PART 1  Understanding SAS Micro Analytic Services  

Chapter 1 • Introduction to SAS Micro Analytic Service  
What Is SAS Micro Analytic Service?  3
What Is SAS Event Stream Processing?  4
What Is SAS Decision Manager?  5

Chapter 2 • Concepts  
Overview  7
User or Business Context  8
Module Context  8
Revision  9
Architecture  9

PART 2  Using SAS Micro Analytic Services with SAS Event Stream Processing  

Chapter 3 • Publishing to SAS Micro Analytic Service in SAS Event Stream Processing  
Overview  13
Object Hierarchy  14
C++ Methods  14
C++ Example  17
XML Example  17
Data Type Mappings  20
Generating Multiple Derived Events from a Single Source Event  22

Chapter 4 • Processing Event Opcodes and Flags  
Operation Codes and Flags  25
DS2 Opcodes Example  26

Chapter 5 • DS2 Programming for SAS Micro Analytic Service  
Overview  27
DS2 Source Code Prerequisites  28
SAS Micro Analytic Service and SAS Foundation  28
Programming Blocks  28
Public and Private Methods and Packages  29
Argument Types Supported in Public Methods  32
Using ASTORE Models  33
### Contents

**Chapter 6 • Best Practices for DS2 Programming in SAS Event Stream Processing**  
Overview ................................................. 37  
Return Results ............................................. 37  
Global Packages versus Local Packages .......................... 38  
Replacing SCAN (and TRANWRD) with DS2 Code ............... 39  
Hash Package ................................................ 41  
Character-to-Numeric Conversions ............................... 41  
Passing Character Values to Methods ........................... 41  
Performing the Computation Once .............................. 42  
Moving Invariant Computations Out of Loops .................... 42

**Chapter 7 • Python Support in SAS Micro Analytic Service**  
Introduction .............................................. 43  
Public and Private Methods ................................... 44  
Example ...................................................... 44  
Configuring Python for SAS Event Stream Processing .......... 46

**Chapter 8 • SAS Micro Analytic Service Logging and Deployment**  
SAS Micro Analytic Service Logging .......................... 51  
Deployment ................................................. 52

**PART 3 Using SAS Micro Analytic Services with SAS Decision Manager**  
**Chapter 9 • DS2 Programming for SAS Micro Analytic Service**  
Overview .................................................. 55  
DS2 Source Code Prerequisites ................................ 55  
SAS Micro Analytic Service and SAS Foundation ............... 56  
I/O .......................................................... 57  
Programming Blocks ....................................... 59  
Public and Private Methods and Packages ....................... 59  
Argument Types Supported in Public Methods ................... 62

**Chapter 10 • Best Practices for DS2 Programming in SAS Decision Manager**  
Overview ................................................... 65  
Return Results ............................................. 65  
Global Packages versus Local Packages ......................... 66  
Replacing SCAN (and TRANWRD) with DS2 Code ............... 67  
Hash Package ................................................ 69  
Character-to-Numeric Conversions ............................... 69  
Passing Character Values to Methods ........................... 69  
Performing the Computation Once .............................. 70  
Moving Invariant Computations Out of Loops .................... 70

**Chapter 11 • Administration**  
SAS Micro Analytic Service Administration ..................... 71  
Starting and Stopping SAS Micro Analytic Service .............. 71  
SAS Micro Analytic Service Configuration ...................... 72  
SAS Micro Analytic Service Logging .......................... 75  
SAS Micro Analytic Service Security and Authorization .......... 77  
Secure DS2 HTTP Package Usage ................................ 78
About This Book

Audience

This guide is intended for developers and information technology administrators.

SAS Decision Manager and SAS Event Stream Processing include SAS Micro Analytic Service as an execution engine. SAS DS2 or Python code can be used to process events through SAS Event Stream Processing or SAS Decision Manager, the latter of which embeds SAS Micro Analytic Service as an internal component.

Developers can use this guide as a resource to author SAS analytics, business logic, and user-written programs for use with SAS Decision Manager and SAS Event Stream Processing. Included in this guide is information about how SAS Micro Analytic Service processes events, as well as tips, best practices, and restrictions on programming DS2 or Python to run in SAS Micro Analytic Service.

Technology administrators can find information about how to configure SAS Micro Analytic Service. Also included is information about how to configure Anaconda Python and SAS Micro Analytic Service to run Python code (optional) or how to configure them to run them inside SAS Event Stream Processing.
About This Book
What’s New in SAS Micro Analytic Service 5.1

Overview

SAS Micro Analytic Service is a memory-resident, high-performance program execution service that is included in selected SAS solutions. It provides hosting for DS2 and Python programs and supports a “compile-once, execute-many-times” usage pattern. In addition to providing generated code, SAS Micro Analytic Service enables you to author DS2 or Python code that is customized to your specific needs.

SAS Micro Analytic Service 5.1 includes the following enhancements:

• Support for ASTORE publishing and execution
• Addition of the composite module type
• SAS Event Stream Processing enhancements
• DS2 Package enhancements
• Documentation enhancements

Support for ASTORE Publishing and Execution

SAS Micro Analytic Service now supports analytic store, or ASTORE, publishing and execution via DS2. An ASTORE is a system that allows the state of a trained predictive model to be saved in a transportable form.

Addition of the Composite Modules Type

SAS Micro Analytic Service now supports composite modules. Composite modules enable DS2 code running in SAS Micro Analytic Service to call one or more ASTORE models.
SAS Event Stream Processing Enhancements

SAS Micro Analytic Service now supports two new SAS Event Stream Processing data types—rstring and blob, and three new array types—32-bit integer, 64-bit integer, and double.

In addition, there is now SAS Event Stream Processing schema validation and improved error messages.

DS2 Package Enhancements

SAS Micro Analytic Service now supports the ability to publish, per revision of a given module, DS2 packages with different names.

Documentation Enhancements

Beginning with SAS Micro Analytic Services 5.1, all content is included in a single guide, replacing the two versions of *SAS Micro Analytic Service: Programming and Administration Guide*—one that was specific to SAS Event Stream Processing, and one that was specific to SAS Decision Manager.
For information about the accessibility of any of the products mentioned in this document, see the usage documentation for that product.
xii  What's New in SAS Micro Analytic Service 5.1
Part 1

Understanding SAS Micro Analytic Services

Chapter 1
Introduction to SAS Micro Analytic Service ............................................. 3

Chapter 2
Concepts ................................................................................................. 7
Chapter 1
Introduction to SAS Micro Analytic Service

What Is SAS Micro Analytic Service?

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

SAS Micro Analytic Service contains a core engine that is written in C for high performance and, when deployed as part of SAS Event Stream Processing, includes C++ classes that integrate with SAS Event Stream Processing. These capabilities allow both to execute within the same process space for maximum performance. The combination of SAS Event Stream Processing and SAS Micro Analytic Service enables SAS analytics, business logic, and user-written programs to operate on streams of data in motion.

When deployed as part of SAS Decision Manager, SAS Micro Analytic Service is called as a web application with a REST interface by both SAS Decision Manager and by other client applications. The REST interface provides easy integration with client applications, and adds persistence and clustering for scalability and high availability.

Users of SAS Event Stream Processing or SAS Decision Manager can publish SAS analytics, such as predictive models that were created with a variety of SAS products and analytical procedures. They can also author custom programs using the SAS DS2 or Python programming languages or in the SAS Decision Manager web application. The custom programs execute inside SAS Event Stream Processing applications. SAS Micro Analytic Service can host multiple programs simultaneously.

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

SAS Micro Analytic Service supports the Anaconda distribution of Python 3.4 and 2.7. Python programs that are written for SAS Micro Analytic Service might include custom functions, and they can use any third-party Python packages that have been deployed to a local Anaconda Python environment.

What Is SAS Event Stream Processing?

Event stream processing is a form of complex event processing technology that is often used in data and decision applications. It analyzes and processes large volumes of streaming data quickly, helping you analyze events in motion—even as they are generated. Instead of storing data and running queries against it, the SAS Event Stream Processing Engine stores the queries and streams data through them. In this way, continuous analysis of data occurs as it is received, updating the intelligence as new events occur. Incoming data is read through adapters that are part of a publish-and-subscribe architecture used to read data feeds. Objects written in XML, DS2, Python, and C++ are used to model business logic within the engine. Data that passes through the engine is examined for patterns and can be filtered to more permanent storage. Prebuilt high-performance adapters publish filtered data to other downstream applications. The SAS Event Stream Processing Engine uses a flexible threaded processing model for submillisecond response and high-volume throughput.

The SAS Event Stream Processing Engine uses data flow models to define an ordering of source and derived event windows; the former defines the schema of event streams flowing into the system while the latter determines how these incoming event streams are processed. These data flow models are directed graphs that are frequently referred to as continuous queries, and they are the defining characteristic of event stream processing systems. The reason these models are referred to as continuous queries is that event stream processing systems use a set of known (or modeled) queries of interest. Also, they continuously update the resultant set of those queries (also known as derived windows) as new events are published to the system. Applications or end users can subscribe to any set of windows (for which they are authorized). Window events can also be queried through SQL in either an ad hoc or on-demand manner.

