

SAS® Micro Analytic Service 1.3 Programming and Administration Guide



SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2016. SAS® Micro Analytic Service 1.3: Programming and Administration Guide. Cary, NC: SAS Institute Inc.

SAS® Micro Analytic Service 1.3: Programming and Administration Guide

Copyright © 2016, SAS Institute Inc., Cary, NC, USA

All rights reserved. Produced in the United States of America.

For a hard-copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a) and DFAR 227.7202-4 and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

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

March 2016

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

Contents

	About This Book	. v
	Accessibility	
Chapter 1 • Intro	duction to SAS Micro Analytic Service	1
	What Is SAS Micro Analytic Service?	
		1
Chapter 2 • Conc	epts	3
	Overview	
	User or Business Context	
	Module Context	
	Revision	
	Interfaces	
	Example: JAVA Interface	. 6
Chapter 3 • DS2	Programming for SAS Micro Analytic Service	
	Overview	
	Publishing DS2 Source Code to SAS Micro Analytic Service	
	SAS Micro Analytic Service and SAS Foundation	
	I/O	. 12
	Programming Blocks	13
	Public and Private Methods and Packages	. 14
	Argument Types Supported in Public Methods	17
	DS2 Interface to Python	
	, ,	
Chapter 4 • Best	Practices for DS2 Programming	23
	Overview .	
	Global Packages Versus Local Packages	
	Replacing SCAN (and TRANWRD) with DS2 Code	
	Hash Package	
	Character-to-Numeric Conversions	
	Passing Character Values to Methods	
	Performing the Computation Once	
	Moving Invariant Computations Out of Loops	28
	an Oran a the OAO Mirror Analytic Oran is a	~~
Chapter 5 • Pythe	on Support in SAS Micro Analytic Service	
	Implementing Python Support	
	Configuring Python	30
Chapter 6 • Java	Interface Reference	33
	Overview	. 33
	Topology	. 34
	Start-Up	. 34
	Shutdown	35
	User Context Methods	
	Module Context Methods	
	Revision Methods	
	Execution Methods	
	Execute Method	
	Revision Monitoring Methods	
		51

	Complete Java Example	
Chapter 7 • SA	AS Micro Analytic Service REST API	
	Overview	
	Terminology	59
	Client Application Features	
	Security and Authentication	
	Life Cycle	
	Media Types	
	SAS Micro Analytic Service Media Types	
	Resources and Collections	
Chapter 8 • Au	dministration	123
onuplei o Al	SAS Micro Analytic Service Logging	
	Secure DS2 HTTP Package Usage	
	Monitoring	
	Moliitoring	124
Chapter 9 • De	eployment and Tuning	
	Pre-Installation Steps	129
	Deployment	
	Post-Installation Steps	131
	Cluster Deployment for SAS Micro Analytic Service	
	Tuning SAS Micro Analytic Service	
Chaptor 10 .	Migration	137
Chapter 10 ° M	Overview	
	Limitations	
	Back Up Disk Stores	
	Restore Script	
	Additional Migration Considerations	139
Appendix 1 • S	SAS Micro Analytic Service Return Codes	143
Annondix 2 • 1	REST Server Error Messages and Resolutions	153
Appendix 3 • 🕻	Table Service Driver Reference	
	DB2 Driver Reference	158
	FedSQL Driver Reference	163
	Greenplum Driver Reference	166
	Netezza Driver Reference	
	ODBC Driver Reference	
	Oracle Reference	
	PostgreSQL Driver Reference	
	SAS Data Set Reference	
	Teradata Reference	
	Recommended Reading	
	Index	

About This Book

Audience

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

Accessibility

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

viii About This Book

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 1.3 is a memory-resident, high-performance program execution service. As a SAS platform service, it is not available for individual license, but is included in selected SAS solutions. SAS Micro Analytic Service 1.3 provides hosting for DS2, Python, and C programs and supports a "compile-once, execute-many-times" usage pattern. SAS Micro Analytic Service is multi-threaded and can be clustered for high availability. It can host multiple programs simultaneously, as well as multiple user or business contexts that are isolated from one another.

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

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

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

2 Chapter 1 • Introduction to SAS Micro Analytic Service

Chapter 2 Concepts

Overview	3
User or Business Context	3
Module Context	1
Revision	1
nterfaces	5
Example: JAVA Interface	5
Instantiate SAS Micro Analytic Service	5
Create a User Context	
Create Modules	7
Basic Steps for Using SAS Micro Analytic Service	3

Overview

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

- *Note:* If you are writing DS2 code to deploy to SAS Micro Analytic Service, follow the programming guidelines described in Chapter 3, "DS2 Programming for SAS Micro Analytic Service," on page 11.
- 1. User or business context
- 2. Module context
- 3. Revision

User or Business Context

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

4 Chapter 2 • Concepts

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

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

Module Context

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

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

Note: The Micro Analytic Service 1.3 REST interface supports running the latest revision only. The Java and C interfaces support multiple revisions.

Revision

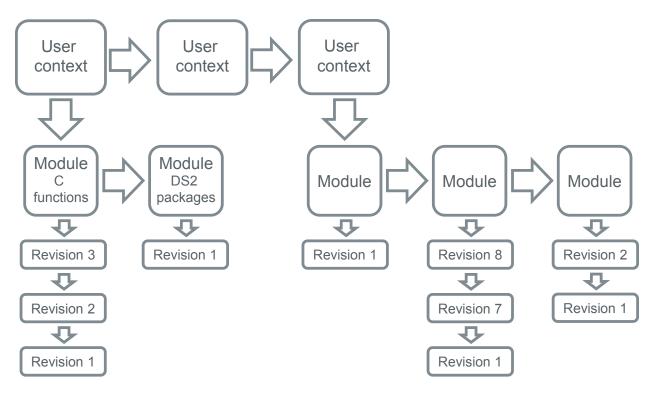
A revision is a version of a module. It is at the leaf level of the object hierarchy. Each revision contains source code, an executable code stream (optimized binary executable), and metadata. The metadata describes the methods, method signatures, and entry points in the code.

Revisions provide several advantages, including the ability to roll back to a previous version of a module. Further, when the Java or C interfaces are used, revisions allow two or more versions of a module to execute concurrently, which might be useful in champion and challenger testing of predictive models.

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

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

Figure 2.1 Component Hierarchy



Interfaces

SAS Micro Analytic Service has a layered set of interfaces:

С

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

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

Java

a thin Java layer that communicates with the C interface through the Java Native Interface (JNI). The Java interface supports the same functionality as the C interface.

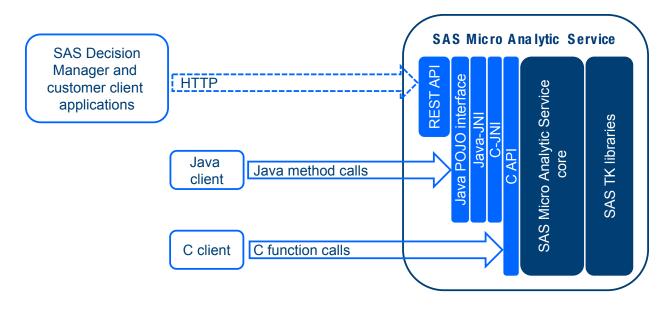
REST/JSON

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

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

The following example illustrates SAS Micro Analytic Service interfaces enabling you to publish code and execute it many times. Although you might be using SAS Micro Analytic Service 1.3 through the REST interface, this example uses the Java interface in order to show all of the steps, including those that the REST interface handles automatically.

Figure 2.2 Interfaces Example



Example: JAVA Interface

Instantiate SAS Micro Analytic Service

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

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

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

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

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

Create a User Context

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

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

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

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

Create Modules

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

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

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

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

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

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

int rev = tk.newRevision(moduleCtx, myCode,

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

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

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

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

tksfValues args = new tksfValues(2, 1);

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

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

Note: The REST API does not support method overloading.

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

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

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

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

Basic Steps for Using SAS Micro Analytic Service

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

Figure 2.3 Annotated Steps

1.	Instantiate SAS Micro Analytic Service. TkLight tk = new tksfjni(32, myLoggingCfg.xml);	TK thread pool size.
		A user context is a module container, and provides an isolated execution environment.
2.	Get a user or business context.	
userCtx = tk.new	userCtx = tk.newUserContext("A user context");	A module context is a revision container, and represents a DS2 package.
3.	Create modules. moduleCtx = tk.newModuleContext(userCtx, Language.DS2, "Network risk score test", 0 revision = tk.newRevision(moduleCtx, testDS2, "Network risk score code", null, 0););

4. Execute many times.

rc = tk.execute(userCtx, moduleCtx, revisionNumber, methodName, arguments);

A revision has an executable code stream with an entry point for each DS2 package method, source code, and signature metadata. Chapter 2 • Concepts

Chapter 3 DS2 Programming for SAS Micro Analytic Service

Overview	. 11
Publishing DS2 Source Code to SAS Micro Analytic Service	11
SAS Micro Analytic Service and SAS Foundation	. 12
Ι/Ο	12
Programming Blocks	. 13
Public and Private Methods and Packages	
Public Method Rules	14 .15
Method Overloading	
Argument Types Supported in Public Methods	17
Unsupported DS2 Data Types	
DS2 Interface to Python	. 18

Overview

SAS Micro Analytic Service 1.3 supports a subset of the DS2 programming language that is suitable for high-performance transaction processing in real time. This chapter covers only that subset. Note that DS2 batch processing is not supported.

