier for an SQL catalog, which groups logically related schemas. For databases that do not support native catalogs, any identifier is valid (for example, catalog=myodbc). For databases like Microsoft SQL Server that do support native catalogs, CATALOG= is not required. The connection defaults to CATALOG=* unless you specify a logical name for the catalog and map it to the native catalog name in the database. For example, to map the logical catalog **mycat** to the native catalog named **newusers**, use the following command: **catalog=(mycat=newusers);**. Catalog name maps can be used only with FedSQL. They are not valid with native SQL. *Note:* The FedSQL language processor automatically quotes SQL identifiers that do not meet the regular naming convention as defined in *SAS FedSQL Reference Guide*.

CONOPTS	**CONOPTS=(*ODBC–compliant database connection string*);**
	Specifies an ODBC-compliant database connection string using ODBC-style syntax. These options, combined with the ODBC_DSN option, must specify a complete connection string to the data source. If you include a DSN= or FILEDSN= specification within the CONOPTS= option, do not use the ODBC_DSN= connection option. However, you can specify the ODBC database-specific connection options by using CONOPTS=. Then you can specify an ODBC DSN that contains other connection information by using the ODBC_DSN= connection option.
	Here is an example string using the CONOPTS option:
	`driver=sql;conopts=((driver=odbc;catalog=acat;`
	`conopts=(dsn=ODBCPgresDD;pwd=Tester2));`
	`(driver=postgres;catalog=bcat;uid=myuid;pwd='123pass';`
	`server=sv.abc.123.com;port=5432;DB=mydb;schema=public))`

DRIVER	**DRIVER=ODBC**; Calls the table services driver for ODBC. This specifies that the data service to which you want to connect must be an ODBC-compliant database. *Note:* DRIVER is a required option. You must specify the driver.
ODBC_DSN	**ODBC_DSN=*odbc dsn name*** Specifies a valid ODBC-compliant database DSN that contains connection information for connecting to the ODBC-compliant database. You can use the CONOPTS= option in addition to ODBC_DSN= option to specify database-specific connection options not provided by table services. Do not specify the ODBC DSN in both CONOPTS= and ODBC_DSN=.

Advanced Connection Options

The driver for ODBC supports the following advanced connection options for an ODBC-compliant database.

Option	Description
CT_PRESERVE	**CT_PRESERVE = STRICT | SAFE | FORCE | FORCE_COL_SIZE** Enables users to control how data types are mapped. Note that data type mapping is disabled when CT_PRESERVE is set to STRICT. If the requested type does not exist on the target database, an error is returned. Here are the options: • **STRICT** The requested type must exist in the target database. No type promotion occurs. If the type does not exist, an error is returned. • **SAFE** Target data types are upscaled only if they do not result in a loss of precision or scale. When character encodings are changed, the new column size is recalculated to ensure that all characters can be stored in the new encoding. • **FORCE** This is the default for all drivers. The best corresponding target data type is chosen, even if it could potentially result in a loss of precision or scale. When character encodings are changed, the new column size is recalculated to ensure that all characters can be stored in the new encoding. • **FORCE_COL_SIZE** This option is the same as FORCE, except that the column size for the new encoding is the same as the original encoding. This option can be used to avoid column size creep. However, the resulting column might be too large or too small for the target data.
ENABLE_MARS	**ENABLE_MARS= NO|YES** Enables or disables the use of multiple active result sets (MARS) on Microsoft SQL Server. FedSQL cannot permit transactions on top of Microsoft SQL Server because Microsoft SQL Server allows only one cursor per transaction. Set this option to YES so that FedSQL can allow transactions under a given Microsoft SQL Server connection.

Option	Description
DEFAULT_ATTR	**DEFAULT_ATTR=(*attr=value;...*)** Used to specify connection handle or statement handle attributes supported for initial connect-time configuration, where **attr=value** corresponds to any of the following options: • **CURSORS=*n*** - Connection handle option. This option controls the driver's use of client-side, result set cursors. The possible values are 0, 1, or 2. 0 Causes the driver to use client-side static cursor emulation if a scrollable cursor is requested but the database server cannot provide one. 1 Causes the driver to always use client-side static cursor emulation if a scrollable cursor is requested. The database server's native cursor is not used. 2 (Default) Causes the driver to never use client-side static cursor emulation if a scrollable cursor is requested. The database server's native cursor is used if available. Otherwise, the cursor is forward-only. Example: **DEFAULT_ATTR=(CURSORS=2)** • **USE_EVP=*n*** - Statement handle option. This option optimizes the driver for large result sets. The possible values are 0 (OFF) or 1 (ON), which is the default. Example: **DEFAULT_ATTR=(USE_EVP=0)** • **XCODE_WARN=*n*** - Statement handle option. Used to warn about possible character transcoding errors that occur during row input or output operations. Possible values are 0 (returns an error), 1 (returns a warning), or 2 (ignore transaction errors). 0 is the default. Example: **DEFAULT_ATTR=(XCODE_WARN=1)**

Option	Description
DEFAULT_CURSOR_TYPE	**DEFAULT_CURSOR_TYPE=FORWARD_ONLY | KEYSET_DRIVEN | DYNAMIC | STATIC;** Specifies a valid default cursor type for new statements. These options are valid: **FORWARD_ONLY** Specifies a non-scrollable cursor that moves only forward through the result set. Forward-only cursors are dynamic in that all changes are detected as the current row is processed. If an application does not require scrolling, the forward-only cursor retrieves data quickly, with the least amount of overhead processing. **KEYSET_DRIVEN** Specifies a scrollable cursor that detects changes that are made to the values of rows in the result set but that does not always detect changes to deletion of rows and changes to the order of rows in the result set. A keyset-driven cursor is based on row keys, which are used to determine the order and set of rows that are included in the result set. As the cursor scrolls the result set, it uses the keys to retrieve the most recent values in the table. It is sometimes helpful to have a cursor that can detect changes in the rows of a result set. A keyset-driven cursor uses a row identifier rather than caching the entire row into memory. It therefore uses much less disk space than other row caching mechanisms. Deleted rows can be detected when a SELECT statement that references the bookmark, row ID, or key column values fails to return a row. **DYNAMIC** Specifies a scrollable cursor that detects changes that are made to the rows in the result set. All INSERT, UPDATE, and DELETE statements that are made by all users are visible through the cursor. The dynamic cursor is good for an application that must detect all concurrent updates that are made by other users. **STATIC** Specifies a scrollable cursor that displays the result set as it existed when the cursor was first opened. The static cursor provides forward and backward scrolling. If the application does not need to detect changes but requires scrolling, the static cursor is a good choice. *Note:* The application can still override this value, but if the application does not explicitly set a cursor type, this value will be in effect