The SAS Event Stream Processing Engine is event driven- and data flow-centric in that event streams are continuously published to the system and processed through a set of data flow models (or continuous queries). As events are absorbed into the platform and processed though the continuous queries, windows (or nodes in the data flow model) can be queried or subscribed to. Each derived window in the graph is defined by relational operators (for example, join, aggregate, or filter), pattern matching, or procedural processing (for example, an extension to relational processing and pattern matching). However, this is not standard for all event stream processes.

SAS Micro Analytic Service executes inside the event stream processing procedural window, enabling continuous queries to process events with SAS analytics and custom logic that is written in DS2 or Python.
What Is SAS Decision Manager?

Enterprise decision management systems can enable businesses to use the information they already have to make better decisions—decisions that are based on predictive analytics rather than on past history. Decision management systems automate the process of making decisions, particularly day-to-day operational decisions. They improve the speed, efficiency, and accuracy of routine business processes, in part by reducing the need for human intervention. By automating decisions, organizations in any industry can improve interactions with customers, partners, suppliers, and employees. In addition, organizations that are highly regulated, such as financial services, health care, and insurance, can more easily achieve compliance as a result of repeatable, traceable decisions.

SAS Decision Manager helps organizations manage data, business rules, analytical models, and optimization techniques. Rule management, model management, and data management are integrated into a consistent interface for easier accessibility.

SAS Decision Manager enables you to manage your list of data tables from within the application. You can create new Base SAS libraries, add and remove tables, view table data and metadata, create and delete table summaries, and associate attachments and comments with tables. The application uses data tables when it needs to access data (for example, for rule discovery, rule flow testing, and for the testing, scoring, training, and performance monitoring of models).

Business rules capture the logic of business decisions and are one of the core components of decision management systems. Business rules make the decision-making process transparent and adaptable, allowing organizations to respond quickly to new information about customers and markets. They allow organizations to identify and deal with fraud, avoid unnecessary risk, and find opportunities hidden in customer data. You can use SAS Decision Manager to create a database of business rules, connect those rules together into rules flows, and publish the rule flows for use by other applications.

Using SAS Decision Manager, you can do all of the following with models:

- store models and organize them within projects or folders
- validate candidate models
- assess candidate models for champion model selection
- publish and monitor champion models in a production environment
- retrain models

All model development and model maintenance personnel, including data modelers, validation testers, scoring officers, and analysts, can use SAS Decision Manager.

SAS Decision Manager includes Decision Builder. Decision Builder enables you to combine analytical models, rule flows, and conditional logic into decisions. You can investigate various scenarios, test and refine the decision logic, and then publish the decisions for use in batch applications and online transactions. You can automate the decision-making process by operationalizing your analytics and business rules. Automating decisions with Decision Builder provides a streamlined mechanism for controlling and monitoring the rules and processes used by your organization.
Overview

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

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

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

For SAS Event Stream Processing, module methods can be used to process events in a continuous query. The results of such processing create derived events that flow to downstream components in the continuous query. In addition to generating derived events, module methods can influence SAS Event Stream Processing by interrogating and setting event opcodes and flags. Event opcodes and flags are covered in more detail in the DS2 and Python programming chapters that follow. Also, for more information about SAS Event Stream Processing, see the product documentation at http://support.sas.com.

For SAS Decision Manager, module methods can be used to automate data-driven decisions. This is accomplished by executing analytical models and business rules against the latest data from online channels, combined with data from operational databases and other data sources.

SAS Micro Analytic Service uses two internal component types to manage the modules that are published to it. These are the module context and the revision. A third component, the user context, provides isolated execution environments that contain sets of module contexts and revisions. In most cases, SAS Micro Analytic Service automatically manages user and module contexts for the user.
SAS Micro Analytic Service automatically manages user contexts for SAS Event Stream Processing by maintaining one user context per event stream processing object. Therefore, unlike previous versions of SAS Micro Analytic Service, user contexts do not appear in the user interfaces.

SAS Decision Manager automatically creates and manages one user context per tenant. Therefore, user contexts do not appear in the SAS Decision Manager user interfaces.

Note the following information before writing modules to deploy to SAS Micro Analytic Service:

- If you plan to write DS2 modules, depending on your environment, see the programming guidelines described in one of the following locations:
  - Decision Manager: Chapter 9, “DS2 Programming for SAS Micro Analytic Service,” on page 55
- If you plan to write Python modules, see the programming guidelines in Chapter 7, “Python Support in SAS Micro Analytic Service,” on page 43.

---

**User or Business Context**

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

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

---

**Module Context**

A module represents program code. In the case of DS2, each module represents exactly one DS2 package. If you are unfamiliar with DS2 packages, see “Understanding DS2 Methods and Packages” in *SAS Viya: DS2 Language Reference*. Every module is owned by exactly one user context.

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

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

*Note:* SAS Micro Analytic Service 5.1 runs the latest revision of a module by default.
Revision

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

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

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

Note: When modules are published to SAS Micro Analytic Service by SAS Event Stream Processing, only the latest revision is retained.

Architecture

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

*Note:* SAS Event Stream Processing integrates with SAS Micro Analytic Service via a private C interface to the core engine. This interface allows SAS Event Stream Processing to call SAS Micro Analytic Service directly, in-process, for maximum performance. When SAS Micro Analytic Service is deployed with SAS Event Stream Processing, Java and REST/JSON layers are omitted.

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

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

*Note:* SAS Decision Manager interfaces with SAS Micro Analytic Service via REST/JSON, and uses all three layers of the architecture.
Part 2

Using SAS Micro Analytic Services with SAS Event Stream Processing

Chapter 3
Publishing to SAS Micro Analytic Service in SAS Event Stream Processing ........................................... 13

Chapter 4
Processing Event Opcodes and Flags ........................................... 25

Chapter 5
DS2 Programming for SAS Micro Analytic Service ............... 27

Chapter 6
Best Practices for DS2 Programming in SAS Event Stream Processing ........................................ 37

Chapter 7
Python Support in SAS Micro Analytic Service ................. 43

Chapter 8
SAS Micro Analytic Service Logging and Deployment ........ 51
Chapter 3
Publishing to SAS Micro Analytic Service in SAS Event Stream Processing

Overview

The SAS Event Stream Processing Engine uses data flow models to define an ordering of source and derived event windows. The former defines the schema of event streams flowing into the system, and the latter determines how these incoming event streams are processed. For more information, see the SAS Event Stream Processing documentation at http://support.sas.com.

The derived event window in which SAS Micro Analytic Service operates is the procedural window. Within a continuous query application, procedural windows might be configured to receive events from one or more source windows. These source events can be processed by SAS Micro Analytic Service, which might generate zero or more derived events. These events, in turn, can be subscribed to by downstream windows.

There are two ways to define a SAS Event Stream Processing continuous query: either by creating a continuous query XML definition or by coding a continuous query in C++. Both enable you to publish DS2 and Python modules to SAS Micro Analytic Service. They also specify which of a module’s methods are used to process each input source window’s events.
SAS Event Stream Processing provides an assortment of elements for use in building continuous query applications, enabling a wide variety of business needs to be met. Some of the elements that incorporate SAS Micro Analytic Service functionality are described briefly below. For more information, see the SAS Event Stream Processing documentation at [http://support.sas.com](http://support.sas.com).

At a minimum, each event stream processing application consists of a SAS Event Stream Processing Engine (a container of projects), at least one project (a container of continuous queries), and at least one continuous query that contains windows and the edges that connect them.

Each project maintains a SAS Micro Analytic Service environment that is shared among the continuous queries within the project. In the C++ case, DS2 and Python modules are published to SAS Micro Analytic Service by calling methods on the project. In the XML case, they are published by including the `<mas-modules>` tag (a sub-element of `<project>`).

SAS Micro Analytic Service operates on the data that is contained in events received by procedural windows. The procedural window provides a C++ method and an XML element for mapping source window events to module methods. The XML tag is named `<mas-map>`, which is a sub-element of `<window-procedural>`.

### C++ Methods

C++ continuous query authors can use the following methods for publishing, replacing, or removing DS2 and Python modules and for mapping source window events to module methods.

```cpp
dfESPproject::publishToMAS(    const char *name,    language_t lang,    const char *sourceCode,    const char *description)
```

dfESPproject::publishToMAS() publishes a DS2 or Python module to SAS Micro Analytic Service in SAS Event Stream Processing, where the source code to publish is in a string.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the module. It is used to refer to the module in any subsequent method calls.</td>
</tr>
<tr>
<td>language</td>
<td>Specify either DS2 or Python.</td>
</tr>
</tbody>
</table>
dfESPProject::publishFileToMAS() publishes a DS2 or Python module to SAS Micro Analytic Service, where the source code is to be read from a text file.

**Parameter** | **Description**
--- | ---
description | An optional string that contains a description of the module’s purpose. Pass in NULL to omit a description.

```
dfESPProject::publishFileToMAS(    const char *name,
    language_t lang,
    const char *filePath,
    const char *description)
```

dfESPProject::replaceMASmodule() replaces an existing DS2 or Python module in SAS Micro Analytic Service, where the replacement source code to publish is in a string. If the module to replace, indicated by the name parameter, does not exist, then replaceMASmodule() does not publish the source code.