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

Publishing DS2 Source Code to SAS Micro Analytic Service

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

"ds2_options sas"

It should end with

"endpackage"

The code cannot contain DATA statements, PROC statements, or 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. This line feed character makes it easier to use compiler warning and error messages that include line numbers.

SAS Micro Analytic Service and SAS Foundation

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

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

ds2_options sas;

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

I/O

SAS Micro Analytic Service 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. If you use the Java API, you can set your connection string by calling setDS2Connection().

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 157.

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 or the Java API, 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 will detect this condition and recompile 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 will occur. 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 157.

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, so the DS2 PACKAGE statement plays a major role in SAS Micro Analytic Service. A DS2 package contains one or more methods, and methods can contain a wide variety of DS2 language constructs. Package methods work well with rapid transaction processing because they can be called over

14 Chapter 3 • DS2 Programming for SAS Micro Analytic Service

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 might not be passed as arguments in public methods. The presence of a DS2 package argument results in the method becoming private.
- The VARARRAY statement might 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 17.

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);</pre>
      /**
       * 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);</pre>
        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;
```

end;

```
/* 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;</pre>
```

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

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

Private Method Example

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

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

Method Overloading

SAS Micro Analytic Service supports method overloading. In DS2, when two or more methods in the same package have the same name, those methods are said to be *overloaded*. When overloaded methods are used, the method signature (list of input and

output parameters and their types) is used to select the correct method to execute. Because a method signature includes both input and output parameters, any output parameter types must always be set in tksfValues before executing the method.

- *Note:* Each module constitutes a separate namespace and corresponds to one DS2 package. Therefore, two DS2 methods with the same name, in different modules, are not considered overloaded.
- *Note:* The C language does not support method overloading. Syntax errors occur if two C functions with the same name exist in the source code of the same C module. Therefore, only DS2 package methods can be overloaded in SAS Micro Analytic Service 1.3.
- *Note:* SAS Micro Analytic Service 1.3 supports one Python script per module. Therefore, method overloading is not applicable to Python.

The following functions enable you to query information about overloaded methods. For more information about these methods, see Chapter 6, "Java Interface Reference," on page 33.

getStepInputs()

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

getStepOutputs()

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

getOverloadedStepCount()

returns the number of overloaded methods that exist in the specified module having the specified name.

isOverloaded()

returns True if the specified method is overloaded and False if not.

Argument Types Supported in Public Methods

Overview

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

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

Supported DS2 Data Types

- 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 29, 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.

Each package instance represents exactly one module revision. You can create as many instances as desired, allowing multiple modules to be used. The module name (or the Python script name) is given as a package constructor argument. The constructor automatically sets up user and module contexts to house the instance.

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

Instantiate the following DS2 package:

py = _new_ maspy("myPythonScript");

Calling publish() compiles your Python script and creates the revision. The Python code is passed as a string in the first argument. A list of argument names is required by Python and is passed as an array of strings in the second argument. As is the case with any method signature in SAS Micro Analytic Service, all of the input arguments must precede the output arguments in the list.

rc = py.publish(python_script, arguments, numinputs);

Call setType methods to set input values before executing the script. Because these setters store arguments by name, they can be called in any order and will 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 following setOut methods reserve space in the argument list for output arguments.

```
rc = setOut<type>(name)
rc = setOut<type>Array(name)
```

Note: If an array is not originally allocated, or if it is allocated with insufficient size, it will be allocated or reallocated with sufficient size to accommodate DS2.

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_script, symbols, number_of_inputs);
py.setString("inString", "A string");
rc = py.setOutLong("outLong");
py.execute()
result = py.getLong("outLong");
```

The requestInterrupt(userContext, moduleContext, revision, entryPoint) method can be used to halt the execution of a runaway module or method, such as a method containing an infinite loop. It can be used to halt the execution of DS2 modules, Python modules, or DS2 modules that call Python. When interrupting a Python module, the entryPoint parameter is ignored, since a Python module represents one entire Python script.

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

Methods for Python module management and execution:

rc = py.publish(python_script, arguments, numinputs);

- rc = py.remove();
- rc = py.isLoaded(); // returns true is Python is available and false otherwise
- rc = py.getModuleContextID();
- rc = 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 =	<pre>py.getString(argument_name);</pre>
int_value	<pre>= py.getBool(argument_name);</pre>
long_value	<pre>= py.getLong(argument_name);</pre>
int_value	<pre>= py.getInt(argument_name);</pre>
double_value	<pre>= py.getDouble(argument_name);</pre>
date_time_value	<pre>= py.getDateTime(argument_name);</pre>
date_value	<pre>= py.getDate(argument_name);</pre>
time_value	<pre>= py.getTime(argument_name);</pre>

Scalar output argument setters:

- rc = py.setOutString(argument_name);
- rc = py.setOutBool(argument_name);
- rc = py.setOutLong(argument_name);
- rc = py.setOutInt(argument_name);
- rc = py.setOutDouble(argument_name);
- rc = py.setOutDateTime(argument_name);
- rc = py.setOutDate(argument_name);
- rc = py.setOutTime(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);

Array output argument setters:

- rc = py.setOutStringArray(argument_name);
- rc = py.setOutBoolArray(argument_name);
- rc = py.setOutLongArray(argument_name);
- rc = py.setOutIntArray(argument_name);
- rc = py.setOutDoubleArray(argument_name);
- rc = py.setOutDateTimeArray(argument_name);
- rc = py.setOutDateArray(argument_name);
- rc = py.setOutTimeArray(argument_name);

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.

Chapter 4 Best Practices for DS2 Programming

Overview	23
Global Packages Versus Local Packages	
Example of Poorly Optimized Code	24
Replacing SCAN (and TRANWRD) with DS2 Code	. 24
Hash Package	27
Character-to-Numeric Conversions	27
Passing Character Values to Methods	27
Performing the Computation Once	28
Moving Invariant Computations Out of Loops	28

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(N²).
- 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;
 /\ast 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.

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);</pre>
```

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;</pre>
```

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

Implementing Python Support	29
Configuring Python	30
Python 2.7 and 3.4 on 64–Bit Windows	30
Python 2.7 and 3.4 on 64–Bit Linux	30
Further Considerations for Configuring Python	31

Implementing Python Support

SAS Micro Analytic Service 1.3 supports modules that are written in the Python programming language. A Python module represents a single Python script. Python modules can be published and called from DS2 (see "DS2 Interface to Python" on page 18). If your SAS solution supports it, Python modules can be published and called directly through the SAS Micro Analytic Service Java interface. To use Python modules directly from the Java interface:

1. Use a value of SF_PYTHON_MODULE when calling newModuleContext() to create a new Module Context.

```
Sasdsmas mas = new tksfjni(numServerThreads,
    logCfgLocation);;
// Create module context
long moduleCtx = mas.moduleCtx(userCtx, Language.PYTHON,
    "My Moudle Name", null);
```

- 2. Call sfNewRevision(), and use the Python script code and an array of strings containing the parameter names.
 - *Note:* The order of the parameter names is important. Match the order in which the parameters were set in the symbol table that is sent to the Execute method. This matching is needed in order to send parameter values between SAS Micro Analytic Service and Python.

```
// Create new Revision
int rev = mas.newRevision(moduleCtx, pythonCode,
    "My Moudle Name", paramNames, null);
```

3. Call execute() to execute the Python script, using the symbol table containing the input values.

```
tksfValues symbolTable = new tksfValues(numOfParameters, numInputs);
... (fill in symbol table with both input and output parameters)
```

int rc = mas.execute(userCtx, moduleCtx, 0, null, symbolTable);

Missing input values are translated to NONE in Python. To indicate a missing output value, set the variables value to NONE within your Python script.

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 you installed the folder in C:\Anaconda3.

 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. 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, once 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/ LevN/Web/WebAppServer/SASServer13_N/bin directory.

Python 2.7 and 3.4 on 64–Bit Linux

- 1. Download Anaconda for Linux, PYTHON 3.5, Linux 64-bit installer from https:// www.continuum.io/downloads.
- 2. After downloading the installer, enter the following 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 **Do you wish the installer to prepend the Anaconda3 install location to PATH in your .bashrc?** These instructions assume that you have used the location /users/myuserid/anaconda3.

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

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

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

source activate python34

5. 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.

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. This does not work 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 the SAS Web Application Server, do so from a shell, in which you have activated Python, thus allowing the server 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.

When you add packages to an Anaconda environment, the packages are placed in <*your-environment-path*>/lib/python3.4/site-packages. To make the Python scripts that use these packages work, 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_ENV_PATH when you activate an environment, and you can use CONDA_ENV_PATH when setting PYTHONPATH.