Option	Description
DRIVER_TRACE	**DRIVER_TRACE='API | SQL | ALL';** Requests tracing information, which logs transaction records to an external file that can be used for debugging purposes. The driver writes a record of each command that is sent to the database to the trace log based on the specified tracing level, which determines the type of tracing information. Here are the tracing levels: • **ALL** Activates all trace levels. • **API** Specifies that API method calls be sent to the trace log. This option is most useful if you are having a problem and need to send a trace log to SAS Technical Support for troubleshooting. • **DRIVER** Specifies that driver-specific information be sent to the trace log. • **SQL** Specifies that SQL statements that are sent to the database management system (DBMS) be sent to the trace log. Tracing information is DBMS specific, but most table services drivers log SQL statements such as SELECT and COMMIT. **Default:** Tracing is not activated. *Note:* If you activate tracing, you must also specify the location of the trace log with DRIVER_TRACEFILE=. Note that DRIVER_TRACEFILE= is resolved against the TRACEFILEPATH set in ALTER SERVER. TRACEFILEPATH is relative to the server's content root location. (Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=. **Interaction:** You can specify one trace level, or you can concatenate more than one by including the | (OR) symbol. For example, **driver_trace='api|sql'** generates tracing information for API calls and SQL statements.
DRIVER_TRACEFILE	**DRIVER_TRACEFILE='*filename*';** Used to specify the name of the text file for the trace log. Include the filename and extension in single or double quotation marks (for example, **driver_tracefile='\mytrace.log'**). **Default:** The default TRACEFILE location applies to a relative filename, and it is placed relative to TRACEFILEPATH. **Requirement:** DRIVER_TRACEFILE is required when activating tracing using DRIVER_TRACE. **Interaction:** (Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=.
DRIVER_TRACEOPTIONS	**DRIVER_TRACEOPTIONS=APPEND | THREADSTAMP | TIMESTAMP;** Specifies options in order to control formatting and other properties for the trace log: • **APPEND** Adds trace information to the end of an existing trace log. The contents of the file are not overwritten. • **THREADSTAMP** Prepends each line of the trace log with a thread identification. • **TIMESTAMP** Prepends each line of the trace log with a time stamp. **Default:** The trace log is overwritten with no thread identification or time stamp.
USER	**USER=*user-ID*;** Specifies the user ID for logging on to the ODBC-compliant database, such as Microsoft SQL Server, with a user ID that differs from the default ID. *Note:* The alias is **UID=**.

Option	Description
PASSWORD	**PASSWORD=*password*;** Specifies the password that corresponds to the user ID in the database. *Note:* The alias is **PWD=**.

Here are example connection strings that use the table services driver for ODBC:

```
driver=sql;conopts=((driver=odbc;catalog=acat;
conopts=(dsn=ODBCPgresDD;pwd=Tester2));
(driver=postgres;catalog=bcat;uid=myuid;pwd='123pass';
server=sv.abc.123.com;port=5432;DB=mydb;schema=public))
```

This connection string specifies catalog name maps to access multiple catalogs on Microsoft SQL Server:

```
driver=odbc; uid=jfox; pw=mypw; odbc_dsn=mySQLdsn;
    catalog=(cat1=mycat; cat2=testcat; cat3=users;
```

Wire Protocol Driver Usage Notes

Overview

There are a number of wire protocol ODBC drivers that communicate directly with a database server, without having to communicate through a client library. When you configure the ODBC drivers on Windows or UNIX, you can set certain options. SAS runs best when these options are selected. Some, but not all, are selected by default.

Windows	The options are located on the **Advanced** or **Performance** tabs in the ODBC Administrator window.
UNIX	The options are available when configuring data sources using the ODBC Administrator tool. Values can also be set by editing the **odbc.ini** file in which their data sources are defined.

Note: A DSN configuration that uses a wire protocol driver with the catalog option selected returns only the schemas that have associated tables or views. To list all existing schemas, create a DSN without selecting the catalog option.

SQL Server and SQL Server Legacy

Configure the following **Advanced** options for the SQL Server Wire Protocol driver and the SQL Server Legacy Wire Protocol driver:

- **Application Using Threads**
- **Enable Quoted Identifiers**
- **Fetch TWFS as Time**
- **Fetch TSWTZ as Timestamp**

Note:

1. Significant performance improvements have been realized when using the SQL Server Legacy Wire Protocol driver, as compared to the SQL Server Wire Protocol driver.

2. The SQL Server Legacy Wire Protocol driver does not support transactions when it is used with FedSQL enabled because the driver allows only a single statement per connection while FedSQL requires multiple statements per connection when using transactions.

Oracle Reference

Understanding the Table Services Driver for Oracle

The table services driver for Oracle enables table services to read and update legacy Oracle tables. In addition, the driver creates Oracle tables that can be accessed by both table services and Oracle.

The driver for Oracle supports most of the FedSQL functionality. The driver also enables an application to submit native Oracle SQL statements.

The driver for Oracle is a remote driver, which means that it connects to a server process in order to access data. The process might be running on the same machine as the table services, or it might be running on another machine in the network.

The table services driver for Oracle uses shared libraries that are referenced as shared objects in UNIX. You must add the location of the shared libraries to one of the system environment variables, and set any other environment variables required by the Oracle client libraries. The following Bourne shell commands provide an example:

```
ORAENV_ASK=NO; export ORAENV_ASK
ORACLE_HOME=/dbi/oracle/11g; export ORACLE_HOME
SASORA=V9; export SASORA
PATH=$ORACLE_HOME/bin:/bin:/usr/bin:/usr/ccs/
   bin:/opt/bin:$PATH; export PATH
TMPDIR=/var/tmp; export TMPDIR
LD_LIBRARY_PATH=/usr/openwin/lib:$ORACLE_HOME/
   lib:$LD_LIBRARY_PATH; export LD_LIBRARY_PATH
TWO_TASK=oraclev11; export TWO_TASK
```

Data Service Connection Options for Oracle

Overview

To access data that is hosted on the table services, a client must submit a connection string, which defines how to connect to the data. The data service connection arguments for an Oracle server include connection options and advanced options.

Connection Options

Connection options are used to establish a connection to a data source. Specify one or more connection options. Here is an example:

```
driver=sql;conopts=(driver=oracle;
catalog=acat;uid=myuid;pwd=myPass9;
path=oraclev11.abc.123.com:1521/ORA11G)
```

The driver for Oracle supports the following connection options.