**Parameter** | **Description**
--- | ---
name | The name of the module. It is used to refer to the module in any subsequent method calls.
language | Specify either DS2 or Python.
description | An optional string that contains a description of the module’s purpose. Pass in NULL to omit a description.

```
dfESPProject::replaceMASmodule(    const char *name,
    language_t lang,
    const char *sourceCode,
    const char *description)
```
### dfESPproject::replaceMASmoduleFromFile()

replaceMASmoduleFromFile() replaces an existing DS2 or Python module in SAS Micro Analytic Service, where the source code is to be read from a text file. If the module to replace, indicated by the name parameter, does not exist, then replaceMASmoduleFromFile() does not publish the source code.

#### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the module. It is used to refer to the module in any subsequent method calls.</td>
</tr>
<tr>
<td>language</td>
<td>Specify either DS2 or Python.</td>
</tr>
<tr>
<td>filePath</td>
<td>The path to the file containing the DS2 or Python source code with which to replace the existing code, including filename and extension. See Chapter 5, “DS2 Programming for SAS Micro Analytic Service,” and Chapter 7, “Python Support in SAS Micro Analytic Service,” for Micro Analytic Service in SAS Event Stream Processing.</td>
</tr>
<tr>
<td>description</td>
<td>An optional string that contains a description of the module’s purpose. Pass in NULL to omit a description.</td>
</tr>
</tbody>
</table>

### dfESPproject::deleteMASmodule()

deleteMASmodule() removes the specified module from SAS Micro Analytic Service. The *name parameter is the name of the module to delete.

### dfESPpcontext::registerMethod_MAS()

registerMethod_MAS() causes the specified source window’s events to be processed by the specified module method. The module containing the method must have been published to SAS Micro Analytic Service before calling registerMethod_MAS().

#### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>The source window whose events are to be processed by the specified module method.</td>
</tr>
<tr>
<td>moduleName</td>
<td>The name of the module containing the specified method.</td>
</tr>
<tr>
<td>methodName</td>
<td>The name of the method to use to process the specified source window’s events.</td>
</tr>
</tbody>
</table>
C++ Example

Here is an example C++ continuous query. SAS Micro Analytic Service related method calls are highlighted. Error checking code has been removed to emphasize the main elements of the program, but should always be included in production.

```cpp
// Initialize the SAS Event Stream Processing
// Engine and create a project.
dfESPengine *myEngine = dfESPengine::initialize(argc, argv, "engine", pubsub_DISABLE);
dfESPproject *project_01 = myEngine->newProject("project_01");

// Publish a DS2 module to SAS Micro Analytic Service,
// in this case a credit risk scoring model.
project_01->publishFileToMAS("credit_risk", DS2,
   "./credit_risk.ds2", "Credit Risk Model");

// Create a continuous query containing source and procedural windows.
dfESPcontquery *cq_01 = project_01->newContquery("contquery_01");
sw_01 = cq_01->newWindow_source("sourceWindow_01",
   dfESPindextypes::pi_HASH, dfESPstring("ID*:int32,custid:int32,
   loanid:int32,amount:money");
dfESPwindow_procedural *pw_01;
pw_01 = cq_01->newWindow_procedural("proceduralWindow_01",
   dfESPindextypes::pi_RBTREE, dfESPstring("ID*:int32,risk_score:double,
   derogRemarks:string");
dfESPpcontext *ctxt = new dfESPpcontext();

// Register the score() method of the credit_risk module to process
// the source window’s events.
ctxt->registerMethod_MAS(sw_01, "credit_risk", "score");

// Add an edge (directed arrow) between the two windows and start the project.
pw_01->registerMethodContext(ctxt);
cq_01->addEdge(sw_01, pw_01);
project_01->setNumThreads(2);
myEngine->startProjects();
```

XML Example

Here is an example of an XML continuous query definition. SAS Micro Analytic Service related elements are highlighted.

```xml
<engine port='55555'>
<description>
This example has one source window and one procedural window. The procedural window uses DS2 code to calculate a value from the source window.

The engine element creates the single engine top level container which sets up dfESP fundamental services such as licensing and logging.

```
This single engine instance wraps one or more projects that wrap one or more continuous queries and different types of windows.

This is to create a project. A project specifies a container that holds one or more continuous queries and are backed by a thread pool of user defined size. You can specify the pubsub port and type, number of threads for the project, index type, and a tag token data flow model.

This specifies the continuous query container that holds a collection of windows and enables you to specify the connectivity between windows. You can turn on tracing for a list of windows, and specify the index type for windows in the query.

This defines a source window. All event streams must enter continuous queries by being published or injected into a source window.
<connector class='fs' name='pub'>
    <properties>
        <property name='type'>pub</property>
        <property name='fstype'>csv</property>
        <property name='fsname'>input.csv</property>
        <property name='transactional'>true</property>
    </properties>
</connector>
</window-source>
<window-procedural name='pw1'>
    <description>
        This defines a procedural window. The window passes all fields of the event as variables to the DS2 program.
    </description>
    <schema>
        <fields>
            <field name='tradeID' type='string' key='true'/>
            <field name='security' type='string'/>
            <field name='quantity' type='int32'/>
            <field name='price' type='double'/>
            <field name='traderID' type='int64'/>
            <field name='time' type='string'/>
            <field name='volume' type='int32' key='true'/>
        </fields>
    </schema>
    <mas-map>
        <window-map module="module_1" revision="0" source="Trades" function="compute_volume"/>
    </mas-map>
    <connectors>
        <connector class='fs' name='sub'>
            <properties>
                <property name='type'>sub</property>
                <property name='fstype'>csv</property>
                <property name='fsname'>output.csv</property>
                <property name='snapshot'>true</property>
            </properties>
        </connector>
    </connectors>
</window-procedural>
<edges>
    <description>
        This fully specifies the continuous query with window connectivity, which is a directed graph.
    </description>
    <edge source='Trades' target='pw1'/>
</edges>
</contquery>
</contqueries>
<project-connectors>
    <connector-groups>
        <connector-group name='sub_group'>
            <connector-entry connector='trades_traders_cq/pw1/sub' state='running'/>
        </connector-group>
    </connector-groups>
</project-connectors>
Data Type Mappings

SAS Micro Analytic Service 2.2 executes within the procedural windows of continuous query applications. A continuous query can specify that events from one or more upstream windows be processed by SAS Micro Analytic Service when those events are received by a given procedural window. This specification maps each such input window’s events to one of the methods that have been published to SAS Micro Analytic Service.

During continuous query initialization, when a window’s events are registered with a method, SAS Micro Analytic Service inspects the window’s event schema and automatically maps the event’s fields to method input parameters by matching event field names with parameter names. It is legal for some, none, or all of the names to match. It is also legal for methods to have no input parameters. In that case, no input mapping is done. SAS Micro Analytic Service performs similar matching of the specified method’s output parameter names with the procedural window’s event field names. Therefore, at run time, when an event is received from the specified window, method input values are taken from that event, the method is executed, and the results are used to create a derived event. The event flows downstream from the procedural window to any subscribers.

The following table describes the data type mappings between event field types and DS2 method parameter types, and between event field types and Python function argument types. SAS Micro Analytic Service automatically translates the data types as needed according to the table below. The Event Stream Processing Event Field Type column lists the schema tag of each data type.

<table>
<thead>
<tr>
<th>Event Stream Processing Event Field Type</th>
<th>Event Stream Processing Type Description</th>
<th>DS2 Method Parameter Type</th>
<th>DS2 Type Description</th>
<th>Python Function Argument Type</th>
<th>Python Type Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int32</td>
<td>32-bit signed integer</td>
<td>int</td>
<td>32-bit signed integer</td>
<td>long</td>
<td>Long integer</td>
</tr>
<tr>
<td>int64</td>
<td>64-bit signed integer</td>
<td>BIGINT</td>
<td>64-bit signed integer</td>
<td>long</td>
<td>Long integer</td>
</tr>
<tr>
<td>double</td>
<td>IEEE double</td>
<td>double</td>
<td>IEEE double</td>
<td>float</td>
<td>Floating-point real</td>
</tr>
<tr>
<td>Event Stream Processing Event Field Type</td>
<td>Event Stream Processing Type Description</td>
<td>DS2 Method Parameter Type</td>
<td>DS2 Type Description</td>
<td>Python Function Argument Type</td>
<td>Python Type Description</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>------------------------------------------</td>
<td>---------------------------</td>
<td>----------------------</td>
<td>-------------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>string</td>
<td>UTF-8 string</td>
<td>CHAR, NCHAR, varchar, NVARCHAR</td>
<td>UTF-8 string</td>
<td>string</td>
<td>Unicode string</td>
</tr>
<tr>
<td>money</td>
<td>192-bit fixed decimal</td>
<td>double</td>
<td>IEEE double</td>
<td>float</td>
<td>Floating-point real</td>
</tr>
<tr>
<td>date</td>
<td>Date and time, as seconds since January 1, 1970</td>
<td>BIGINT</td>
<td>Seconds since January 1, 1970</td>
<td>long</td>
<td>Seconds since January 1, 1970</td>
</tr>
<tr>
<td>stamp</td>
<td>Date and time, as microseconds since January 1, 1970</td>
<td>BIGINT</td>
<td>Microseconds since January 1, 1970</td>
<td>long</td>
<td>Microseconds since January 1, 1970</td>
</tr>
<tr>
<td>array(i32)</td>
<td>32-bit signed integer array</td>
<td>int parameter[]</td>
<td>32-bit signed integer array</td>
<td>list</td>
<td>List of longs</td>
</tr>
<tr>
<td>array(i64)</td>
<td>64-bit signed integer array</td>
<td>bigint parameter[]</td>
<td>64-bit signed integer array</td>
<td>list</td>
<td>List of longs</td>
</tr>
<tr>
<td>array(dbl)</td>
<td>IEEE double array</td>
<td>double parameter[]</td>
<td>IEEE double array</td>
<td>list</td>
<td>List of floating point real</td>
</tr>
<tr>
<td>rstring</td>
<td>UTF-8 string</td>
<td>char parameter</td>
<td>UTF-8 string</td>
<td>string</td>
<td>Unicode string</td>
</tr>
<tr>
<td>blob</td>
<td>Binary large object</td>
<td>binary</td>
<td>Fixed-length binary data</td>
<td>Not supported</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