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_ENV_PATH/lib/python3.4/site-packages
export PYTHONPATH=.:\$CONDA_ENV_PATH/lib/python3.4
export PYTHONPATH=\$PYTHONPATH:\$SITE_PACKAGES
export PYTHONPATH=\$PYTHONPATH:\$SITE_PACKAGES/numpy
export PYTHONPATH=\$PYTHONPATH:\$SITE_PACKAGES/numpy/lib

Chapter 6 Java Interface Reference

Overview	33
Topology	. 34
Start-Up	. 34
Shutdown	. 35
User Context Methods	35
Module Context Methods	. 37
Revision Methods	. 40 . 40
Execution Methods	. 45 45 . 46 . 47
Execute Method	50
Revision Monitoring Methods	51
Complete Java Example	. 51

Overview

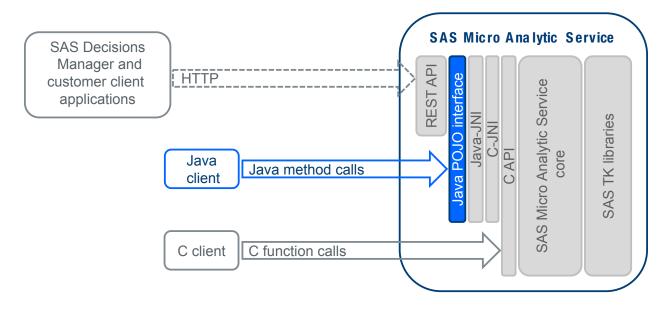
The Java interface allows tightly coupled Java client applications to drive SAS Micro Analytic Service directly though Java method calls. This is made possible because the Java interface provides fine-grained control of SAS Micro Analytic Service, and does not hide detailed functionality. By contrast,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

The Java interface enables client-supplied DS2, Python, and C programs to be published to SAS Micro Analytic Service, where they are compiled into modules and made available for repeated execution. The interface also includes methods for querying information about currently loaded content, such as user contexts, modules, methods, and step signatures (input and output arguments).

Topology

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

Figure 6.1 Java Interface



Start-Up

The SAS Micro Analytic Service 1.3 Java interface is a public Java interface named TkLight, obtained by instantiating Java class tksfjni, which implements the interface.

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

CAUTION:

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

The first tksfjni constructor argument specifies the size of the threaded kernel thread pool to be maintained by SAS Micro Analytic Service. For best performance, the size of the thread pool should be about equal to the number of processor cores in the server hosting SAS Micro Analytic Service. The exact number of threaded kernel threads to specify for best performance varies somewhat, depending on the characteristics of the code to be published. When the number of threads that is specified exceeds four times the number of cores of the hosting server, SAS Micro Analytic Service fails to start.

If zero is specified for the number of threads, SAS Micro Analytic Service queries the operating system for the number of processor cores of the hosting server and uses a thread pool size equal to the number of cores. If zero is specified for the number of threads, and the host operating system does not support querying the number of processors, SAS Micro Analytic Service uses a default thread pool size of 5.

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

Shutdown

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

Because SAS Micro Analytic Service is an in-memory service, all published artifacts (user contexts, modules, and revisions) are purged. If SAS Micro Analytic Service is restarted when using the Java interface, any previously purged artifacts are not restored and must be re-published if desired. However, the REST interface includes an artifact persistence feature that restores the state of SAS Micro Analytic Service automatically. For more information, see Chapter 7, "SAS Micro Analytic Service REST API," on page 57.

User Context Methods

User (or business) contexts have two primary functions:

- A user context is a container of modules. Every module is parented to one and only one user context.
- User contexts provide isolated execution environments. The operations of one user context are not visible to any other user context.
- *Note:* Recall that there is no functional difference between a user context and a business context. Whether contexts are used to provide each user with their own environment, or whether they are used to insulate business units from one another, is application-specific. The set of functionality is the same. SAS Micro Analytic Service uses the UserContext construct for both business and user contexts.

Here is a list of available methods:

newUserContext

long newUserContext(java.lang.String displayName)

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

Parameter

displayName - The name that is used in log messages.

Return

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

deleteUserContext

void deleteUserContext(long userContext)

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

Parameter

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

userContextExists

boolean userContextExists(long userContext)

Queries the existence of a user context.

Parameter

userContext - The opaque user context handle.

Return

Boolean - True if the context exists, False otherwise.

getUserContextDisplayName

java.lang.String getUserContextDisplayName(long userContext)

Retrieves the display name of the given user context.

Parameter

userContext - The opaque user context handle.

getModuleIDs

long[] getModuleIDs(long userContext)

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

Return

The long array of the module IDs.

getModuleNames

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

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

Returns

The string array of the module IDs.

- *Note:* The C modules are named according to the value of the displayName argument passed to newModuleContext(). DS2 modules are named for the DS2 package that the module represents (such as the package name given in the source code).
- *Note:* If the value of the displayName argument to newModuleContext() differs from the DS2 package name that was given in the source code, the module name changes when the first revision of the DS2 package is created.

getUserContextCreationDateTime

java.util.Date getUserContextCreationDateTime(long userContext)

Gets the user context creation datetime.

Parameter

userContext - The opaque user context handle.

Return

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

getUserContextLastModifiedDateTime

java.util.Date getUserContextLastModifiedDateTime(long userContext)

Gets the user context last modified datetime.

```
Parameter
```

userContext - The opaque user context handle.

Return

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

Here is an example of creating a new user context:

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

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

Module Context Methods

The terms *module* and *module context* are used interchangeably. In SAS Micro Analytic Service 1.3, a module context represents any of the following:

named C module

a collection of related C functions.

DS2 package

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

Python script

a Python script with associated data dictionary.

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

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

Here is a list of available module contexts:

newModuleContext

long newModuleContext(long userContext, TkLight.Language language, java.lang.String displayName, java.util.EnumSet<TkLight.ModuleOptions> options) Creates a new module context within the given user context. A module is a collection of related steps (a DS2 package or collection of C routines). Add code to the module context by calling newRevision().

- *Note:* For DS2 packages, the module displayName is assigned the DS2 package name when the first revision is created.
- *Note:* DS2 packages containing non-void methods or methods with DS2 package arguments are treated as internal and are not returned in queries for the list of methods. It is permissible to include private methods that do not produce errors, as long as the code is valid. See "Public and Private Methods and Packages" on page 14.

Parameters

userContext - The opaque handle returned from newUserContext().

language - Source language (C or DS2).

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

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

Return

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

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

deleteModuleContext

void deleteModuleContext(long moduleContext)

Deletes the specified module context and its modules and revisions.

Parameter

moduleContext - The opaque module context handle

moduleContextExists

boolean moduleContextExists(long moduleContext)

Queries the existence of a module context.

Parameter

moduleContext - The opaque module context handle

Return

Boolean - True if context exists, False otherwise.

getModuleContextDisplayName

java.lang.String
getModuleContextDisplayName(long moduleContext)

Retrieves the display name of the given module context.

Parameter

moduleContext - The opaque module context handle.

getModuleContextByName

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

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

Parameters

userContext - The opaque handle returned from newUserContext().

name - DS2 package name or C module display name.

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

getModuleContextCreationDateTime

```
java.util.Date
```

getModuleContextCreationDateTime(long moduleContext)

Gets the module context creation datetime.

Parameter

moduleContext - The opaque handle returned from newModuleContext().

Returns

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

getModuleContextLastModifiedDateTime

```
java.util.Date
```

getModuleContextLastModifiedDateTime(long moduleContext)

Gets the module context's last modified datetime.

Parameter

moduleContext - The opaque handle returned from newModuleContext().

Return

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

registerModuleName

int registerModuleName(long moduleContext,

java.lang.String name)

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

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

Parameter

moduleContext - The opaque handle returned from newModuleContext().

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

Return

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

Revision Methods

Overview

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

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

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

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

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

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

Parameter Descriptions

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

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

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

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

intArray_t

Method Descriptions

Here are the available revision methods:

newRevision

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

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

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

- *Note:* For DS2 packages, the parent module context displayName is assigned the DS2 package name (given in the source code) when the first revision of that module context is successfully created. After the first revision is created, the name cannot be changed. Attempts to create a revision of a DS2 module with a different package name than the original revision result in an error.
- *Note:* DS2 packages containing non-void methods or methods with DS2 package arguments are treated as private and are not returned in queries for methods. It is permissible to include private methods, and private methods do not produce errors as long as the code is valid. For more information, see "Public and Private Methods and Packages" on page 14.

Parameters

moduleContext - The opaque handle returned from newModuleContext().

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

description - User-supplied text.

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

Return

revisionNumber - The new revision number.

deleteRevision

void deleteRevision(long moduleContext,

```
int revision)
```

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

Parameter

moduleContext - The opaque handle returned from newModuleContext().

getCompilationMessages

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

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

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

Parameter

moduleContext - The opaque handle returned from newModuleContext().

Return

An array of strings containing compiler diagnostic messages.

revisionIsValid

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

Parameters

moduleContext - The opaque handle returned from newModuleContext().

revision - The revision number returned from newRevision().

Return

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

getStepIDs

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

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

Parameters

moduleContext - The opaque handle returned from newModuleContext().

revision - The revision number returned from newRevision().

Return

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

getStepDescription

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

Retrieves the step description.

Parameters

moduleContext - The opaque handle returned from newModuleContext().

revision - The revision number returned from newRevision().

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

Return

The description of the step.

getStepInputs

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

Retrieves the step input arguments.

Parameters

moduleContext - The opaque handle returned from newModuleContext().

revision - The revision number returned from newRevision().

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

Return

An ArrayList of parameter definitions. For more information, see "Parameter Descriptions" on page 40.

getStepInputs

java.util.ArrayList<tksfParmdef> getStepInputs

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

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

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

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

Parameters

moduleContext - The opaque handle returned from newModuleContext().

revision - The revision number returned from newRevision().