Option	Description
CATALOG	**CATALOG=*catalog–identifier*;** Specifies an arbitrary identifier for an SQL catalog, which groups logically related schemas. Any identifier is valid such as **catalog=oracle_test**. You must specify a catalog. For the Oracle database, this is a logical catalog name to use as an SQL catalog identifier. *Note:* The FedSQL language processor automatically quotes SQL identifiers that do not meet the regular naming convention as defined in *SAS FedSQL Reference Guide*.
DRIVER	**DRIVER=ORACLE**; Identifies the data service to which you want to connect, which is an Oracle database. *Note:* You must specify the driver.
PATH	**PATH=*database-specification*;** Specifies the Oracle connect identifier. A connect identifier can be a net service name, a database service name, or a net service alias.
UID	**UID=*user-id*;** Specifies an optional Oracle user ID. If the user ID contains blanks or national characters, enclose it in quotation marks. If you omit an Oracle user ID and password, the default Oracle user ID **OPS$sysid** is used, if it is enabled.
PWD	**PWD=*password*;** Specifies an optional Oracle database password that is associated with the Oracle user ID. **PWD=** is always used with **UID=** and the associated password is case-sensitive. If you omit **PWD=**, the password for the default Oracle user ID **OPS$sysid** is used, if it is active.

Advanced Connection Options

The driver for Oracle supports the following advanced connection options.

Option	Description
CT_PRESERVE	**CT_PRESERVE = STRICT | SAFE | FORCE | FORCE_COL_SIZE** Enables users to control how data types are mapped. Note that data type mapping is disabled when CT_PRESERVE is set to STRICT. If the requested type does not exist on the target database, an error is returned. Here are the options: • **STRICT** The requested type must exist in the target database. No type promotion occurs. If the type does not exist, an error is returned. • **SAFE** Target data types are upscaled only if they do not result in a loss of precision or scale. When character encodings are changed, the new column size is recalculated to ensure all characters can be stored in the new encoding. • **FORCE** This is the default for all drivers. The best corresponding target data type is chosen, even if it could potentially result in a loss of precision or scale. When character encodings are changed, the new column size is recalculated to ensure all characters can be stored in the new encoding. • **FORCE_COL_SIZE** This option is the same as FORCE, except that the column size for the new encoding is the same as the original encoding. This option can be used to avoid column size creep. However, the resulting column might be too large or too small for the target data.

Option	Description
DEFAULT_ATTR	**DEFAULT_ATTR=(*attr=value;...*)** Used to specify connection handle or statement handle attributes that are supported for initial connect-time configuration, where **attr=value** corresponds to any of the following options: • **CURSORS=*n*** - Connection handle option. This option controls the driver's use of client-side, result set cursors. The possible values are 0, 1, or 2. 0 Causes the driver to use client-side static cursor emulation if a scrollable cursor is requested but the database server cannot provide one. 1 Causes the driver to always use client-side static cursor emulation if a scrollable cursor is requested. The database server's native cursor is not used. 2 (Default) Causes the driver to never use client-side static cursor emulation if a scrollable cursor is requested. The database server's native cursor is used if available. Otherwise, the cursor is forward-only. Example: **DEFAULT_ATTR=(CURSORS=2)** • **USE_EVP=*n*** - Statement handle option. This option optimizes the driver for large result sets. The possible values are 0 (OFF) or 1 (ON), which is the default. Example: **DEFAULT_ATTR=(USE_EVP=0)** • **XCODE_WARN=*n*** - Statement handle option. Used to warn about possible character transcoding errors that occur during row input or output operations. Possible values are 0 (returns an error), 1 (returns a warning), or 2 (ignore transaction errors). 0 is the default. Example: **DEFAULT_ATTR=(XCODE_WARN=1)**
DRIVER_TRACE	**DRIVER_TRACE='API | SQL | ALL'** Requests tracing information, which logs transaction records to an external file that can be used for debugging purposes. The driver writes a record of each command that is sent to the database to the trace log based on the specified tracing level, which determines the type of tracing information. Here are the tracing levels: • **ALL** Activates all trace levels. • **API** Specifies that API method calls be sent to the trace log. This option is most useful if you are having a problem and need to send a trace log to SAS Technical Support for troubleshooting. • **DRIVER** Specifies that driver-specific information be sent to the trace log. • **SQL** Specifies that SQL statements that are sent to the database management system (DBMS) be sent to the trace log. Tracing information is DBMS specific, but most table services drivers log SQL statements such as SELECT and COMMIT. **Default:** Tracing is not activated. *Note:* If you activate tracing, you must also specify the location of the trace log with DRIVER_TRACEFILE=. Note that DRIVER_TRACEFILE= is resolved against the TRACEFILEPATH set in ALTER SERVER. TRACEFILEPATH is relative to the server's content root location. (Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=. **Interaction:** You can specify one trace level, or you can concatenate more than one by including the | (OR) symbol. For example, **driver_trace='api|sql'** generates tracing information for API calls and SQL statements.

Option	Description
DRIVER_TRACEFILE	**DRIVER_TRACEFILE='*filename*';** Used to specify the name of the text file for the trace log. Include the filename and extension in single or double quotation marks (for example, **driver_tracefile='\mytrace.log'**). **Default:** The default TRACEFILE location applies to a relative filename, and it is placed relative to TRACEFILEPATH. **Requirement:** DRIVER_TRACEFILE is required when activating tracing using DRIVER_TRACE. **Interaction:** (Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=.
DRIVER_TRACEOPTIONS	**DRIVER_TRACEOPTIONS=APPEND | THREADSTAMP | TIMESTAMP;** Specifies options in order to control formatting and other properties for the trace log: • **APPEND** Adds trace information to the end of an existing trace log. The contents of the file are not overwritten. • **THREADSTAMP** Prepends each line of the trace log with a thread identification. • **TIMESTAMP** Prepends each line of the trace log with a time stamp. **Default:** The trace log is overwritten with no thread identification or time stamp.
ORA_ENCODING	**ORA_ENCODING=UNICODE;** Specifies that the Oracle data be returned in Unicode to table services. **UNICODE** is the default setting and is independent of the **NLS_LANG** environment variable setting.
ORNUMERIC	**ORANUMERIC=NO | YES** Specifies how numbers that are read from or inserted into the Oracle NUMBER column are treated. This option defaults to YES so that a NUMBER column with precision or scale is described as **TKTS_NUMERIC**. This option can be specified as both a connection option and a table option. When specified as both a connection and table option, the table option value overrides the connection option. • **NO** Indicates that the numbers are treated as **TKTS_DOUBLE** values. They might not have precision beyond 14 digits. • **YES** Indicates that non-integer values with explicit precision are treated as TKTS_NUMERIC values. This is the default setting.
USE_CACHED_CATALOG	**USE_CACHED_CATALOG=YES | NO;** Specifies whether to use the cached catalog rather than compiling a new catalog on every run. Setting this option to YES can improve the performance of the TKTSForeignKeys API. The default setting is YES. *Note:* Before you can use this option, you must complete the following steps: 1. Create a materialized view. See the example code in "Creating a Materialized View (USE_CACHED_CATALOG)" on page 172. 2. Use the ALTER DSN statement to add the USE_CACHED_CATALOG connection option.