Note: **String** translates to either a single character type or to a variable length string type in DS2, depending on how the DS2 method parameter is declared. Be careful not to pass a multi-character string to a single CHAR argument in DS2, as run-time errors might occur.

Note: The **money** type is presented to DS2 or Python as double or float, respectively.

The SAS Event Stream Processing Engine uses the UNIX epoch for date and time values (January, 1, 1970). The values are presented to DS2 or Python as BIGINT or long, respectively, using the SAS epoch (January 1, 1960). Native DS2 and Python date and time types are not supported for public arguments. Seconds since 1960 is the SAS datetime value, which makes calling SAS date and time functions convenient in DS2. **Stamp** translates similarly, but with microsecond resolution rather than second resolution.
For the base data types (string, int32, int64, double, datetime, timestamp, and money), data is stored inline in the event. This allows for fast indexing and serialization.

For the array(i32), array(i64), array(dbl), blob, and rstring data types, data is not stored in an event, but rather in another location in memory. The event contains a pointer to the actual data. All of these object data types are reference counted at the object level. This allows an object to be referenced in multiple events, which saves memory and the amount of time that it would take to create a new object and copy the data. However, note that these data types cannot be used as key fields for an event.

Generating Multiple Derived Events from a Single Source Event

Overview

The methods that are published to SAS Micro Analytic Service can be used to generate multiple derived events when a single source event is received. This is accomplished by coding an array output argument for each derived event field. The values in the arrays are then used to populate the fields of multiple derived events.

A SAS Micro Analytic Service module method can produce one or more arrays. When such a method is mapped to a SAS Event Stream Processing source window, and when at least one of the output array names matches a derived event field name, multiple derived events can be generated. The number of elements in the matching array or arrays at run time determines the number of events that are generated. As is the case with scalar outputs, any array output that does not match a derived event field name is ignored.

When you are authoring a method that is capable of generating more than one event, the best practice is to produce parallel arrays of the same size. When you follow this practice, each array element contributes one value to each of the derived events, in order. The arrays can contain different numbers of elements across method calls (for example, they might generate one derived event for one call, three derived events for another, and so on). However, for a single method call, the best practice is to produce the same number of elements in each array.

If you ignore these best practices, you can still generate multiple events, but keep the following rules in mind:

- The longest array that maps to a derived event field determines the number of events that are generated.
- When the longest array is being determined, trailing missing values are not counted.
- If an array is shorter than the number of events to be generated, missing values are set in the corresponding field of the derived events for which the array has no data.

Scalar output values, if any, are repeated in the corresponding fields of every derived event.

Unique Keys

SAS Event Stream Processing requires every event to have a unique key composed of one or more event fields. Therefore, the source event’s key cannot be duplicated in multiple generated derived events. In that case, SAS Event Stream Processing halts processing and returns a duplicate key error. Because of this, the method author must ensure that each derived event has a unique key value. To do that, produce an array
containing unique key values and mark the matching event field as a key in the derived event schema. This technique is useful whenever you author your own module.

SAS Micro Analytic Service can execute modules that are generated by SAS analytics products, where the analytic functions do not produce arrays of unique key values. To accommodate such functions, SAS Micro Analytic Service provides a key generation feature. To use this feature, add a key field named _masRowNum of type int32 to the derived event schema, making it part of a composite key. When _masRowNum is present, SAS Micro Analytic Service populates the field with the row number of each derived event, in order, starting with 1. This feature ensures that the composite key value is unique across multiple generated derived events.

Here is an example of a derived event schema:

```
ID*:int32,_masRowNum*:int32,symbol:int32,
quantity:int32,price:double,total:double
```

Note: The module method does not need to produce a SAS Event Stream Processing key value when the meta field _masRowNum is present.

**Setting Opcodes and Flags on Multiple Derived Events**

When a set of derived events is generated from a single source event, you can explicitly set opcodes or flags in the derived events either individually or as a group. To set individual opcodes, use the _outOpcodeArray meta argument. Similar to _outOpcode, _outOpcodeArray can also be used to suppress the generation of any of the individual derived events.

To set flags in the derived events individually, use the _outFlagsArray meta argument.

To set the same opcode in all the derived events that are generated from a single source event, use the _outOpcode meta argument and omit the _outOpcodeArray meta argument. Similarly, to set the same flags in all the derived events that are generated from a single source event, use the _outFlags meta argument and omit the _outFlagsArray meta argument.

For information about the meta arguments _outOpcodeArray, _outFlagsArray, _outOpcode, and _outFlags, see “Operation Codes and Flags” on page 25.
Chapter 4
Processing Event Opcodes and Flags

Operation Codes and Flags

Each event contains an operation code, or opcode, and a set of flags. For a detailed explanation of these constructs, see SAS Event Stream Processing: Overview. SAS Micro Analytic Service module methods offer you the option of examining a source window event's opcode and flags and setting the opcode and flags of a derived event.

Note: This practice is recommended only for advanced SAS Event Stream Processing users.

These are the opcodes:
- insert
- update
- delete
- upsert
- safedelete

One or more flags can be set in a given event, depending on the event opcode and circumstances. Here are the possible flags:
- N — normal
- P — partial update
- R — retention

DS2 and Python module authors can add zero or more of the following special arguments to their DS2 method or Python function signatures:

_inOpcode
populated with the source window event opcode when the module method is called. _inOpcode is an input argument and, if included, must appear before any output arguments in the method signature. _inOpcode is a string type, and its value must be insert, update, delete, upsert, or safedelete when the method is called.
_outOpcode
used to either set the opcode of the derived event to emit, or to cause no derived event to be emitted. If _outOpcode is omitted from the method signature, SAS Micro Analytic Service transfers the opcode of the source window event to the derived event. This is the standard behavior under normal circumstances. If _outOpcode is included in the method signature and is set to missing, emission of the derived event is skipped. If _outOpcode is included and is not set to missing, the value of _outOpcode is used to set the opcode of the derived event. The value that is set must be either insert, update, delete, upsert, or safedelete. To achieve normal processing when _outOpcode is included, the method author must also include _inOpcode and set _outOpcode=_inOpcode. _outOpcode is an output argument. Therefore, it must appear after all input arguments in the method signature.

_inFlags
populated with the source window flags when the module method is called. _inFlags is a string type containing one character per source window event flag that is set. For example, N and NR are possible values. Reserve space for at least three characters for the _inFlags argument. _inFlags is an input argument. Therefore, if included, it must appear before any output arguments in the method signature.

_outFlags
used to set the flags of the derived event to emit. If _outFlags is omitted from the method signature, SAS Micro Analytic Service transfers the flags of the source window event to the derived event. This is the standard behavior under normal circumstances. If _outFlags is included and is set to missing, SAS Micro Analytic Service defaults to standard behavior and copies the source window event flags to the derived event.

---

DS2 Opcodes Example

The following simple example illustrates how to conditionally set the derived event opcode.

In this example, if the source window event's opcode is delete, and its quantity is greater than 2000, set the derived event's opcode to update, and set the price to 8.50. Otherwise, preserve normal processing by assigning _inOpcode to _outOpcode, and set the price to 10.00.

ds2_options sas;
package module_1/overwrite=yes;
method test_function(varchar(16) _inOpcode, int quantity, in_out double price, in_out varchar _outOpcode);
  if (_inOpcode = 'delete') and (quantity > 2000) then do;
    _outOpcode = 'update';
    price = 8.50;
  end;
  else do;
    _outOpcode = _inOpcode;
    price = 10.00;
  end;
end;
endpackage;
Overview

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

For more information about the DS2 programming language, see *SAS Viya: DS2 Language Reference*. 
DS2 Source Code Prerequisites

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

"ds2_options sas"

This 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. It should end with this statement:

"endpackage"

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

It is a best practice to include a line feed character at the end of each source code line. 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 that does not support either the full SAS language or PROC statements. Therefore, PROC statements cannot be used. However, here is an effective DS2 authoring and testing mechanism: develop your DS2 packages in SAS Foundation using PROC DS2 and publish those packages to SAS Micro Analytic Service after removing the surrounding PROC DS2 syntax.