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

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

Return

An ArrayList of parameter definitions. For more information, see "Parameter Descriptions" on page 40.

getStepOutputs

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

Retrieves the step output arguments.

Parameters

moduleContext - The opaque handle returned from newModuleContext().

revision - The revision number returned from newRevision().

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

Return

An ArrayList of parameter definitions. For more information, see "Parameter Descriptions" on page 40.

getStepOutputs

java.util.ArrayList<tksfParmdef> getStepOutputs

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

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

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

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

Parameters

moduleContext - The opaque handle returned from newModuleContext().

revision - The revision number returned from newRevision().

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

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

Return

An ArrayList of parameter definitions. For more information, see "Parameter Descriptions" on page 40.

getOverloadedStepCount

int getOverloadedStepCount(long moduleContext,

```
int revision,
java.lang.String stepID)
```

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

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

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

Parameters

moduleContext - The opaque handle returned from newModuleContext().

revision - The revision number returned from newRevision().

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

Return

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

isOverloaded

boolean isOverloaded(long moduleContext,

int revision, java.lang.String stepID)

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

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

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

Parameters

moduleContext - The opaque handle returned from newModuleContext().

revision - The revision number returned from newRevision().

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

Return

True, if the method is overloaded, False otherwise.

getRevisionCreationDateTime

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

Gets the revision creation datetime.

Parameters

moduleContext - The opaque handle returned from newModuleContext().

revision - The revision number returned from newRevision().

Return

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

Execution Methods

Overview

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

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

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

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

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

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

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

Java Data Types

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

- String
- Character
- Long
- Integer
- Double

The following Java array types are supported:

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

Method Arguments

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

tksfValues p = new tksfValues(numArgsTotal, numInputs);

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

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

Output arguments must be set for two reasons:

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

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

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

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

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

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

Incorrect:

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

Correct:

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

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

Argument Setter Methods

Overview

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

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

Setters with an Implicit Index

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

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

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

Setters with an Explicit Index

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

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

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

Missing Value Setters with an Implicit Index

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

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

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

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

Missing Value Setters with an Explicit Index

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

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

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

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

Output Setters with an Implicit Index

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

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

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

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

Output Setters with an Explicit Index

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

These methods use explicit indexes, and set the value of the input argument at the zerobased index position given. The argument to the array setters (dim) specifies the size of the one-dimensional array. Each element of the array is set to a missing value.

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

Argument Getter Methods

Overview

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

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

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

Missing Value Check

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

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

Getters with an Implicit Index

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

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

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

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

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

Getters with an Explicit Index

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

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

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

Miscellaneous tksfValues Methods

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

public void clear();

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

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

Execute Method

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

Here are the parameters:

userContext

The opaque handle returned from newUserContext().

moduleContext

The opaque handle returned from newModuleContext().

revision

The revision number returned from newRevision().

entryPoint

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

arguments

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

Revision Monitoring Methods

Here are the revision monitoring methods:

getRevisionHitCount

long getRevisionHitCount(long moduleContext,

int revision)

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

Parameters

moduleContext - The opaque pointer returned from newModuleContext().

revision - The revision number returned from newRevision().

Return

The number of times the revision methods have been called.

getRevisionAvgLatency

double getRevisionAvgLatency(long moduleContext,

int revision)

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

Parameters

moduleContext - The opaque pointer returned from newModuleContext().

revision - The revision number returned from newRevision().

Return

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

getTotalRevisionExecutionTimeSeconds

double getTotalRevisionExecutionTimeSeconds(long moduleContext,

int revision)

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

Parameters

moduleContext - The opaque pointer returned from newModuleContext().

revision - The revision number returned from newRevision().

Return

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

Complete Java Example

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

The example code performs the following sequence of steps:

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

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

```
package com.sas.mas.test;
import java.util.ArrayList;
import com.sas.mas.TkLight;
import com.sas.mas.tksfParmdef;
import com.sas.mas.tksfValues;
import com.sas.mas.TkLight.Language;
import com.sas.mas.jni.tksfjni;
public class SimpleDS2Example {
    /*
       This is a simple DS2 package with two methods. The code could have
       been read from a file, but is included here for easy reference.
       The source code starts with "ds2_options sas" and ends with
       with "endpackage." This pattern should be used with all DS2 to be
       published to SAS Micro Analytic Service.
       Note: each source code line ends with a line-end character.
       This best practice facilitates use of the line numbers included in
       compiler messages, making it easier to locate syntax errors.
    */
    static String DS2 =
"ds2 options sas;
                                                                               \n" +
"package simple example /overwrite=yes;
                                                                               \n" +
                                                                               \n" +
   method str2double (char(12) numericString, in_out double number);
п
                                                                               \n" +
п
       number = put( numericString, 8.0);
                                                                               \n" +
       /* Include an undeclared variable to illustrate a compiler warning */ \n" +
п
п
       anotherNumber = number;
                                                                               \n" +
п
   end;
                                                                               \n" +
п
                                                                               \n" +
```

```
method flip_string(varchar(32767) in_string, in_out varchar out_string); \n" +
      /* Reverse the input string and set in output string */
                                                                            \n" +
      out string=reverse(in string);
                                                                            \n" +
                                                                            \n" +
   end;
                                                                            \n" +
"endpackage;
                                                                            n";
   /**
    * @param parms
    */
   public static void main(String[] parms) {
       int
                  rc;
       String stringToReverse = new String("This is a test...");
       String stringToConvert = new String("0.9997");
       long userCtx
                                  = -1;
       long moduleCtx
                                   = -1;
       // Start SAS Micro Analtyic Service with four threads and
       // no logging configuration file location (null second argument)
       int threads = 4;
       TkLight tk = new tksfjni(threads, null);
       System.out.println("*** Simple example of using SAS Micro Analytic Service ***");
       // Create a user context.
       userCtx = tk.newUserContext("A user context");
       if (userCtx <= 0) {
           System.out.println(" User context creation failed.");
           return;
       }
       else {
           System.out.println(" User context created at " +
                   tk.getUserContextCreationDateTime(userCtx) + ".");
       }
       /* Create a module context.
           This module context is owned by the user context just created.
           The language is specified as DS2. Therefore, all revisions
        of the module must be DS2.
           The module is named for the DS2 package it represents, or
            "simple example" in this case.
           Pass null for the last argument, which specifies the default options.
        */
       moduleCtx = tk.newModuleContext(userCtx,
                                       Language.DS2,
                                       "simple_example",
                                       null);
       if (moduleCtx <= 0) {</pre>
           System.out.println(" Module context creation failed.");
           tk.term();
           return;
       }
       else {
           System.out.println(" Module context created at " +
                   tk.getModuleContextCreationDateTime(moduleCtx) + ".");
       }
```

н п

п

п

п

```
/* Publish the DS2 to SAS Micro Analytic Service by calling newRevision().
            Pass in the module context ID, the source code String, and an
           optional description.
           Pass null for the fouth argument, a list of entry points, which
           is only used for C modules.
           Pass null for the last argument, which specifies default options.
        */
       int revision = tk.newRevision(moduleCtx, DS2,
                "A simple DS2 example to illustrate SAS Micro Analytic Service usage.",
               null, null);
       if (revision <= 0) {
           System.out.println(" Compilation failed");
           String[] messages = tk.getCompilationMessages(moduleCtx);
           if (messages.length > 0) {
               System.out.println("
                                     Compiler messages:");
               for (String msg : messages) {
                   System.out.println("
                                                 " + msg);
               }
            }
           System.out.println(" Revision creation failed.");
           tk.term();
           return;
       }
       else {
           System.out.println(" Revision " + revision + " created at " +
                      tk.getRevisionCreationDateTime(moduleCtx, revision)
+ ".");
       }
       System.out.println(" DS2 Package: " +
                           tk.getModuleContextDisplayName(moduleCtx));
       /* Even successful compilations can produce warning messages from the compiler.
          The undeclared variable included in the source code generates such a
          warning. However, it will not prevent the code from publishing and executing
          properly.
        */
       String[] messages = tk.getCompilationMessages(moduleCtx);
       if (messages.length > 0) {
           System.out.println("
                                     Compiler messages:");
           for (String msg : messages) {
               System.out.println("
                                            " + msg);
            }
       }
        /* Once published, SAS Micro Analtic Service can be queried for information
       about the revision. Here you are asked for the inputs to the str2double
       method. Parameter metadata is represented by class tksfParmdef.
        */
       ArrayList<tksfParmdef> inputs = tk.getStepInputs(moduleCtx,
                                                        revision,
                                                         "str2double");
       System.out.println(" Input arguments to method 'str2double':");
       for (tksfParmdef p : inputs) {
            System.out.println("
                                 Input name: '" + p.name + "' type: " +
```

```
p.getType());
 }
  /* Setup arguments to call method "str2double", which has one String input and
    one double output.
 Note: Setters are being used with implicit indices.
  */
 int
           numArgs = 2;
 int
           numInputs = 1;
 tksfValues args = new tksfValues(numArgs, numInputs); // DS2 method arguments
 args.setString(stringToConvert);
 args.setOutDouble();
 String methodName = "str2double";
 // Execute the DS2 package method "str2double"
 rc = tk.execute(userCtx, moduleCtx, revision, methodName, args);
 if (rc != 0) {
     System.out.println(" Bad return code from execute:" + rc);
     if (rc == 29) {
         System.out.println(" Exception occurred during execution of " +
                     methodName + " in the TK environment.");
     }
     tk.term();
     return;
  }
 else {
     // Print results (Getters are being used with explicit indices.)
     System.out.println(" Results of calling str2double:");
     System.out.println(" Input String: " + args.getString(0));
     System.out.println(" Output double (rounded): " + args.getDouble(1));
 }
 // Setup arguments to call the method "flip_string", which has one String input
 // and one double output. Setters are being used with implicit indices.
 numArgs = 2;
 numInputs = 1;
 args.setString(stringToReverse);
 args.setOutString();
 methodName = "flip_string";
 // Execute the DS2 package method "flip_string"
 rc = tk.execute(userCtx, moduleCtx, revision, methodName, args);
 if (rc != 0) {
     System.out.println(" Bad return code from execute: " + rc);
     if (rc == 29) {
         System.out.println(" Exception occurred during execution of " +
                     methodName + " in the TK environment.");
     }
     tk.term();
     return;
 }
 else {
     // Print results (Getters are being used with explicit indices.)
     System.out.println(" Results of calling flip_string:");
     System.out.println("
                            Input String: " + args.getString(0));
```