Creating a Materialized View (USE_CACHED_CATALOG)

The following example shows you how to create a materialized view. Use this script if **USE_CACHED_CATALOG** is set to YES above.

```
/*----------------------SAS_CACHED_CATALOG.SQL-------------------------------*/
/* This script is used to create the materialized and the synonym needed to
   get the ForeignKey metadata.  Work with your DBA to set this up.
   Materialized views can be complex and so thorough understanding will help us
   use them effectively. Especially deciding how to do the refreshes.
   Here we provide the simplest possible steps to create the required materialized
   view and the command to refresh it manually. The materialized view below can
   be created in any schema with any name. Feel free to add whatever REFRESH
   options suits your purpose. Note that you might need additional steps based
   on the REFRESH option setting. Here we provide the simplest possible way to
   do this.  The PUBLIC synonym pointing to this Materialized view must be
   named "SAS_CACHED_FK_CATALOG_PSYN". This synonym must be visible to
   PUBLIC (or the set of users who will be needing Foreignkey metadata) so that
   it is accessible from any   schema.
*/

Create materialized view SAS_CACHED_FK_CATALOG_MATVIEW REFRESH ON DEMAND as SELECT
PKAC.OWNER as PKTABLE_SCHEM,
PKAC.TABLE_NAME as PKTABLE_NAME,
PKACC.COLUMN_NAME as PKCOLUMN_NAME,
FKAC.OWNER as FKTABLE_SCHEM,
FKAC.TABLE_NAME as FKTABLE_NAME,
FKACC.COLUMN_NAME as FKCOLUMN_NAME,
FKACC.POSITION as KEY_SEQ,
FKAC.CONSTRAINT_NAME as FK_NAME,
PKAC.CONSTRAINT_NAME as PK_NAME
from
sys.all_constraints PKAC, sys.all_constraints FKAC,
sys.all_cons_columns PKACC, sys.all_cons_columns FKACC

where

FKAC.r_constraint_name=PKAC.constraint_name and
FKAC.constraint_name=FKACC.constraint_name and
PKAC.constraint_name=PKACC.constraint_name and PKAC.constraint_type='P' and
FKAC.constraint_type='R' and FKAC.owner=FKACC.owner and PKAC.owner=PKACC.owner
and PKAC.table_name=PKACC.table_name and FKAC.table_name=FKACC.table_name and
FKACC.position = PKACC.position ;

/* The synonym name *must* be SAS_CACHED_FK_CATALOG_PUBLIC_SYNONYM */
create public synonym  SAS_CACHED_FK_CATALOG_PSYN for SAS_CACHED_FK_CATALOG_MATVIEW;
grant all on  SAS_CACHED_FK_CATALOG_PSYN to PUBLIC;

/*---------Manual REFRESH of the Materialized View----------------------------*/
/* Note there are several ways to do this, consult with your DBA.
   Here are a couple of ways:
*/
execute DBMS_MVIEW.REFRESH('SAS_CACHED_FK_CATALOG_MATVIEW');
execute DBMS_SNAPSHOT.REFRESH('SAS_CACHED_FK_CATALOG_MATVIEW', '?');
```

Oracle Wire Protocol Driver Usage Notes

Wire protocol ODBC drivers communicate directly with a database server without having to communicate through a client library. When you configure the ODBC drivers on Windows or UNIX, you can set certain options. SAS runs best when these options are selected. Some, but not all, are selected by default.

Windows	The options are located on the **Advanced** or **Performance** tabs in the ODBC Administrator.
UNIX	The options are available when you are configuring data sources using the ODBC Administrator tool. Values can also be set by editing the **`odbc.ini`** file in which their data sources are defined.

Note: When you use a wire protocol driver to create an ODBC connection, the following special considerations apply:

1. A DSN configuration that uses a wire protocol driver with the catalog option selected returns only the schemas that have associated tables or views. To list all existing schemas, create a DSN without selecting the catalog option.
2. Verify that the Enable Bulk Load option is active in the ODBC DSN for databases that support this option. The Enable Bulk Load option is not enabled by default in the newer wire protocol drivers. As a result, insert performance suffers.

When configuring an ODBC DSN using the Oracle Wire Protocol driver, set the following advanced options:

- **Application Using Threads**
- **Enable SQLDescribeParam**
- **Describe at Prepare**
- **Enable N-CHAR Support**
- **Enable Scrollable Cursors**

PostgreSQL Driver Reference

Understanding the SAS Federation Server Driver for PostgreSQL

The table services driver for PostgreSQL enables table services to read and update legacy PostgreSQL tables. In addition, the driver creates PostgreSQL tables that can be accessed by both the table services and the PostgreSQL data management system.

The driver for PostgreSQL supports most of the FedSQL functionality. The driver also enables an application to submit native SQL statements.

The driver for PostgreSQL is a remote driver, which means that it connects to a server process in order to access data. The process might be running on the same machine as the table services, or it might be running on another machine in the network.

The table services driver for PostgreSQL uses shared libraries that are referenced as shared objects in UNIX. You must add the location of the shared libraries to one of the system environment variables, and set any other environment variables required by the PostgreSQL client libraries. The following Korn shell commands provide an example:

```
LD_LIBRARY_PATH=/dbi/odbc/unixodbc2310/lib:/dbi/
   postgres/9.03.04/lib:${LD_LIBRARY_PATH}
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH%:}
export ODBCSYSINI=/dbi/postgres/9.03.04
export PATH=/dbi/postgres/9.03.04/bin:$PATH
unset LANG
export PGCLIENTENCODING=UTF8
```

Data Service Connection Options for PostgreSQL

Overview

To access data that is hosted on the table services, a client must submit a connection string, which defines how to connect to the data. The data service connection arguments for PostgreSQL include connection options and advanced options.

Connection Options

Connection options are used to establish a connection to a data source. Specify one or more connection options when defining a data service. Here is an example:

```
driver=sql;conopts=(driver=postgres;catalog=acat;
uid=myuid;pwd='123pass';server=sv.abc.123.com;
port=5432;DB=mydb;schema=public)
```

The following connection options are supported for PostgreSQL data sources.

Option	Description
CATALOG	**CATALOG=*catalog-identifier*;** Specifies an arbitrary identifier for an SQL catalog, which groups schemas that are logically related (for example, **catalog=ptgtest**). *Note:* The FedSQL language processor automatically quotes SQL identifiers that do not meet the regular naming convention as defined in *SAS FedSQL Reference Guide*.