Programming Blocks

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

The following code blocks are supported:

- PACKAGE...ENDPACKAGE
- METHOD...END
- DO...END

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

- TABLE...ENDTABLE
Public and Private Methods and Packages

Overview

Public methods are DS2 package methods that can be called by clients that are external to SAS Micro Analytic Service, such as SAS Event Stream Processing.

When a public method is registered with SAS Event Stream Processing as an event processor, the method's arguments are automatically mapped to the fields of the given source window and procedural window events. You register the method either by calling `dfESPproject::registerMethod_MAS()` or by including it in a window-map entry of an XML project definition.

Note: The method-argument-to-event-field mappings are by name and are case sensitive.

DS2 package methods to be used for event processing must follow all of the public method rules described below.

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 62.

**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.

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

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

i = lowerIndex;
j = higherIndex;

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

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

30  Chapter 5  •  DS2 Programming for SAS Micro Analytic Service
/* Move the index to the next position on both sides. */
    i = i+1;
    j = j-1;
end;
end;

/* Call quickSort recursively. */
if (lowerIndex < j) then do;
    quickSortStep(lowerIndex, j, numbers);
end;
if (i < higherIndex) then do;
    quickSortStep(i, higherIndex, numbers);
end;
end;

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

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

Private Method Example

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

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

Method Overloading

SAS Micro Analytic Service does not support method overloading. The DS2 programming language does support method overloading for programs running in other environments, but not when running in SAS Micro Analytic Service.
CAUTION:
If you publish a DS2 package that contains overloaded methods, run-time errors can occur.

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
• BINARY(n)
• CHAR(n)
• DOUBLE
• INTEGER
• NCHAR(n)
• NVARCHAR(n)
• VARCHAR(n)

Unsupported DS2 Data Types
• DATE
• DECIMAL(p, s)
• NUMERIC(p, s)
• PACKAGE
• TIME(p)
• TIMESTAMP(p)
• TINYINT
• VARBINARY(n)
Using ASTORE Models

About ASTORE Models

An Analytic Store, or ASTORE, is a system that allows the state of a trained predictive model to be saved in a transportable form. This enables it to subsequently be used to score new data in a variety of environments. Many SAS analytical procedures save the results from the training phase of model development as ASTORE models. A key feature of an ASTORE is that it can be easily transported from one platform to another. When an ASTORE is published to SAS Micro Analytic Service, the state of the predictive model is restored and is available for scoring new data.

Note: To use ASTORE models, SAS Event Stream Processing Analytics must be licensed.

Publishing an ASTORE Model

Unlike DS2 and Python modules, ASTORE models are not published to SAS Micro Analytic Service as source code. Instead, ASTORE models consist of binary code and metadata. Client applications deliver ASTORE models to SAS Micro Analytic Service as disk files.

Note: For information about calling an ASTORE model by a DS2 module, see the next section “Calling ASTORE Models Using DS2”.

Calling ASTORE Models Using DS2

If an ASTORE model has been registered with SAS Micro Analytic Service, it can be called by a DS2 module.

Note: For information about publishing an ASTORE model as a SAS Micro Analytic Service module for direct execution by calling clients, see the previous section “Publishing an ASTORE Model”.

Note: Because of the different ASTORE configurations that are required by the two execution methods, a given ASTORE model can be used for direct execution, or for execution by DS2, but not both at the same time.

A DS2 module that calls an ASTORE model must include an init() method that invokes the score package's setvars() and setkey() methods. It must also include the sc.setOption('_destroyFirstInstLast_', 0) option. This option makes ASTORE threading compatible with SAS Micro Analytic Service threading.

Note: Failure to set this option can cause the system to stop responding on module deletion or on shutdown.

The setvars() method is used by the DS2 score package to map variables to the ASTORE model's input and output parameters. The setkey() method takes a SHA-1 hexadecimal key as input and uses it to look up the ASTORE model.

SAS Micro Analytic Service automatically calls the init() method, if present, when a DS2 module is published.
Example

ds2_options sas;
package astoretest/overwrite=yes;
dcl package score sc();
dcl double CLAGE;
dcl double CLNO;
dcl double DEBTINC;
dcl double DELINQ;
dcl double NINQ;
dcl double VALUE;
dcl double _P_;
dcl double P__EVENT_0;
dcl double P__EVENT_1;
dcl nchar(32) I__EVENT_;
dcl nchar(4) _WARN_;
varlist allvars [_all_];

method init();
  sc.setOption('_destroyFirstInstLast_', 0);
  sc.setvars(allvars);
  sc.setkey(n'EB3D1CA20AA0CB74465D25EBB2290E13692AF750');
end;

method preCode();
  _P_ = 0.999;
end;

method postCode();
end;

method term();
end;

method astoreScore(double inCLAGE, double inCLNO, double inDEBTINC,
  double inDELINQ, double inNINQ, double inVALUE, 
  in_out double out_P_, in_out double outP__EVENT_0, 
  in_out double outP__EVENT_1, in_out nchar outI__EVENT_,
  in_out nchar out_WARN_); 
CLAGE = inCLAGE;
CLNO = inCLNO;
DEBTINC = inDEBTINC;
DELINQ = inDELINQ;
NINQ = inNINQ;
VALUE = inVALUE;

  preCode();
  sc.scoreRecord();
  postCode();

  out_P_ = _P_; 
  outP__EVENT_0 = P__EVENT_0;
  outP__EVENT_1 = P__EVENT_1;
  outI__EVENT_ = I__EVENT_;
Composite Modules

Composite modules enable DS2 code running in SAS Micro Analytic Service to call one or more ASTORE models. A composite module consists of one DS2 module and zero or more ASTORE models, which are known as the members of the composite. All members of a composite module are published, replaced, or deleted as a set. For example, if one member of a composite module fails to publish, then all member of the composite will fail to publish. Also, any prior revisions of the composite module will remain unchanged.

Like other module types, the revisions of a composite module are owned by a module context, have dictionaries, and can be queried for compilation messages, creation dates, and so on. The members of a composite module can include either a DS2 module or an ASTORE model, or both. Other module types, such as Python and C, are not supported as composite members.

Only the methods of the DS2 module of the composite are exposed for client applications to call.

To avoid name collisions with DS2 modules that exist outside a composite module, each composite revision has its own namespace. Therefore, each composite module must be self-contained and without dependencies on any non-member module. Unlike other module types, composite modules can be called only externally (for example, driven by SAS Event Stream Processing events or, in the case of SAS Decision Manager, by web service requests).

In SAS Event Stream Processing, a composite module is defined by specifying a DS2 module that calls an ASTORE model or models in the `<mas-module>` tag and by including the new `<module-members>` and `<module-member>` sub-tags of `<mas-module>`. Each `<module-member>` sub-tag should specify the location of an ASTORE model on disk. The following example is an excerpt from an ESP continuous query XML definition:

```xml
<mas-modules>
  <mas-module language="ds2" module="module_1" func-names='astoreScore'>
    <code-file>/your_ds2_folder/your astore_caller.ds2</code-file>
    <module-members>
      <module-member member='astore_1'
          SHAkey='EB3D1CA20Ah0CB74465D25EEB2290E13692AF750' type='astore'>
        <code-file>/your astore_model_folder/your astore_model</code-file>
      </module-member>
    </module-members>
  </mas-module>
</mas-modules>
```

For more information, see *SAS Event Stream Processing: Using SAS Event Stream Processing Studio*. 
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.

Return Results

If a DS2 method, or any method it calls, can result in a status code or failure, always include a method output argument for returning the result to the caller.
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.

```sas
/** 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.

```sas
/** 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;
```
Replacing SCAN (and TRANWRD) with DS2 Code

Consider the following code:

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

You can make the following observations:

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

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

```sas
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.

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

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

/* Are there more fields? 1 means there are more fields. 0 means there are no more fields. */
method hasmore() returns integer;
    if (strt >= blen) then return 0;
    return 1;
end;

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

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

/* Construct a STRTOK object using the parameters as initial values. */
method strtok(varchar(32766) bufinit, char(1) delim);
    load(bufinit, delim);
end;
/* Construct a STRTOK object without an initial buffer to be consumed. */

method strtok();
    strt = 0; blen = 0;
end;
endpackage; run;

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

Hash Package

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

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

/* If k1 and k2 are always smaller than 200, then */
/* size them smaller to reduce the time spent in */
/* the hash function when adding and finding values */
/* in the hash package. */

dcl package hash([k1 k2], [d1 d2]);

Character-to-Numeric Conversions

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

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

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

Passing Character Values to Methods

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

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

```sas
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:

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

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

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

---

### Moving Invariant Computations Out of Loops

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

```sas
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:

```sas
computed_x = compute(x);
do i = 1 to dim(a);
   if (computed_x eq a[i]) then ...;
end;
```
Chapter 7
Python Support in SAS Micro Analytic Service

Introduction

SAS Micro Analytic Service supports modules that are written in the Python programming language. A Python module represents a group of related Python functions.