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

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

```
*** Simple example of using SAS Micro Analytic Service ***
  User context created at Tue Apr 07 17:50:28 EDT 2015.
  Module context created at Tue Apr 07 17:50:28 EDT 2015.
  Revision 1 created at Tue Apr 07 17:50:28 EDT 2015.
  DS2 Package: simple_example
     Compiler messages:
        Line 6: No DECLARE for assigned-to variable anothernumber;
         creating it as a global variable of type double.
   Input arguments to method 'str2double':
   Input name: 'numericString' type: string t
  Results of calling str2double:
     Input String:
                                                            0.9997
     Output double (should round to nearest whole number): 1.0
  Results of calling flip_string:
     Input String: This is a test...
     Output String: ...tset a si sihT
*** Simple DS2 example complete ***
```

Chapter 7 SAS Micro Analytic Service REST API

Overview	8
Terminology	9
Micro Analytic Service	9
Micro Analytic Module	9
Micro Analytic Step	
Package	
Method	
Signature	9
Input Signature	
Output Signature	
Module	
Module ID	
Module Name	
Step	
Step ID	
Source Code	0
Client Application Features	
Post Load or Create Modules	
Get Input or Output Step Signatures	
Post Validate Input Variables	
Post Execute Modules	
Put Update Modules	
Delete Modules	I
Security and Authentication	1
Life Cycle	2
Media Types	า
Externally Defined Media Types	
SAS Micro Analytic Service Media Types	
application/vnd.sas.microanalytic.module	
application/vnd.sas.microanalytic.module.definition	
application/vnd.sas.microanalytic.module.source	8
application/vnd.sas.microanalytic.module.step	0
application/vnd.sas.microanalytic.module.step.input	
application/vnd.sas.microanalytic.module.step.input.validity	6
application/vnd.sas.microanalytic.module.step.output	7
Resources and Collections	9
Resource /	9

Collection /modules	. 80
Resource /modules/{moduleId}	. 92
Resource /modules/{moduleId}/source	100
Collection /modules/{moduleId}/steps	. 102
$Resource \ /modules / \{moduleId\} / steps / \{stepId\} \ \ldots \ $	115

Overview

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

The API provides the following POST methods:

Create module

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

Validate steps

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

Execute step

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

The API provides the following PUT method:

Update module

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

The API provides the following DELETE method:

Delete module

removes analytic code from memory.

The API provides the following GET methods:

- Query an individual module returns detailed information about a module
- Query steps by module returns a list of steps available by module.
- Query an individual step by module returns detailed information about the inputs required by the step and the outputs produced by the step.

Retrieve module details

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

The implementation supports only JSON resource representations.

Note: The REST API does not support method overloading.

Terminology

Micro Analytic Service

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

Micro Analytic Module

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

Micro Analytic Step

	A unit of analytical logic that is executed. It includes input and output values. Here is an example: the name value pairs that contain the input values required to execute the step and the output values that are generated as a result of its execution. For DS2 source, a step is defined as a method. When the step is executed, a specific method in the module is executed.
Package	An assembly of methods defined by a DS2 source.
Method	A unit of DS2 source that has input and output variables.
Signature	Variables defined as inputs into a method and outputs from the execution of a method.
Input Signature	A description of the input values required to execute the step. The attributes of the input signature include the input variable, its data type, and the dimensions where applicable.
Output Signature	
	A description of the output values. Here is an example: the name value pairs that describe the name of the output variable, its data type, and the dimensions where applicable.
Module	
	A container of units of analytical code to be executed. For a DS2 source, it is a package.

Module ID	
	A generated unique string that identifies a module in an installation. When the installation is a cluster, no two modules created on two different cluster nodes have the same ID.
Module Name	
	A name associated with a module. For a DS2 source, this corresponds to the package name. A DS2 package name can be quoted. Because of that, it is not convenient to use it on the URL to specify the module for an HTTP operation. Even though the module name is not used to identify a module, each module name has to be unique in an installation.
Step	
-	A unit of analytical code to be executed. For a DS2 source, it is a method.
Step ID	
	The name of a step that is included in the micro analytic module. For a DS2 source, this corresponds to the name of a method. The combination of module ID and step ID is used to retrieve the individual step.
Source Code	
	The input analytic source code that is compiled into a micro analytic module containing one or more steps.

Client Application Features

Post Load or Create Modules

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

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

Get Input or Output Step Signatures

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

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

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

Post Validate Input Variables

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

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

Post Execute Modules

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

Put Update Modules

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

Delete Modules

The client application deletes a module through its module ID.

Security and Authentication

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

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

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

All modules are discoverable and usable by an authenticated user.

Life Cycle

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

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

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

Media Types

Externally Defined Media Types

application/vnd.sas.collection

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

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

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

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

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

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

Note: The order of the fields can vary.

application/vnd.sas.error

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

errorCode

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

httpStatusCode

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

message

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

details

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

remediation

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

version

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

links

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

application/vnd.sas.link

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

text/vnd.sas.source.ds2

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

SAS Micro Analytic Service Media Types

application/vnd.sas.microanalytic.module

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

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

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

Name	Туре	Description
version	integer	The media type's schema version number. This representation is version 1.
id	string	A generated unique string identifying a module in an installation.
description	string	Text describing the rules and logic performed by the module. The description is specified in the POST or PUT request body and carried over.
name	string	The name associated with the module.
creationTimeStamp	string	The formatted time stamp that tells when the module was initially created.
modifiedTimeStamp	string	The formatted time stamp that tells when the module was last revised.
revision	integer	The revision number of the module. It is a whole number starting from one and increases by one each time the module is revised.
scope	string (ENUM)	The scope restricts how a module can be used. There are two possible values:
		public The module is available to be called outside another module.
		private The module can be called only from within another module.
steps	array of string	An array of step IDs in the module.
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 63. 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.

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

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	Туре	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	агтау	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.

Name	Туре	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.

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.

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/ { <i>ModuleID</i> }
		Media type: application/vnd.sas.microanalytic.module

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

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

Name	Туре	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 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

Relationship	HTTP Method	Description
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	Туре	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	Туре	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	Туре	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:

```
},
   {
      "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
   }
1
```

}

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

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

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

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

Name	Туре	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
ир	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	Туре	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":
    {
        "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.

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

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

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

Name	Туре	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	Туре	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/
                 fldcdlaf-6ab2-4ac0-a5c6-5c64d5c09016",
         "uri":"/modules/fldcd1af-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/
                 fldcdlaf-6ab2-4ac0-a5c6-5c64d5c09016/source",
         "uri":"/modules/fldcd1af-6ab2-4ac0-a5c6-5c64d5c09016/source",
         "type":"application/vnd.sas.microanalytic.module.source"
      },
      {
         "method": "GET",
         "rel":"steps",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 fldcd1af-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/
                 fldcdlaf-6ab2-4ac0-a5c6-5c64d5c09016",
         "uri":"/modules/fldcd1af-6ab2-4ac0-a5c6-5c64d5c09016",
         "type": "application/vnd.sas.microanalytic.module"
      },
      {
         "method": "DELETE",
         "rel":"delete",
         "href": "http://www.example.com/SASMicroAnalyticService/rest/modules/
                 fldcdlaf-6ab2-4ac0-a5c6-5c64d5c09016",
         "uri":"/modules/fldcd1af-6ab2-4ac0-a5c6-5c64d5c09016"
```

```
}
  ],
   "description": "Module C",
   "version":1,
   "scope":"public",
   "id":"fldcdlaf-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; \ln \ln \text{ext}(2), in out double out array[2], in out double out array[2]); \ln
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
}</pre>
```

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, \ 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; \ln \ln \text{ext}(2), in out double out array[2], in out double out array[2]); \ln
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 charl2 = in charl2;\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 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 d_i \  n d_i \  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 d_i \  n d_i \  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	Туре	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
   }
1
       },
"links":[
   {
      "method":"GET",
      "rel":"self",
      "href": "http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy bigint",
      "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_bigint",
      "type":"application/vnd.sas.microanalytic.module.step"
   },
   {
      "method":"GET",
      "rel":"up",
      "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
      "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps",
      "type": "application/vnd.sas.collection"
   },
   {
      "method": "POST",
      "rel":"validate",
      "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
              36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy bigint/validations",
      "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_bigint/validations",
      "type":"application/vnd.sas.microanalytic.module.step.input.validity"
```