CONOPTS	**CONOPTS=(*ODBC–compliant database connection string*);**
	Specifies an ODBC-compliant database connection string using ODBC-style syntax. These options, combined with the ODBC_DSN option, must specify a complete connection string to the data source. If you include a DSN= or FILEDSN= specification within the CONOPTS= option, do not use the ODBC_DSN= connection option. However, you can specify the ODBC database-specific connection options by using CONOPTS=. Then you can specify an ODBC DSN that contains other connection information by using the ODBC_DSN= connection option.
	Here is an example string using the CONOPTS option:
	`driver=sql;conopts=`
	`((driver=odbc;catalog=acat;conopts=(dsn=ODBCPgresDD;pwd=Tester2));`
	`(driver=postgres;catalog=bcat;uid=myuid2;pwd='123mypass';`
	`server=sv.abc.123.com;port=5432;DB=mydb;schema=public))"`

Option	Description
DRIVER	**DRIVER=*postgres***; Specifies the data service for the PostgreSQL database to which you want to connect. *Note:* DRIVER is a required option. You must specify a driver.
DATABASE	**DATABASE=*database-name*;** Specifies the name of the PostgreSQL database. Enclose the database name in single quotation marks if it contains spaces or non-alphanumeric characters. You can also specify DATABASE= with the DB= alias.**database=sample, DB=sample**.
DSN	**DSN=*data-source-identifier*;** Specifies the data source name to which you want to connect.
PWD	**PWD=*password*;** Specifies the password associated with the user ID. Enclose password in single quotation marks if it contains spaces or non-alphanumeric characters. You can also specify PASSWORD= with the PWD=, PASS=, and PW= aliases.
PORT	**PORT=*port_number*** Specifies the port number that is used to connect to the specified PostgreSQL Server. If you do not specify a port, the default is 5432.
SERVER	**SERVER='*server-name*'** Specifies the server name or IP address of the PostgreSQL server to which you want to connect. Enclose the server name in single quotation marks if the name contains spaces or non-alphanumeric characters: **SERVER='*server name*'**.
USER	**USER=*user-name*** Specifies the PostgreSQL user name (also called the user ID) that you use to connect to your database. If the user name contains spaces or non-alphanumeric characters, you must enclose it in quotation marks.

Advanced Options

The following advanced options are supported for PostgreSQL data sources.

Option	Description
ALLOW_UNQUOTED_NAMES	**ALLOW_UNQUOTED_NAMES=NO | YES** Specifies whether to enclose table and column names in quotation marks. Tables and columns are quoted when this option is set at NO. If set to YES, the driver does not automatically add quotation marks to table and column names if they are not specified. This allows PostgreSQL tables and columns to be created in the default lowercase. The default option is NO.
CLIENT_ENCODING	**CLIENT_ENCODING=*cei*** Used to specify encoding for the client.

Option	Description
CT_PRESERVE	**CT_PRESERVE=STRICT | SAFE | FORCE | FORCE_COL_SIZE** Enables users to control how data types are mapped. Note that data type mapping is disabled when CT_PRESERVE is set to STRICT. If the requested type does not exist on the target database, an error is returned. Here are the options: • **STRICT** The requested type must exist in the target database. No type promotion occurs. If the type does not exist, an error is returned. • **SAFE** Target data types are upscaled only if they do not result in a loss of precision or scale. When character encodings are changed, the new column size is recalculated to ensure all characters can be stored in the new encoding. • **FORCE** This is the default for all drivers. The best corresponding target data type is chosen, even if it could potentially result in a loss of precision or scale. When character encodings are changed, the new column size is recalculated to ensure all characters can be stored in the new encoding. • **FORCE_COL_SIZE** This option is the same as FORCE, except that the column size for the new encoding is the same as the original encoding. This option can be used to avoid column size creep. However, the resulting column might be too large or too small for the target data.
DEFAULT_ATTR	**DEFAULT_ATTR=(*attr=value*;...)** Used to specify connection handle or statement handle attributes supported for initial connect-time configuration, where **attr=value** corresponds to any of the following options: • **CURSORS=*n*-** Connection handle option. This option controls the driver's use of client-side, result set cursors. The possible values are 0, 1, or 2. 0 Causes the driver to use client-side static cursor emulation if a scrollable cursor is requested but the database server cannot provide one. 1 Causes the driver to always use client-side static cursor emulation if a scrollable cursor is requested. The database server's native cursor is not used. 2 (Default) Causes the driver to never use client-side static cursor emulation if a scrollable cursor is requested. The database server's native cursor is used if available. Otherwise, the cursor is forward-only. Example: **DEFAULT_ATTR=(CURSORS=2)** • **USE_EVP=*n*** - Statement handle option. This option optimizes the driver for large result sets. The possible values are 0 (OFF) or 1 (ON), which is the default. Example: **DEFAULT_ATTR=(USE_EVP=0)** • **XCODE_WARN=*n*** - Statement handle option. Used to warn about possible character transcoding errors that occur during row input or output operations. Possible values are 0 (returns an error), 1 (returns a warning), or 2 (ignore transaction errors). 0 is the default. Example: **DEFAULT_ATTR=(XCODE_WARN=1)**

Option	Description
DRIVER_TRACE	**DRIVER_TRACE='API | SQL | ALL';** Requests tracing information, which logs transaction records to an external file that can be used for debugging purposes. The driver writes a record of each command that is sent to the database to the trace log based on the specified tracing level, which determines the type of tracing information. Here are the tracing levels: • **ALL** Activates all trace levels. • **API** Specifies that API method calls be sent to the trace log. This option is most useful if you are having a problem and need to send a trace log to SAS Technical Support for troubleshooting. • **DRIVER** Specifies that driver-specific information be sent to the trace log. • **SQL** Specifies that SQL statements that are sent to the database management system (DBMS) be sent to the trace log. Tracing information is DBMS specific, but most table services drivers log SQL statements such as SELECT and COMMIT. **Default:** Tracing is not activated. *Note:* If you activate tracing, you must also specify the location of the trace log with DRIVER_TRACEFILE=. Note that DRIVER_TRACEFILE= is resolved against the TRACEFILEPATH set in ALTER SERVER. TRACEFILEPATH is relative to the server's content root location. (Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=. **Interaction:** You can specify one trace level, or you can concatenate more than one by including the | (OR) symbol. For example, **driver_trace='api|sql'** generates tracing information for API calls and SQL statements.
DRIVER_TRACEFILE	**DRIVER_TRACEFILE='*filename*'**; Used to specify the name of the text file for the trace log. Include the filename and extension in single or double quotation marks (for example, **driver_tracefile='\mytrace.log'**). **Default:** The default TRACEFILE location applies to a relative filename, and it is placed relative to TRACEFILEPATH. **Requirement:** DRIVER_TRACEFILE is required when activating tracing using DRIVER_TRACE. **Interaction:** (Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=.
DRIVER_TRACEOPTIONS	**DRIVER_TRACEOPTIONS=APPEND | THREADSTAMP | TIMESTAMP;** Specifies options in order to control formatting and other properties for the trace log: • **APPEND** Adds trace information to the end of an existing trace log. The contents of the file are not overwritten. • **THREADSTAMP** Prepends each line of the trace log with a thread identification. • **TIMESTAMP** Prepends each line of the trace log with a time stamp. **Default:** The trace log is overwritten with no thread identification or time stamp.
MAX_BINARY_LEN	**MAX_BINARY_LEN=*value;*** Specifies a value, in bytes, that limits the length of long binary fields (LONG VARBINARY). Unlike other databases, PostgreSQL does not have a size limit for long binary fields. The default is 1048576.