Here is an example of a Python public function that can be hosted by SAS Micro Analytic Service. This example has no output. Input arguments are given in the function's argument list. This example has input variables a and b. Outputs of the function must be listed after "Output:" in the quoted string that follows the function definition. The output variables should match the variables listed in the return statement.

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

Note: Input and output argument names live in a single namespace. Therefore, they cannot be the same. This means that update arguments are not supported. This is true for all module types in SAS Micro Analytic Service, even though the Python language does not enforce such a restriction.
Public and Private Methods

Private and public methods are SAS Micro Analytic Service concepts, rather than Python features. Any method having the "Output:" doc string is considered a public method. If a method does not have the "Output:" doc string, then it is considered a private method. SAS Micro Analytic Service can host public and private Python methods, where a method is a Python function. Although a private method cannot be called directly, it can be called by another method (public or private). Private methods are useful as utility functions. Private methods are not callable externally. Therefore, they do not show up when querying the list of methods within a package. However, they can be called internally by other methods.

Python modules can be published containing all public methods, or a mixture of public and private methods. Both public and private methods can call other functions that either exist within the module internally or in external Python packages, including third-party libraries.

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

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

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

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

Example

The following example illustrates the use of each SAS Event Stream Processing data type as input to and as output from a public Python function.

```python
# # Name: scalarsTest.py # Purpose: Test the Python program for scalar types #
# # Inputs (name) (type) # inString String # inBool Boolean # inLong Long # inDouble Double # inTimestamp Long microseconds since 1960 # inDatetime Long seconds since 1960 # inMoney Double # # # Outputs (name) (type)
```
import datetime

def scalarsTest(inString, inBool, inLong, inDouble, inTimestamp, inDatetime, inMoney):
    "Output: outString, outBool, outLong, outDouble, outTimestamp, outDatetime, outMoney"

    if inString == None:
        outString = None
    else:
        # Convert the casing of the string input.
        outString = inString.swapcase()
        print ("\n inString=", inString, " outString=", outString, " (reverse case)"")

    if inBool == None:
        outBool = None
    else:
        # Reverse value of the Boolean.
        outBool = not inBool
        print ("\n inBool=", inBool, " outBool=", outBool, " (not inBool)"")

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

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

    if inTimestamp == None:
        outTimestamp = None
    else:
        # Since this is defined as a stamp in event stream processing
```
# schema, this number is long microseconds since 1960.
# Add one second == 1000000 microseconds.
outTimestamp = inTimestamp + 1000000
print ("inTimestamp=", inTimestamp, " outTimestamp=",
       outTimestamp, " (add one second)*")

if inDatetime == None:
    outDatetime = None
else:
    # Since this is defined as date in the event stream processing schema,
    # this number is long seconds since 1960.
    # Add one day.
    outDatetime = inDatetime + (3600 * 24)
print ("inDatetime=", inDatetime, " outDatetime=", outDatetime,
       " (add one day)*")

if inMoney == None:
    outMoney = None
else:
    # Add 25 cents.
    outMoney = inMoney + 0.25
print ("inMoney=", inMoney, " outMoney=", outMoney,
       " (add 25 cents)*")

# Return all of the outputs.
return outString, outBool, outLong, outDouble, outTimestamp,
      outDatetime, outMoney
```

---

**Configuring Python for SAS Event Stream Processing**

**Python 2.7 and 3.4 on 64–Bit Windows**


   *Note:* The Python 2.7 and 3.x installers include both version 2 and version 3 of Python. Regardless of which installer you download, you can create environments for both of the supported versions: 2.7 and 3.4.

   *Note:* During the installation process, you are prompted for the destination folder. These instructions assume you installed the folder in C:\Anaconda3.

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

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

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

   ```
   activate python34
   ```

4. When multiple distributions of Python are installed, Python often misjudges which environment or distribution you want to use. Therefore, it is recommended that you
set the PYTHONHOME environment variable to the value of your environment's top-level directory. Here is an example:

```
export PYTHONHOME=/users/myuserid/anaconda3/envs/python34
```

**Python 2.7 and 3.4 on 64–Bit Linux**

1. Download and run the latest Anaconda Linux 64-bit installer from https://www.continuum.io/downloads.

   *Note:* The Python 2.7 and 3.x installers include both version 2 and version 3 of Python. Regardless of which installer you download, you can create environments for both of the supported versions: 2.7 and 3.4.

2. After downloading the installer, enter the following in a terminal window. Provide the Python version that you installed. In the following example, Python 3.6 is used.

   ```
bash Anaconda3-4.3.0-Linux-x86_64.sh
```

   Answer yes to the question, **Do you wish the installer to prepend the Anaconda3 install location to PATH in your .bashrc?** These instructions assume that you 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%:}
```

6. When multiple distributions of Python are installed, Python often misjudges which environment or distribution you want to use. Therefore, it is recommended that you set the PYTHONHOME environment variable to the value of your environment's top-level directory. Here is an example:

   ```
export PYTHONHOME=/users/myuserid/anaconda3/envs/python34
```

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

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. Therefore, these issues affect Python 2.x only. When using Python 3.x, you can use the default encoding, and you can simply enclose literals in quotation marks.

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. You cannot do this with embedded Python. Therefore, it is recommended that you use the Python 3.x installer for both Python 2.7 and 3.4.

When starting SAS Event Stream Processing, do so from a shell in which you have activated Python, thus allowing the process to use Python.

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

Note: The use of the configuration scripts espenv and espenv_print, when running Python is highly recommended. These scripts are described in the next section, “Configuration Helper Scripts”.

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

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

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

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

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

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

Configuration Helper Scripts

SAS Event Stream Processing includes two shell scripts to assist with configuring environment variables for running SAS Event Stream Processing. Using these scripts to run Python modules in SAS Micro Analytic Service is optional.

The script that is called espenv_print prints a list of shell commands that can be copied and pasted into your .bashrc file or other shell script of your choice. The following example shows a sample command and its output, as run in a current Bash shell:

```bash
source /opt/sas/viya/home/SASEventStreamProcessingEngine/5.1.0/bin/espenv_print -i sas -m 3.4
```

# setting environment for official SAS install

#
The script that is named espenv can generate the same shell commands as espenv_print. However, instead of printing the commands to the console, it executes the commands, configuring the current shell to run Python modules in SAS Event Stream Processing.

In addition to the commands above, you can add other environment settings to your .bashrc file or your script of choice. For example, to override the default SAS Micro Analytic Service worker thread stack size, use the following command:
export MASktstacksize=<value>

*Note:* For information about valid value formats for the MASktstacksize environment variable, see “Deployment” on page 52.

If you plan to use only one version of Python, add the output of espenv_print to your .bashrc file. This runs the configuration automatically whenever you launch a Bash shell.

Use the espenv function if you switch between Python 2.7 and Python 3.4. To make Python configuration as convenient as possible, add the espenv function to your .bashrc file using the following steps. This ensures that the function is available to call from any folder, as long as you are in a Bash shell.

*Note:* The espenv_print script assumes that Python 2.7 and 3.4 environments are named “python27” and “python34”. If your environment uses different names, modify the following line accordingly in your .bashrc file:

```bash
source activate <python_environment_name>
```

1. Open espenv_func and copy the code.
2. Change directories to your home directory.
3. Paste the code into your .bashrc file.
4. Copy espenv_print to your home directory.
5. Edit espenv_print and then search for the following lines:

```bash
#***********************************************************************
# MODIFY PYTHONPATH (BELOW) WHENEVER PYTHON PACKAGES HAVE BEEN ADDED  *
# OR REMOVED FROM YOUR ANACONDA PYTHON 3.4 ENVIRONMENT                *
#***********************************************************************
```

6. The script contains two lines that set PYTHONPATH, one for Python 3.4 and the other for Python 2.7. These are located below each instance of the text that you searched for in the previous step.

Edit the appropriate line for PYTHONPATH and add paths for each Python package that you have installed and intend to use. Be sure to separate each path with a colon. $CP represents the root location where packages are stored in your Anaconda environment (such as the folder called site-packages). espenv_print automatically retrieves this location from Anaconda when the script is executed.

*Note:* Some Python packages require you to store more than one path because they store Python modules and executables in more than one folder.

7. Save your changes.

*Note:* Here is an alternative method:

1. Run espenv_print.
2. Copy and paste the output into your .bashrc file.
3. Append :$PATH to the end of the line that starts with PATH.
4. Append the following to the end of LD_LIBRARY_PATH:

   ```bash
   :$LD_LIBRARY_PATH
   ```

For a complete description of these scripts and their options, see *SAS Event Stream Processing on Linux: Deployment Guide*. 
Chapter 8
SAS Micro Analytic Service Logging and Deployment

SAS Micro Analytic Service Logging

SAS Micro Analytic Service uses the SAS Logging Facility. For more information, see SAS Viya Administration: Logging. SAS Event Stream Processing provides a default logging configuration file, and that file specifies loggers and appenders in addition to those described in this chapter. For more information about SAS Event Stream Processing, access the SAS product documentation at http://support.sas.com.

SAS Micro Analytic Service uses three loggers as follows:

- App.tk.MAS: logs start-up, shutdown, and method execution events.
- App.tk.MAS.Python: logs events related to Python.
- App.tk.MAS.CodeGen: logs code compilation and generation logging events, such as compiler warnings and errors.

Code that is hosted by SAS Micro Analytic Service, or the functions that it calls, can use additional loggers.

Normal operations, such as start-up and shutdown, are logged at the INFO level. Detailed information about operations such as compilation start and finish is logged at the DEBUG level. Warning and error conditions are logged at the WARN or ERROR levels, as appropriate. By default, App.tk.MAS is set to the ERROR level. Compiler messages are also retrieved and logged by SAS Event Stream Processing whenever it publishes a module to SAS Micro Analytic Service.

Note: When App.tk.MAS is set to TRACE, DEBUG, or INFO level, you see a message logging event that provides the SAS Micro Analytic Service version number in the log. Here is an example:

2017Oct12 17:37:55 INFO App.tk.MAS (tksf.c:4139) - Micro Analytic Service version information: 5.1.0, V.03.03M0D10122017, Thu Oct 12 17:36:00 EDT 2017

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

---

**Deployment**

SAS Micro Analytic Service is deployed automatically when SAS Event Stream Processing is deployed. If necessary, you can add environment customizations to your .bashrc file. For example, if you will use Python modules, you must complete the deployment and configuration steps that are described in “Configuring Python for SAS Event Stream Processing” on page 46.

If you encounter stack overflow errors when you are using the default SAS Micro Analytic Service worker thread stack size of 8 MB, you can override the default value by adding a definition of the MAStktstacksize environment variable to your .bashrc file. Note the following information when assigning a value to MAStktstacksize:

- You can specify a value using a unit of kilobytes or megabytes. The default value unit is kilobytes.
- To specify a value using megabytes, you must include “m” or “M” as a suffix following the number (for example, 10M or 10m).
- To specify a value using kilobytes, you can specify no suffix or add either a “k” or a “K” following the number. This means that any of the following are valid values: 10240, 10240k, 10240K.

Here are examples of different ways to assign a value of 10 MB to the MAStktstacksize environment variable:

- `export MAStktstacksize=10240`
- `export MAStktstacksize=10240k`
- `export MAStktstacksize=10240K`
- `export MAStktstacksize=10m`
- `export MAStktstacksize=10M`
Part 3

Using SAS Micro Analytic Services with SAS Decision Manager

Chapter 9
  DS2 Programming for SAS Micro Analytic Service ................. 55

Chapter 10
  Best Practices for DS2 Programming in SAS Decision Manager . . 65

Chapter 11
  Administration ................................................................. 71
Overview

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

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

DS2 Source Code Prerequisites

The DS2 source code submitted to SAS Micro Analytic Service should begin with the following statement, just above the PACKAGE statement:
"ds2_options sas"

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

"endpackage"

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

It is a best practice to include a line feed character at the end of each source code line. It is possible to place all of your source code on a single line. However, doing so makes it difficult to use compiler warning and error messages that include line numbers.

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

SAS Micro Analytic Service and SAS Foundation

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

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

```sas
proc ds2;
	ds2_options sas;
package myPackage/overwrite=yes;
method copyArray(char(12) in_array[4], in_out char(12) out_array[4]);
    out_array := in_array;
end;
endpackage;
run;

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

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

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

**Architectural Considerations**

If the SQLSTMT package is used to access third-party databases, the associated SAS/ACCESS products and third-party client libraries are required on each system that is running SAS Micro Analytic Service. For more information, see the SAS Viya deployment documentation on support.sas.com.

Vendor libraries are provided by the database vendor, such as IBM (for DB2) or Oracle Corporation (for an Oracle database). They must be installed separately according to the instructions that are provided by the vendor. In addition, SAS Micro Analytic Service must be configured to use these vendor libraries by setting appropriate environment variables. For information, see “Using Third-Party Database Drivers” on page 58.

SAS Micro Analytic Service is a multi-threaded service in which each thread requires concurrent access to the database. Access to SAS data sets is supported. However, since SAS data sets use file-level locking, they are not suitable for writing from multiple threads. It is recommended that you carefully set appropriate connection options before reading SAS data sets from multiple threads, since problems can lead to a deadlock situation. For these reasons, the use of a third-party database management system is highly recommended.

**Connection Strings and Configuration**

**Overview**

Connection strings are used to specify database connection information such as host, port, driver, database, catalog, schema, credentials, and options. The SQLSTMT package supports the FedSQL dialect. Therefore, the connection string should begin with the following information:

```
DRIVER=SQL;CONOPTS=(
```

In the above string, `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))
```
For more information about connection string structure, see the SQLSTMT package information in *SAS Viya: DS2 Language Reference* and Appendix 3, “Table Service Driver Reference,” on page 97.

**Defining and Configuring Connection Strings**

SAS Micro Analytic Service allows the connection string to be supplied through a configuration process. Although the SQLSTMT package allows specifying a connection string in code, SAS recommends that it be supplied using the SAS Micro Analytic Service configuration. The benefits to doing this are as follows:

- When connection information is not included in the code, the module is more portable. This enables it to be moved to other SAS Micro Analytic Service deployments without requiring code modifications. This can help implement strategies for environment promotion (Development-Test-Production) as well as for hot standby failover.

- When SAS Micro Analytic Service manages the database connection, it can detect whether connectivity to the database is lost, and automatically tries to reconnect on a periodic basis.

- If severe errors occur during database access, SAS Micro Analytic Service automatically attempts to recover from the error by recompiling the DS2 packages.

The connection string for SAS Micro Analytic Service deployments is configured by setting the *sas.microanalyticservice.service.connectionstring* property in the configuration section of SAS Environment Manager. For more information, see “SAS Micro Analytic Service Configuration” on page 72.

This property is configured on a per-tenant property basis. This means that every tenant has its own private copy of the property. It is not shared with other tenants.

*Note:* Although only one connection string can be specified per tenant, the connection string itself can be federated, which allows multiple databases to be accessed using catalog and schema names.

**Using Third-Party Database Drivers**

As mentioned in the “Overview”, database vendor-specific libraries (often referred to as database driver software) must be installed according to the instructions that are provided by the database vendor. As part of the installation, these drivers provide either manual instructions or scripts that set up environment variables (for example, PATH and LD_LIBRARY_PATH). To use these libraries, client software such as SAS Micro Analytic Service must have access to these environment variables.

To enable access to these environment variables, edit the following file to include the definitions of the required environment variables:

```
/opt/sas/viya/config/etc/sysconfig/microanalyticservice.conf
```

In the path above, *viya* is the deployment ID.

Note that this file does not exist in a standard installation and must be created. Each node of a cluster deployment that has a deployment of SAS Micro Analytic Service must be updated with the third-party driver installation and this file.

After installing the drivers and updating the file with the appropriate values, you must restart SAS Micro Analytic Service to apply the changes.
To set this configuration, it can be helpful to view all environment variables that are available to SAS Micro Analytic Service. To accomplish this, enable TRACE logging for com.sas.mas.impl.MASFactivityImpl.

SAS Micro Analytic Service enables access to HTTP and HTTPS web services through the DS2 HTTP package. This package 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, see *SAS Viya: DS2 Language Reference* and *SAS Viya: FedSQL Programming for SAS Cloud Analytic Services*.

---

**Programming Blocks**

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

The following code blocks are supported:

- PACKAGE...ENDPACKAGE
- METHOD...END
- DO...END

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

- TABLE...ENDTABLE
- THREAD...ENDTHREAD

Similarly, the following statements are not supported:

- OUTPUT
- SET

---

**Public and Private Methods and Packages**

**Overview**

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

SAS Micro Analytic Service can host public DS2 packages and private DS2 packages. Private DS2 packages have fewer restrictions on the DS2 features that can be used than public packages have. Although a private DS2 package cannot be called directly, it can be called by another DS2 package. Private DS2 packages are useful as utility functions, as solution-specific built-in functions, or for solution infrastructure. See your SAS solution documentation for a description of the solution-specific built-in functions that you can use when authoring custom DS2 modules.
A public DS2 package can contain private methods, as long as it contains at least one public method. Any method that does not conform to the rules for public methods is automatically treated as private. Private methods are allowed and do not produce errors if they contain correct DS2 syntax. Private methods are not callable externally. Therefore, they do not show up when querying the list of methods within a package. However, they can be called internally by other DS2 package methods. Here are several typical uses of private methods:

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

Public Method Rules

Public methods must conform to the following rules:

- The return type must be void. Rather than using a single return type, public methods can return multiple outputs, where each output argument specifies the in_out keyword in the method declaration. Non-void methods are treated as private.
- Arguments that are passed by reference (meaning ones that specify in_out) are treated as output only. True update arguments are not supported by public methods. This restriction results in more efficient parameter marshaling and supports all interface layers, including REST.
- Input arguments must precede output arguments in the method declaration. It is permissible for a method to have only inputs or only outputs. However, if both are present, all inputs must precede the outputs.
- DS2 packages must not be passed as arguments in public methods. The presence of a DS2 package argument results in the method becoming private.
- The VARARRAY statement must not be present in the argument list of a public method. VARARRAY is a DS2 statement, not a data type. The presence of VARARRAY in a methods argument list causes the method to become private.
- For a full list of data types that can be used as public method arguments, see “Supported DS2 Data Types” on page 62.

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.