{

{

```
},
      {
         "method": "POST",
         "rel":"execute",
         "href":"http://www.example.com/SASMicroAnalyticService/rest/modules/
                 36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_bigint",
         "uri":"/modules/36af8e3c-6a37-4494-a8e0-9cc96ad62232/steps/copy_bigint",
         "type": "application/vnd.sas.microanalytic.module.step.output"
      }
   ],
   "id":"copy_bigint",
   "moduleId":"36af8e3c-6a37-4494-a8e0-9cc96ad62232",
   "inputs":[
      {
         "name":"in int",
         "type":"bigint",
         "dim":0,
         "size":0
      }
  ],
   "outputs":[
      {
         "name":"out int",
         "type":"bigint",
         "dim":0,
         "size":0
      }
   ]
},
   "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"
      }
   1,
   "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,
```

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",
```

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": 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
      1
    },
    {
      "name": "int_arr",
      "value": [
       7654321,
        7654321,
        7654321
     ]
    },
    {
      "name": "double_arr",
      "value": [
        0.9997,
        0.9997,
```

```
0.9997
]
],
"version": 1
}
```

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

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

Here is an example error response:

```
{
   "errorCode":-1958744015,
   "message":"Step ID echo_arrays failed to execute.",
   "details":[
        "Method not found."
   ],
        "remediation":"",
        "links":[],
        "version":1,
        "httpStatusCode":400
}
```

Chapter 8 Administration

SAS Micro Analytic Service Logging	123
Secure DS2 HTTP Package Usage	124
Monitoring	124
Monitoring SAS Micro Analytic Service	124
Monitoring SAS Micro Analytic Service Using SAS Environment Manager	125

SAS Micro Analytic Service Logging

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

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

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

App.tk.MAS.CodeGen is used for logging compiler-generated messages, such as compilation warnings and errors. Compiler messages can also be retrieved programmatically through the Java and REST interfaces. (See getCompilationMessages in "Method Descriptions" on page 41.) 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.

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 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. When the tcServer does not start up, check gemfire.log to see whether GemFire is waiting for data availability. Look for a log entry in a form that is similar to the following:

```
[info 2015/06/04 15:44:09.187 EDT <localhost-startStop-1> tid=0x15]
Region /sas_gemfire_region_surrogatekeytomodulereplica initialized with data
from /10.xx.xxx.yy:/data1/SAS-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
```

The wait will eventually time out, and SAS Micro Analytic Service will not start correctly. This is most likely to happen in a clustered environment. For more information, see "Cluster Deployment for SAS Micro Analytic Service" on page 132.

The REST service log file is located at SAS-configuration-directory/ LevN/Web/Logs/SASServer13_1/SASMicroAnalyticService1.3.log. The current day log entries are in that file. The first log entry that occurs after midnight causes the previous day's log file to roll over to another file with the format SASMicroAnalyticService1.3.log.yyyy-mm-dd. SASMicroAnalyticService1.3.log is created fresh with the first log entry. The service logs are at INFO level. Therefore, they capture start-up entries, module creation, update and deletion boundary entries, as well as errors from all operations. When there is an error, and more information must be captured to identify the cause of the error, update the REST service's Log4j configuration file to set logging level to DEBUG, and restart the service.

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

A related log file in the same directory is the SAS Micro Analytic Service log. The filename has the format SASMicroAnalyticService1.3MAS.log.yyyy-mm-dd.pid. Pid is the process ID of the JVM process that hosts SAS Micro Analytic Service. Each time the REST service restarts, a new log file is used and then the log file rolls over to another file at midnight. The SAS Micro Analytic Service log file can capture compilation errors of modules, as well as any anomaly that is encountered by the SAS Micro Analytic Service.

The application's Log4j configuration file is in the directory **SAS-configurationdirectory/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.
- 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**.
- 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** dropdown menu. Click **OK**.
- 5. You should receive two messages on the service window. The first should tell you that your service has been created. The second should ask you to set the configuration properties. Click **Configuration Properties** in the second message.
- 6. Under Configuration Properties, set the following:
 - a. Set the **port** field. The default is 7980.
 - b. Set the **hostname** field to the location where SAS Micro Analytic Service is installed.
 - c. Set the path field to /SASMicroAnalyticService.
 - d. Select GET from the method drop-down menu.
 - e. Click OK.

- 7. Click **Alert** and then **Configure**.
- 8. Click New.
- 9. Provide the information about the New Alert Definition window. Click **OK**.

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

Chapter 8 • Administration

Chapter 9 Deployment and Tuning

Pre-Installation Steps 1	.29
Deployment 1	30
Deploying SAS Micro Analytic Service 1	30
Adding Whitelist Websites to SAS Micro Analytic Service 1	31
Post-Installation Steps 1	31
Cluster Deployment for SAS Micro Analytic Service 1	.32
Deploying Clusters 1	32
License Files for Clusters 1	33
Tuning SAS Micro Analytic Service 1	.33
Adjust Thread Pool Size 1	33
Adjust Serial or Parallel Content Creation 1	34
Adjust DS2 Module Compilation Mode 1	34
Adjust Session Time-out Value	
Increase Module Execution Throughput of the REST Interface 1	
Prevent HTTP Error Messages 1	
Create and Update Connection Strings 1	

Pre-Installation Steps

Locate the file information that is listed in the table and then complete the preinstallation steps before running the SAS Deployment Wizard to install and configure SAS Micro Analytic Service 1.3.

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 the SAS Micro Analytic Service. The license location should be the fully qualified location and filename of the SAS installation data file. For more information, see "License Files for Clusters" on page 133.

SAS Micro Analytic Service 1.3 supports database data access. During configuration, you are prompted for database information depending on the database type 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 12.

130 Chapter 9 • Deployment and Tuning

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		
SAS Only		
SAS Library Name		
Path to SAS Data		
Oracle Only		
Oracle Default Schema Name	e	
SQL Server Only		
SQL Server ODBC Data Source Name		
Greenplum, PostgreSQL, DB	2, 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 tuned for performance and high availability. For information about how to tune the SAS Micro Analytic Service web application for optimum performance, see *SAS 9.4 Web Applications Tuning for Performance and Scalability*.