Option	Description
MAX_CHAR_LEN	**MAX_CHAR_LEN=*value*;** Specifies a value that limits the length of character fields (CHAR and VARCHAR). The default is 2000.
MAX_TEXT_LEN	**MAX_TEXT_LEN=*value*;** Specifies a value that limits the length of long character fields (LONG VARCHAR). The default is 409500.
SCHEMA	**SCHEMA=*value*;** Specifies the default schema for the connection. If not specified, the schema, or list of schemas, is determined based on the value of the schema search path that is defined on the database server.
STRIP_BLANKS	**STRIP_BLANKS=YES|NO;** Specifies whether to strip blanks from character fields.

SAS Data Set Reference

Overview

The SAS data set is a SASProprietary file format, which contains data values that are organized as a table of rows (SAS observations) and columns (SAS variables). A supported SAS data set uses the extension **.sas7bdat**.

Understanding the Driver for Base SAS

The table services driver for Base SAS is a SASProprietary driver that provides Read and Update access to legacy SAS data sets. With the table services driver for Base, you can create SAS data sets that can be accessed by both the legacy and the table services data access services.

The driver supports much of the Base SAS functionality, such as SAS indexing and general integrity constraints, as well as much of the Federated Query Language (FedSQL) functionality.

The table services driver for Base SAS is an in-process driver, which means that it accesses data in the same process that executes the data access services. All server connections that are made with the table services driver for Base SAS use LOCKTABLE=SHARED and PATH_BIND=ACCESS connection options.

Data Service Connection Options for SAS Data Sets

Connection Options

To access data that is hosted on the table services, a client must submit a connection string, which defines how to connect to the data. The data service connection arguments for a SAS data set include connection options and advanced options. Here is an example:

```
driver=sql;conopts=(driver=base;catalog=acat;
schema=(name=dblib;primarypath=/u/path/mydir))
```

The following connection options are supported for SAS data sets:

Option	Description
CATALOG	**CATALOG=*catalog-identifier*;** Specifies an arbitrary identifier for an SQL catalog, which groups logically related schemas. A catalog name can be up to 32 characters long. You must specify a catalog. *Note:* The FedSQL language processor automatically quotes SQL identifiers that do not meet the regular naming convention as defined in *SAS FedSQL Reference Guide*.
DRIVER	**DRIVER=BASE;** Identifies the data service to which you want to connect, which is a SAS data set. *Note:* You must specify DRIVER=BASE to access a SAS data set.
(SCHEMA) NAME	**NAME=*schema-identifier*;** Specifies an arbitrary identifier for an SQL schema. Any identifier is valid (for example, name=myfiles). The schema identifier is an alias for the physical location of the SAS library, which is much like the Base SAS libref. A schema name must be a valid SAS name and can be up to 32 characters long. You must specify a schema identifier.
PRIMARY PATH	**PRIMARYPATH=*physical-location*;** Specifies the physical location for the SAS library, which is a collection of one or more SAS files. For example, in directory-based operating environments, a SAS library is a group of SAS files that are stored in the same directory. *Note:* You must specify a primary path.
SCHEMA (ATTRIBUTES)	**SCHEMA=(*attributes*);** Specifies schema attributes that are specific to a SAS data set. A schema is a data container object that groups tables. The schema contains a name, which is unique within the catalog that qualifies table names. For a SAS data set, a schema is similar to a SAS library, which is a collection of tables with assigned attributes.

Advanced Options

Advanced driver options are additional options that are not required in order to connect to the data source. They are used to establish connections to catalogs, data source names (DSNs), and schemas. Although advanced options can also be used when connecting to a data service, doing so causes the specified options to apply to all data service connections.

The following advanced options are supported for SAS data sets:

Option	Description
ACCESS	**ACCESS=READONLY | TEMP;** • **READONLY** Assigns a read-only attribute to the schema. You cannot open a SAS data set to update or write new information. • **TEMP** specifies that the SAS data sets be treated as scratch files. That is, the system will not consume CPU cycles to ensure that the files do not become corrupted. ***TIP*** Use ACCESS=TEMP to save resources only when the data is recoverable. If TEMP is specified, data in memory might not be written to disk on a regular basis. This saves I/O, but could cause a loss of data if there is a crash.
CT_PRESERVE	**CT_PRESERVE = STRICT | SAFE | FORCE | FORCE_COL_SIZE** Enables users to control how data types are mapped. Note that data type mapping is disabled when CT_PRESERVE is set to STRICT. If the requested type does not exist on the target database, an error is returned. Here are the options: • **STRICT** The requested type must exist in the target database. No type promotion occurs. If the type does not exist, an error is returned. • **SAFE** Target data types are upscaled only if they do not result in a loss of precision or scale. When character encodings are changed, the new column size is recalculated to ensure all characters can be stored in the new encoding. • **FORCE** This is the default for all drivers. The best corresponding target data type is chosen, even if it could potentially result in a loss of precision or scale. When character encodings are changed, the new column size is recalculated to ensure that all characters can be stored in the new encoding. • **FORCE_COL_SIZE** This option is the same as FORCE, except that the column size for the new encoding is the same as the original encoding. This option can be used to avoid column size creep. However, the resulting column might be too large or too small for the target data.
COMPRESS	**COMPRESS=NO | YES | CHAR | BINARY;** Controls the compression of rows in created SAS data sets. • **NO** Specifies that the rows in a newly created SAS data set are uncompressed (fixed-length records). This setting is the default. • **YES | CHAR** Specifies that the rows in a newly created SAS data set are compressed (variable-length records) by using RLE (Run Length Encoding). RLE compresses rows by reducing repeated consecutive characters (including blanks) to two- or three-byte representations. ***TIP*** Use this compression algorithm for character data. • **BINARY** Specifies that the rows in a newly created SAS data set are compressed (variable-length records) by using RDC (Ross Data Compression). RDC combines run-length encoding and sliding-window compression to compress the file. ***TIP*** This method is highly effective for compressing medium to large (several hundred bytes or larger) blocks of binary data (numeric columns). Because the compression function operates on a single record at a time, the record length must be several hundred bytes or larger for effective compression.