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 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 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"}]}
```

- When you have completed the validation steps found 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 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, then 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. The web server dispatches the requests in round-robin to the nodes in the cluster, unless a different policy is specified in the web server configuration.

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

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

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

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

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

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

EC5213E970F06558E63F145001F64CEC.c74b1b873e98ef08 505dee685863e7b2 SASServer13 1

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

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

Clustering relies on GemFire, a third-party distributed data management platform. GemFire persists data to files that are stored in SAS/config/LevN/Web/ WebAppServer/SASServer13_X/logs. The filenames contain the masgemfire substring. Those files should be left alone. Also, make sure that enough disk space is allocated to the SAS/config/LevN/Web/WebAppServer/SASServer13_X/logs directory so that the cache files grow.

CAUTION:

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

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

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 run applications 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.

Tuning SAS Micro Analytic Service

Adjust Thread Pool Size

Tasks in SAS Micro Analytic Service, such as revision compilations and method executions, are performed by special worker threads, which are part of the SAS threaded kernel architecture. These worker threads are maintained in a thread pool. The size of the thread pool to use is provided to SAS Micro Analytic Service as a start-up parameter. By default, the thread pool size is set to 4. Optimum performance is usually achieved by

134 Chapter 9 • Deployment and Tuning

setting the thread pool size about equal to the number of cores in the hosting server. However, the optimum setting might vary depending on the characteristics of the programs that are run by SAS Micro Analytic Service.

To change the worker thread pool size:

- 1. In SAS Management Console, expand Application Management.
- 2. Expand SAS Application Infrastructure.
- 3. Right-click SAS Micro Analytic Service 1.3.
- 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. They are taken in the order of arrival to the REST server's processing queue, one after another is done. To adjust this setting to allow them to be done in parallel:

- 1. In SAS Management Console, expand Application Management.
- 2. Expand SAS Application Infrastructure.
- 3. Right-click SAS Micro Analytic Service 1.3.
- 4. Select Properties.
- 5. Click the Advanced tab.
- 6. Unlock masintfc.tk.serializecontentcreation in the Property Name column.
- 7. Change the value. The choices are true and false. The default value is True.
- 8. Click OK.

Adjust DS2 Module Compilation Mode

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

- 1. In SAS Management Console, expand Application Management.
- 2. Expand SAS Application Infrastructure.
- 3. Right-click SAS Micro Analytic Service 1.3.
- 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 duration instead of fewer and longer garbage collection durations that might reduce the responsiveness of the REST server.

Increase Module ExecutionThroughput of the REST Interface

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

-Dsas.mas.access.mode=private

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

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

The place to edit the JVM option is host specific:

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

Prevent HTTP Error Messages

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

Create and Update Connection Strings

To create and update connection strings:

- 1. In SAS Management Console, expand Application Management ⇒ Configuration Management ⇒ SAS Application Infrastructure.
- 2. Right-click SAS Micro Analytic Service 1.3 and select Properties.
- 3. On the Advanced tab, update the masintfc.db.connectionstring property's value.
- 4. Click OK.

136 Chapter 9 • Deployment and Tuning

The connection string can contain a federation of multiple connection strings. For more information about federated connection strings, see "I/O" on page 12.

Chapter 10 Migration

Overview	137
Limitations	137
Back Up Disk Stores	138
Restore Script	138
Additional Migration Considerations	139
Migration Considerations for 64-Bit Windows	139
Additional Migration Considerations for 64-Bit HP-UX Itanium	140
Additional Migration Steps in a Clustered Environment	140

Overview

To upgrade from SAS Micro Analytic Service 1.2 to 1.3, you must migrate the modules that are loaded in the system. This requires first backing up a source system and then restoring that backup to a destination system. Because the modules are held in GemFire persistence, GemFire backup and restore procedures are used.

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 Migration Considerations" on page 139.

Limitations

Use the migration method described in this chapter to migrate across the same operating system. If you need to migrate between Windows and UNIX, contact your SAS Technical Support representative.

Back Up Disk Stores

To begin the migration 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 in 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 they have stopped any SASServer13 processes, 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 Migration Considerations

Migration 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\.
- 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-applicationcache=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 Migration 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.

- Locate the GemFire folder under SAS-Configuration-Directory/ LevN/Web/gemfire.
- 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 Migration 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.

142 Chapter 10 • Migration

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.

Attempt to retrieve internal use flag from nodule context %s while deletion pending. Attempt to retrieve revision %d creation time from module context %s while deletion pending.
rom module context %s while deletion
Attempt to retrieve compilation messages from nodule context %s while deletion pending.
Attempt to register name while deletion bending for module context %s.
Attempt to retrieve language of module context %s while deletion pending.
Attempt to retrieve display name from module context %s while deletion pending.
Attempt to retrieve C source code from module context %s revision %d while deletion pending.
Attempt to retrieve packages from user context %s while deletion pending.
Attempt to retrieve methods from module context %s while deletion pending.
Entry point not found.
Attempt to retrieve method %s signature from nodule context %s while deletion pending.
Private load out-of-the-box packages for user context %s while deletion pending.
Attempt to publish internal package %s to user context %s while deletion pending.
Attempt to remove internal package %s from user context %s while deletion pending.
Attempt to create garbage collection control atructures failed.
Garbage collection interval.
Garbage collection found assets ready to lelete.

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 shutdown.
-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.

Appendix 1 • SAS Micro Analytic Service Return Codes

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.

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 143 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	. 158
Understanding the Table Services Driver for DB2	158
Data Service Connection Options for DB2	. 159
DB2 Wire Protocol Driver Usage Notes	162
FedSQL Driver Reference	. 163
Overview	163
Connection Options	. 163
Greenplum Driver Reference	. 166
Understanding the Table Services Driver for Greenplum	. 166
Data Service Connection Options for Greenplum	. 166
Greenplum Wire Protocol Driver Usage Notes	
Netezza Driver Reference	170
Understanding the Table Services Driver for Netezza	
Data Service Connection Options for Netezza	. 171
ODBC Driver Reference	. 175
About ODBC	
Understanding the Table Services Driver for ODBC	. 175
Data Service Connection Options for ODBC	175
Wire Protocol Driver Usage Notes	. 181
Oracle Reference	182
Understanding the Table Services Driver for Oracle	
Data Service Connection Options for Oracle	. 182
Oracle Wire Protocol Driver Usage Notes	. 187
PostgreSQL Driver Reference	. 187
Understanding the SAS Federation Server Driver for PostgreSQL	. 187
Data Service Connection Options for PostgreSQL	. 188
SAS Data Set Reference	. 192
Overview	192
Understanding the Driver for Base SAS	. 192
Data Service Connection Options for SAS Data Sets	. 192
Teradata Reference	. 196
Understanding the Table Services Driver for Teradata	
Data Service Connection Options for Teradata	. 196

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.

```
ATX
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.
	<i>Note:</i> The FedSQL language processor automatically quotes SQL identifiers that do not meet the regular naming convention as defined in <i>SAS FedSQL Reference Guide</i> .
DATABASE DB	DATABASE=database-specification;
	Specifies the name of the DB2 database (for example, database=sample, DB=sample).
	<i>Note:</i> You must specify a database name.
DRIVER	DRIVER=DB2;
	Identifies the DB2 data source to which you want to connect.
	<i>Note:</i> 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_ENCODIN	CLIENT_ENCODING=encoding-value
G	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.
	<i>Note:</i> 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.
	<i>Note:</i> 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= <i>n</i> - 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= <i>n</i> - 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.		
	<i>Note:</i> 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_TRACEFIL	DRIVER_TRACEFILE='filename';
Е	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_TRACEOP	DRIVER_TRACEOPTIONS=APPEND THREADSTAMP TIMESTAMP;
TIONS	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.
	<i>Note:</i> SAS Federation Server automatically quotes SQL identifiers that do not meet the regular naming convention as defined in <i>SAS FedSQL Reference Guide</i> .

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_UNQUOTE	ALLOW_UNQUOTED_NAMES=NO YES;
D_NAMES	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=(<i>attr=value</i> ;)
	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= <i>n</i> - 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= <i>n</i> - 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.
	<i>Note:</i> 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_TRACEFIL E	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_TRACEOPT	DRIVER_TRACEOPTIONS=APPEND THREADSTAMP TIMESTAMP;
IONS	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	NUMBYTESPERCHAR=value;
CHAR	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=.
	<i>Note:</i> 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=.
	<i>Note:</i> 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.
	<i>Note:</i> Table services automatically quotes SQL identifiers that do not meet the regular naming convention as defined in <i>SAS FedSQL Reference Guide</i> .
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= <i>NETEZZA</i> ;
	Specifies the data service for the Netezza database to which you want to connect.
	<i>Note:</i> 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_ENCODIN G	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= <i>n</i> - 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= <i>n</i> - 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.
	 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
	 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.
	 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
	 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.
	 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

DRIVER_TRACEFI DRIVER_TRACEFILE='filename'; E 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)'\uptrace.log'). Default: The default TRACEFILE location applies to a relative filename, and it is placed relative to TRACEFILEPATH. Requirement: DRIVER_TRACE Interaction: (Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS DRIVER_TRACEO 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 thread identification. TIMESTAMP Prepends each line of the trace log with a thread identification. TIMESTAMP Prepends each line of the trace log with a thread identification. TIMESTAMP Prepends each line of the trace log with a thread identification. TOWER_TRACEOPTIONS-APPEND THREADSTAMP TIMESTAMP. 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 PASSWORD option with USER. STRIP_BLANKS STRIP_BLANKS=YES NO; Specifies whether to optice to the Netezza database in Re	Option	Description
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_TRACEOPTIONS DRIVER_TRACEOP DRIVER_TRACEOPTIONS-APPEND THREADSTAMP TIMESTAMP; TIONS PREDENDAdds trace informating 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. - THREADSTAMP Prepends each line of the trace log with a time stamp. Default: The trace log is overwritten with no thread identification or time stamp. Default: The trace log is overwritten with no thread identification or time stamp. USER USER= "user-id"; 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 enneet to the Netazza database in Read-Only mode. The default is NO. Alias: READ_ONLY=YES NO; Specifies whether to conneet to the Netazza database in Read-Only mode. The default is NO. Alias: READ_ONLY SHOWSYSTEMTAB SHOWSYSTEMTABLE	—	DRIVER_TRACEFILE='filename';
relative to TRACEFILEPATH. Requirement: DRIVER_TRACE. Interaction: (Optional) You can control trace log formatting with DRIVER_TRACE. DRIVER_TRACEOP DRIVER_TRACEOPTIONS= DEFault: The trace log inverters Specifies a Netezza user ID. If the ID contains blanks or national characters, enclose it in quotation marks. Alias: UD. No		single or double quotation marks (for example,
DRIVER_TRACE. Interaction: (Optional) You can control trace log formatting with DRIVER_TRACEOP DRIVER_TRACEOPTIONS=. DRIVER_TRACEOPTIONS TIONS DRIVER_TRACEOPTIONS=. USER USER DSPecifies a Netzza user ID. If the ID contains blanks or national characters, enclose it in quotation marks. Alias: UID. <td></td> <td></td>		
DRIVER_TRACEOPTIONS=. DRIVER_TRACEOP DRIVER_TRACEOPTIONS=APPEND THREADSTAMP TIMESTAMP; TIONS 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 time stamp. Default: The trace log is overwritten with no thread identification or time stamp. USER USER="user-id"; Specifies a Netzza 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. 		
TIONS 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 READONLY READONLY SHOWSYSTEMTABLES=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 NUMBERBYTESPE NUMBYTESPERCHAR=v		
Specifies options in order to control formating 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. • THREADSTAMP Prepends each line of the trace log with a time stamp. Default: The trace log is overwritten with no thread identification or 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; ShowSYSTEMTAB 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 NUMBERBYTESPE NUMBYTESPERCHAR=value;		DRIVER_TRACEOPTIONS=APPEND THREADSTAMP TIMESTAMP;
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.USERUSER="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.PASSWORDPASSWORD=password; Specifies a password for the ID passed through the USER= option. Alias: PWD. Note: You must specify the PASSWORD option with USER.STRIP_BLANKSSTRIP_BLANKS=YES NO; Specifies whether to strip blanks from character fields.READONLYREADONLY=YES NO; Specifies whether to connect to the Netezza database in Read-Only mode. The default is NO. Alias: READ_ONLYSHOWSYSTEMTAB LESSHOWSYSTEMTABLES=YES NO; Specifies whether tables are included in the available table list. The default setting is NO. Alias: SSTNUMBERBYTESSPE NUMBERBYTESSPENUMBYTESPERCHAR=value;	HONS	Specifies options in order to control formatting and other properties for the trace log:
• 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 ShOwSYSTEMTAB LES ShOwSYSTEMTAB NUMBERBYTESPE NUMBYTESPERCHAR=value;		
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 SHOWSYSTEMTAB LES 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 NUMBERBYTESPE PCHARACCTER NUMBYTESPERCHAR=value;		• THREADSTAMP Prepends each line of the trace log with a thread identification.
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; SHOWSYSTEMTAB SHOWSYSTEMTABLES=YES NO; SHOWSYSTEMTAB Showsystemtables 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 NUMBERBYTESPE NUMBYTESPERCHAR=value;		• TIMESTAMP Prepends each line of the trace log with a time stamp.
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 SHOWSYSTEMTAB SHOWSYSTEMTAB 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. If set to YES or TRUE, system tables are included in the available table list. SST NUMBERBYTESPE NUMBYTESPERCHAR=value;		Default: The trace log is overwritten with no thread identification or time stamp.
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 SHOWSYSTEMTAB LES 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 NUMBERBYTESPE NUMBYTESPERCHAR=value;	USER	USER="user-id";
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 SHOWSYSTEMTAB LES 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 NUMBERBYTESPE NUMBYTESPERCHAR=value;		
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 SHOWSYSTEMTAB LES ShowSYSTEMTAB ShowSYSTEMTAB Pecifies 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 NUMBERBYTESPE NUMBYTESPERCHAR=value;		Note: You must specify the USER option.
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 SHOWSYSTEMTAB LES 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 NUMBERBYTESPE NUMBYTESPERCHAR=value;	PASSWORD	PASSWORD=password;
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 SHOWSYSTEMTAB SHOWSYSTEMTABLES=YES NO; LES 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 NUMBERBYTESPE NUMBYTESPERCHAR=value;		Specifies a password for the ID passed through the USER= option. Alias: PWD.
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 SHOWSYSTEMTAB SHOWSYSTEMTAB SHOWSYSTEMTAB SHOWSYSTEMTAB SHOWSYSTEMTAB SHOWSYSTEMTAB 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 NUMBERBYTESPE NUMBYTESPERCHAR=value;		Note: You must specify the PASSWORD option with USER.
READONLY READONLY=YES NO; Specifies whether to connect to the Netezza database in Read-Only mode. The default is NO. Alias: READ_ONLY SHOWSYSTEMTAB SHOWSYSTEMTABLES=YES NO; LES 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 NUMBERBYTESPE NUMBYTESPERCHAR=value;	STRIP_BLANKS	STRIP_BLANKS=YES NO;
Specifies whether to connect to the Netezza database in Read-Only mode. The default is NO. Alias: READ_ONLY SHOWSYSTEMTAB SHOWSYSTEMTAB SHOWSYSTEMTAB 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 NUMBERBYTESPE NUMBYTESPERCHAR=value;		Specifies whether to strip blanks from character fields.
Alias: READ_ONLY SHOWSYSTEMTAB LES 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 NUMBERBYTESPE NUMBYTESPERCHAR=value;	READONLY	READONLY=YES NO;
LES 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 NUMBERBYTESPE NUMBYTESPERCHAR=value;		
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 NUMBERBYTESPE NUMBERBYTESPE NUMBERBYTESPE		SHOWSYSTEMTABLES=YES NO;
PCHARACTER	LES	
RCHARACTER Specifies the default number of bytes per character.		NUMBYTESPERCHAR=value;
	RCHARACTER	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 name d newusers, use the following command: catalog=(mycat=newusers); Catalog name maps can be used only with FedSQL. They are not valid with native SQL.
	<i>Note:</i> The FedSQL language processor automatically quotes SQL identifiers that do not meet the regular naming convention as defined in <i>SAS FedSQL Reference Guide</i> .
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:
	<pre>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))</pre>
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= <i>odbc</i> 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_CURSO R_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.
	<i>Note:</i> 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_TRACEFI	DRIVER_TRACEFILE='filename';
LE	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_TRACEO	DRIVER_TRACEOPTIONS=APPEND THREADSTAMP TIMESTAMP;
PTIONS	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.
	<i>Note:</i> 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:

 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/l1g; export ORACLE_HOME
SASORA=V9; export SASORA
PATH=$ORACLE_HOME/bin:/bin:/usr/ccs/
bin:/opt/bin:$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.
	<i>Note:</i> The FedSQL language processor automatically quotes SQL identifiers that do not meet the regular naming convention as defined in <i>SAS FedSQL Reference Guide</i> .
DRIVER	DRIVER=ORACLE;
	Identifies the data service to which you want to connect, which is an Oracle database.
	<i>Note:</i> You must specify the driver.
РАТН	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= <i>n</i> - 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= <i>n</i> - 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.
	<i>Note:</i> 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.

Description
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=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=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.
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=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:
 Create a materialized view. See the example code in "Creating a Materialized View (USE_CACHED_CATALOG)" on page 186.
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, FKACC.COLUMN_NAME as FKTABLE_NAME, FKACC.COLUMN_NAME as FKCOLUMN_NAME, FKACC.COLUMN_NAME as FKCOLUMN_NAME, FKACC.CONSTRAINT_NAME as FK_NAME, PKAC.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.cowner=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 configuraton 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).
	<i>Note:</i> The FedSQL language processor automatically quotes SQL identifiers that do not meet the regular naming convention as defined in <i>SAS FedSQL Reference Guide</i> .
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:
	<pre>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))"</pre>

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_UNQUOTE D_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_ENCODIN G	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= <i>n</i> - 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= <i>n</i> - 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.
	<i>Note:</i> 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_TRACEFIL	DRIVER_TRACEFILE='filename';
E	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_TRACEOP	DRIVER_TRACEOPTIONS=APPEND THREADSTAMP TIMESTAMP;
TIONS	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.
	<i>Note:</i> The FedSQL language processor automatically quotes SQL identifiers that do not meet th regular naming convention as defined in <i>SAS FedSQL Reference Guide</i> .
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.
	<i>Note:</i> You must specify a primary path.
SCHEMA	SCHEMA=(attributes);
(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= <i>n</i> - 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= <i>n</i> - 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.
	<i>Note:</i> 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.
	<i>Note:</i> 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_ENCODIN G	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_OU
	LATIN9_0A THAI874 4A0
	LATIN1250 1A0
	CYRILLIC1251 2A0
	 LATIN1254 7A0
	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
	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= <i>n</i> - 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= <i>n</i> - 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.
	<i>Note:</i> 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_TRACEFI	DRIVER TRACEFILE= `filename';
—	2.1. 2 /
LE	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').
—	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,
—	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
—	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
—	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.
LE	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=.
LE DRIVER_TRACEO	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=APPEND THREADSTAMP TIMESTAMP;
LE DRIVER_TRACEO	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=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
LE DRIVER_TRACEO	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=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.
LE DRIVER_TRACEO	 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=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.
LE DRIVER_TRACEO	 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=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 time stamp.
LE DRIVER_TRACEO PTIONS	 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=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.

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=.
	<i>Note:</i> 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

202 Recommended Reading

Index

Numbers

32-bit wire protocol driver 187

Α

administration logging 123 argument types 17

В

back up disk stores 138 business context 3

С

character-to-numeric conversions 27 configuring Python 30

D

deploying 130 DS2 best practices 23, 24, 27, 28 DS2 programming 11, 12, 13, 14, 17

G

global packages 23 Greenplum 64-bit wire protocol driver 170

Н

hash package 27

I

interfaces 5 invariant computations 28

J

Java business context methods 35 Java interface topology 34 Java POJO 33 execution methods 45 module context methods 37 revision methods 40 shutdown 35 start-up 34 user context methods 35

L

local packages 23

Μ

migration 138 module 4

0

ODBC 175 SQL server legacy wire protocol driver 181 Oracle 32-bit wire protocol driver 187 64-bit wire protocol driver 187

Ρ

passing character values to methods 27 pre-installation 129 private methods 14 programming blocks 13 public methods 14 public packages 14 publishing DS2 source code 11 Python 29, 30 Python support 29

R

REST API 58 revision 4

S

SAS table driver 192 SAS Micro Analytic Service concepts 3, 4, 5 SCAN 24 single computation 28 SQL server legacy wire protocol driver 181 tuning adjusting thread pool size 133

U

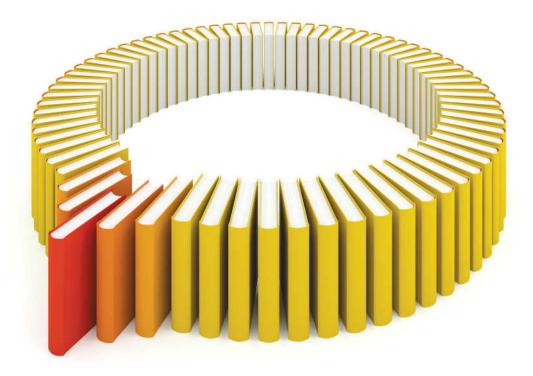
user context 3

W

wire protocol driver 187

Т

TRANWRD 24



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

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





SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries, @ indicates USA registration. Other brand and product names are trademarks of their respective companies. © 2013 SAS Institute Inc. All rights reserved. S107969US.0613