Option	Description
DEFAULT_ATTR	**DEFAULT_ATTR=(*attr=value;...*)** Used to specify connection handle or statement handle attributes that are supported for initial connect-time configuration, where **attr=value** corresponds to any of the following options: • **CURSORS=*n*** - Connection handle option. This option controls the driver's use of client-side, result set cursors. The possible values are 0, 1, or 2. 0 Causes the driver to use client-side static cursor emulation if a scrollable cursor is requested but the database server cannot provide one. 1 Causes the driver to always use client-side static cursor emulation if a scrollable cursor is requested. The database server's native cursor is not used. 2 (Default) Causes the driver to never use client-side static cursor emulation if a scrollable cursor is requested. The database server's native cursor is used if available. Otherwise, the cursor is forward-only. Example: **DEFAULT_ATTR=(CURSORS=2)** • **USE_EVP=*n*** - Statement handle option. This option optimizes the driver for large result sets. The possible values are 0 (OFF) or 1 (ON), which is the default. Example: **DEFAULT_ATTR=(USE_EVP=0)** • **XCODE_WARN=*n*** - Statement handle option. Used to warn about possible character transcoding errors that occur during row input or output operations. Possible values are 0 (returns an error), 1 (returns a warning), or 2 (ignore transaction errors). 0 is the default. Example: **DEFAULT_ATTR=(XCODE_WARN=1)**
ENCODING	**ENCODING=*encoding-value;*** Overrides and transcodes the encoding for input or output processing of SAS data sets. *Note:* The default value is the current operating system setting.
LOCKTABLE	**LOCKTABLE=SHARED | EXCLUSIVE** Places exclusive or shared locks on SAS data sets. You can lock tables only if you are the owner or have been granted the necessary privilege. The default value for the table services is SHARED. • **SHARED** Locks tables in shared mode, allowing other users or processes to read data from the tables, but preventing other users from updating. • **EXCLUSIVE** Locks tables exclusively, preventing other users from accessing any table that you open.
PATH_BIND	**PATH_BIND=CONNECT | ACCESS** Specifies when and how schemas are validated during connection. CONNECT validates the entire connection string at the time of connection and returns an error if one or more schemas is invalid. ACCESS validates schemas when they are accessed so that processing continues regardless of errors in the schema portion of the connection string. ACCESS is the default for the table services.

Teradata Reference

Understanding the Table Services Driver for Teradata

The table services driver for Teradata provides Read and Update access to Teradata database tables and creates tables that can be accessed by both table services and Teradata.

The table services driver for Teradata supports most of the FedSQL functionality. The driver also enables an application to submit native Teradata SQL statements.

The table services driver for Teradata is a remote driver, which means that it connects to a server process to access data. The process might be running on the same machine as the table services, or it might be running on another machine in the network.

The table services driver for uses shared libraries that are referenced as shared objects in UNIX. You must add the location of the shared libraries to one of the system environment variables, and set any other environment variables that are required by the Teradata client libraries. The following Korn shell commands provide an example:

```
LD_LIBRARY_PATH=/opt/teradata/client/14.10/
   lib64:/opt/teradata/client/14.10/tbuild/lib64:/
   opt/teradata/client/14.10/tdicu/lib64:${LD_LIBRARY_PATH}
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH%:}
export COPERR=/opt/teradata/client/14.10/lib
export COPLIB=/opt/teradata/client/14.10/lib
export NLSPATH=/opt/teradata/client/14.10/tbuild/msg64/%N
```

Data Service Connection Options for Teradata

Connection Options

Connection options are used to establish a connection to a data source. Specify one or more connection options when defining a data service. Here is an example:

```
driver=sql;conopts=(driver=teradata;catalog=acat;
uid=myuid;pwd='{sas002}C5DDFFF91B5D31DFFFCE9FFF';
server=terasoar;database=model)
```

The following connection options are supported for a Teradata database.

Option	Description
CATALOG	**CATALOG=*catalog-identifier;*** Specifies an arbitrary identifier for an SQL catalog, which groups logically related schemas. Any identifier is valid (for example, catalog=tera). *Note:* You must specify a catalog.
DATABASE	**DATABASE=*database-name;*** Specifies the Teradata database. If you do not specify DATABASE=, you connect to the default Teradata database, which is often named the same as your user ID. If the database value that you specify contains spaces or non-alphanumeric characters, you must enclose it in quotation marks.

Option	Description
DRIVER	**DRIVER=TERA;** Identifies the data service to which you want to connect, which is a Teradata database. *Note:* You must specify the driver.
SERVER	**SERVER=*server-name*;** Specifies the Teradata server identifier.

Advanced Connection Options

The following advanced options are supported for Teradata database.

Option	Description
ACCOUNT	**ACCOUNT=*account-ID*;** Specifies an optional account number that you want to charge for the Teradata session.
CLIENT_ENCODING	**CLIENT_ENCODING=*encoding-value*** Used to specify the character set for the session. UTF8 is the default if encoding is not specified. These character sets are supported: ASCII EBCDIC EBCDIC037_0E KATAKANAEBCDIC KANJIEUC_0U LATIN9_0A THAI874_4A0 LATIN1250_1A0 CYRILLIC1251_2A0 LATIN1254_7A0 HEBREW1255_5A0 ARABIC1256_6A0 LATIN1258_8A0 TCHBIG5_1R0 SCHINESE936_6R0 KANJI932_1S0 HANGUL949_7R0 TCHINESE950_8R0 LATIN1252_3A0 SCHEBCDIC935_2IJ TCHEBCDIC937_3IB HANGULEBCDIC933_1II EBCDIC273_0E EBCDIC277_0E KANJIEBCDIC5035_0I KANJIEBCDIC5026_0I UTF8 UTF16

Option	Description
CT_PRESERVE	**CT_PRESERVE = STRICT | SAFE | FORCE | FORCE_COL_SIZE** Enables users to control how data types are mapped. Note that data type mapping is disabled when CT_PRESERVE is set to STRICT. If the requested type does not exist on the target database, an error is returned. Here are the options: • **STRICT** The requested type must exist in the target database. No type promotion occurs. If the type does not exist, an error is returned. • **SAFE** Target data types are upscaled only if they do not result in a loss of precision or scale. When character encodings are changed, the new column size is recalculated to ensure all characters can be stored in the new encoding. • **FORCE** This is the default for all drivers. The best corresponding target data type is chosen, even if it could potentially result in a loss of precision or scale. When character encodings are changed, the new column size is recalculated to ensure that all characters can be stored in the new encoding. • **FORCE_COL_SIZE** This option is the same as FORCE, except that the column size for the new encoding is the same as the original encoding. This option can be used to avoid column size creep. However, the resulting column might be too large or too small for the target data.
DEFAULT_ATTR	**DEFAULT_ATTR=(*attr=value;...*)** Used to specify connection handle or statement handle attributes supported for initial connect-time configuration, where **attr=value** corresponds to any of the following options: • **CURSORS=*n*** - Connection handle option. This option controls the driver's use of client-side, result set cursors. The possible values are 0, 1, or 2. 0 Causes the driver to use client-side static cursor emulation if a scrollable cursor is requested but the database server cannot provide one. 1 Causes the driver to always use client-side static cursor emulation if a scrollable cursor is requested. The database server's native cursor is not used. 2 (Default) Causes the driver to never use client-side static cursor emulation if a scrollable cursor is requested. The database server's native cursor is used if available. Otherwise, the cursor is forward-only. Example: **DEFAULT_ATTR=(CURSORS=2)** • **USE_EVP=*n*** - Statement handle option. This option optimizes the driver for large result sets. The possible values are 0 (OFF) or 1 (ON), which is the default. Example: **DEFAULT_ATTR=(USE_EVP=0)** • **XCODE_WARN=*n*** - Statement handle option. Used to warn about possible character transcoding errors that occur during row input or output operations. Possible values are 0 (returns an error), 1 (returns a warning), or 2 (ignore transaction errors). 0 is the default. Example: **DEFAULT_ATTR=(XCODE_WARN=1)**

Option	Description
DRIVER_TRACE	**DRIVER_TRACE='API | SQL | ALL';** Requests tracing information, which logs transaction records to an external file that can be used for debugging purposes. The driver writes a record of each command that is sent to the trace log based on the specified tracing level, which determines the type of tracing information. Here are the tracing levels: • **ALL** Activates all trace levels. • **API** Specifies that API method calls be sent to the trace log. This option is most useful if you are having a problem and need to send a trace log to SAS Technical Support for troubleshooting. • **DRIVER** Specifies that driver-specific information be sent to the trace log. • **SQL** Specifies that SQL statements that are sent to the database management system (DBMS) be sent to the trace log. Tracing information is DBMS specific, but most table services drivers log SQL statements such as SELECT and COMMIT. **Default:** Tracing is not activated. *Note:* If you activate tracing, you must also specify the location of the trace log with DRIVER_TRACEFILE=. Note that DRIVER_TRACEFILE= is resolved against the TRACEFILEPATH set in ALTER SERVER. TRACEFILEPATH is relative to the server's content root location. (Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=. **Interaction:** You can specify one trace level, or you can concatenate more than one by including the | (OR) symbol. For example, **driver_trace='api|sql'** generates tracing information for API calls and SQL statements.
DRIVER_TRACEFILE	**DRIVER_TRACEFILE='*filename*';** Used to specify the name of the text file for the trace log. Include the filename and extension in single or double quotation marks (for example, **driver_tracefile='\mytrace.log'**). **Default:** The default TRACEFILE location applies to a relative filename, and it is placed relative to TRACEFILEPATH. **Requirement:** DRIVER_TRACEFILE is required when activating tracing using DRIVER_TRACE. **Interaction:** (Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=.
DRIVER_TRACEOPTIONS	**DRIVER_TRACEOPTIONS=APPEND | THREADSTAMP | TIMESTAMP;** Specifies options in order to control formatting and other properties for the trace log: • **APPEND** Adds trace information to the end of an existing trace log. The contents of the file are not overwritten. • **THREADSTAMP** Prepends each line of the trace log with a thread identification. • **TIMESTAMP** Prepends each line of the trace log with a time stamp. **Default:** The trace log is overwritten with no thread identification or time stamp.
PASSWORD	**PASSWORD=*password*;** Specifies a Teradata password. The password must match your USER= value. The alias is PWD=. *Note:* You must specify the PASSWORD= option.

Option	Description
ROLE	**ROLE=*security-role*;** Specifies a security role for the session.
USER	**USER=*user-id*;** Specifies a Teradata user ID. If the ID contains blanks or national characters, enclose it in quotation marks. The alias is UID=. *Note:* You must specify the USER= option.

Recommended Reading

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

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

Numbers
32-bit wire protocol driver 173

A
administration logging 39
argument types 13

B
backup disk stores 54
business context 4

C
character-to-numeric conversions 25
configuring Python 32

D
deploying 46
DS2 best practices 21, 22, 25, 26
DS2 programming 7, 8, 10, 11, 13

G
global packages 21
Greenplum
  64-bit wire protocol driver 156

H
hash package 25

I
interfaces 5
invariant computations 26

L
local packages 21

M
module 4

O
ODBC 161
  SQL server legacy wire protocol driver 167
Oracle
  32-bit wire protocol driver 173
  64-bit wire protocol driver 173

P
passing character values to methods 25
pre-installation 45
private methods 11
private packages 11
programming blocks 10
public methods 11
public packages 11
publishing DS2 source code 7
Python 27, 32
Python support 27

R
REST API 62
revision 4

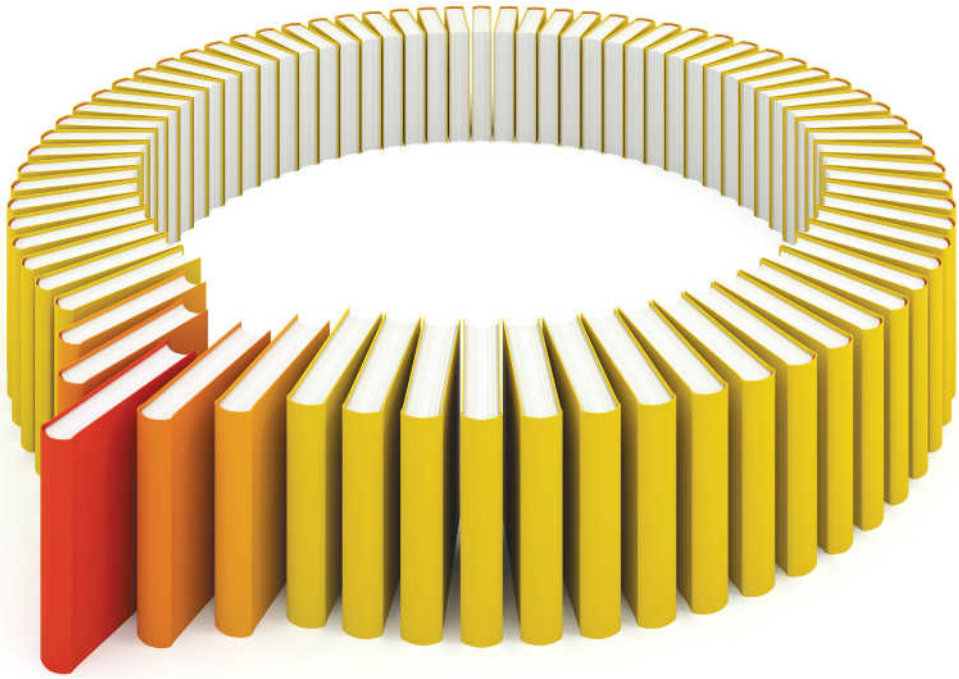
S
SAS
  table driver 178
SAS Micro Analytic Service
  concepts 3, 4, 5
SCAN 22
single computation 26
SQL server legacy wire protocol driver 167

T
TRANWRD 22

tuning
  adjusting thread pool size 50

U
user context 4

W
wire protocol driver 173



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.