```ds2
method quickSortStep (int lowerIndex, int higherIndex, in_out double numbers[10]);

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

i = lowerIndex;
j = higherIndex;

/* Calculate the pivot number, taking the pivot as the * middle index number. */
pivot = numbers[ceil(lowerIndex+(higherIndex-lowerIndex)/2)];
```
/* Divide into two arrays */
do while (i <= j);
/**
 * In each iteration, identify a number from the left side that
 * is greater than the pivot value. Also identify a number
 * from the right side that is less than the pivot value.
 * Once the search is done, then exchange both numbers.
 */
do while (numbers[i] < pivot);
i = i+1;
end;
do while (numbers[j] > pivot);
j = j-1;
end;
if (i <= j) then do;
temp = numbers[i];
numbers[i] = numbers[j];
numbers[j] = temp;

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

/* Call quickSort recursively. */
if (lowerIndex < j) then do;
    quickSortStep(lowerIndex, j, numbers);
end;
if (i < higherIndex) then do;
    quickSortStep(i, higherIndex, numbers);
end;
end;

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

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

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

Method Overloading

SAS Micro Analytic Service does not support method overloading.

Argument Types Supported in Public Methods

Overview

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

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

Supported DS2 Data Types

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

 Unsupported DS2 Data Types

- BINARY(n)
- DATE
- DECIMAL(p, s)
- NUMERIC(p, s)
- PACKAGE
- TIME(p)
- TIMESTAMP(p)
- TINYINT
- VARBINARY(n)
Chapter 10
Best Practices for DS2 Programming in SAS Decision Manager

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.

Return Results
If a DS2 method, or any method it calls, can result in a status code or failure, always include a method output argument for returning the result to the caller.
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;
end;
Replacing SCAN (and TRANWRD) with DS2 Code

Consider the following code:

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

You can make the following observations:

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

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

```sas
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.

```sas
/** 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;
```
method load(in_out varchar bufinit, char(1) delim);
  _buffer = bufinit .. delim;
  _delim = delim;
  strt = 1;
  blen = length(_buffer);
end;

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

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;

method getnext() returns varchar(32767);
  dcl varchar(32767) tok;
  getnext(tok);
  return tok;
end;

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();
    strtr = 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 "I" with the hash package is the length of the key. The longer the key, the longer the hash function takes to operate.

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

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

---

Character-to-Numeric Conversions

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

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

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

---

Passing Character Values to Methods

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

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

```sas
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:

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

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

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

---

**Moving Invariant Computations Out of Loops**

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

```sas
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:

```sas
computed_x = compute(x);
do i = 1 to dim(a);
  if (computed_x eq a[i]) then ...;
end;
```
SAS Micro Analytic Service Administration

SAS Micro Analytic Service is implemented as a SAS Viya microservice. Most of the administrative tasks and capabilities are described in the SAS Viya Administration Guide.

Because SAS Micro Analytic Service is implemented using both Java and SAS threaded kernel technology, there are some areas in which it differs from a standard SAS Viya microservice.

The items that are implemented using SAS threaded kernel technology are referred to as the Micro Analytic Service core component. The items that are implemented using Java are referred to as the REST service.

Starting and Stopping SAS Micro Analytic Service

To start or stop SAS Micro Analytic Service, use the service start or stop commands as described in the SAS Viya Administration Guide.

The following commands are examples that are specific to SAS Micro Analytic Service. In these examples, viya is the deployment ID and default is the instance ID.

These commands start and stop the service:

```
sudo service sas-viya-microanalyticservice-default start
```
sudo service sas-viya-microanalyticservice-default stop

This command verifies whether the service is running:

sudo service sas-viya-microanalyticservice-default status

Note: For a multi-machine deployment, you must start and stop the service on each deployment server.

SAS Micro Analytic Service Configuration

You use SAS Environment Manager to configure SAS Micro Analytic Service. SAS Environment Manager is a web application for managing a SAS Viya environment. It includes a dashboard view, which provides a quick overall look of your environment’s health and status, as well as detailed views that enable you to examine and manage your environment in detail. For more information, see SAS Viya Administration: Using SAS Environment Manager.

The SAS Micro Analytic Service configuration parameters are organized into sections as follows:

- **JVM**: Contains the parameters to configure the Java virtual machine to run SAS Micro Analytic Service.
  The most commonly used parameters are the minimum and maximum heap space sizes, and the Java thread stack size. The default value for the minimum and maximum heap space size parameters is 1GB. The default value for the Java thread stack size is 512K.

  Note the following about these parameters:
  - You must set the Java thread stack size value to be greater than 512K.
  - You might need to increase the heap space size values depending on the number and size of the modules.
  - After you change configuration parameters, you must restart SAS Micro Analytic Service.

- **Logging levels**: Contains the parameters to configure what messages are logged. For information, see “Logging Levels” on page 77.

- **sas.microanalyticservice**: Contains two configuration items that are specific to SAS Micro Analytic Service:
  - **sas.microanalyticservice.system**: Contains items that are specific to system-wide configuration of SAS Micro Analytic Service.
  - **sas.microanalyticservice.properties**: For a multi-tenant deployment, contains tenant-specific items. For example, the database connection string is configured in this section.

For each sas.microanalyticservice sub-section, the configuration items are categorized into two groups—core and service. The core group contains items that affect the Micro Analytic Service core component. The service group configures the REST service component. There is also a custom section that is used to add and remove properties.
The following configuration parameters apply to SAS Micro Analytic Service core:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| numthreads                | Integer | 4             | Specifies the number of threads in the SAS Micro Analytic Service core. The amount of memory that is used by SAS Micro Analytic Service core is determined by the number of threads. Note the following information when setting a value:  
  • The compilation time for modules is proportional to the number of threads.  
  • You can increase or decrease the value, depending on the availability of system memory.  
  • Setting the value to 0 causes the SAS Micro Analytic Service core to use all possible cores on the system. For a system in which SAS Micro Analytic Service is the primary application, this is recommended. |
| gcintervalseconds         | Integer | 60            | The time, in seconds, between garbage collection runs.                                                                                                                                                     |
| graceperiodseconds        | Integer | 10            | The amount of time to wait, in seconds, before cleaning up the assets that are associated with a deleted revision.  
  Note: A request to execute a deleted revision, even during the grace period, is rejected with an error message that indicates the revision has been deleted. |
| nativebuffersize          | Integer | 512           | The size of the buffer, in bytes, to exchange data between the REST service and SAS Micro Analytic Service core.                                                                                           |
| dbconnretries             | Integer | 10            | The number of times that the system attempts to connect to a database.                                                                                                                                   |
| dbconretryintervalseconds | Integer | 5             | The amount of time, in seconds, that the system waits before retrying to connect to a database.                                                                                                           |
| ds2maxrecompilecount      | Integer | 1000          | If there is an error because of failing database connections, the maximum number of times that the system tries to recompile DS2 code before ejecting the module.                                            |
| connectionstring          | String  |               | The FedSQL connection string used to connect to a database.                                                                                                                                                |
The following configuration parameters apply to the REST service component:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sasmode</td>
<td>Boolean</td>
<td>True</td>
<td>If this is set to True, the DS2 code is compiled with ds2_options sas.</td>
</tr>
<tr>
<td>moduleidgeneration</td>
<td></td>
<td></td>
<td>A sub-category containing two properties that control how the module ID is determined. The properties are factorytype and forcelowercase.</td>
</tr>
<tr>
<td>factorytype</td>
<td>One of the following:</td>
<td>ModuleNameOverride Package</td>
<td>This value determines how the module ID is created for a new module. The module name is the one that is passed to the system during module creation.</td>
</tr>
<tr>
<td>forcelowercase</td>
<td>Boolean</td>
<td>True</td>
<td>If this is set to True, the module ID returned is always lowercase.</td>
</tr>
<tr>
<td>defaultmasuserctxname</td>
<td>String</td>
<td>defaultMAS UserCtxName</td>
<td>Default value of the user context name, if the tenant name is not used.</td>
</tr>
</tbody>
</table>

The following items are custom SAS Micro Analytic Service parameters that are not documented in SAS Environment Manager:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>core.tktstacksizekbytes</td>
<td>Integer</td>
<td>8192</td>
<td>Sets stack size, in kilobytes, for worker threads in SAS Micro Analytic Service core. The default value is sufficient for most purposes. You might need to increase the value to compile large DS2 packages that contain numerous (for example, more than 5,000) IF-THEN/ELSE statements.</td>
</tr>
<tr>
<td>core.profilesamplefrequency</td>
<td>Integer</td>
<td>2000</td>
<td>Used in conjunction with profile logging. Specifies the sample rate to log the execution time of the module code.</td>
</tr>
</tbody>
</table>
## SAS Micro Analytic Service Logging

### Overview

The SAS Micro Analytic Service core and the REST service each create log files. A new log file is created according to the resource logging configuration. This might include the occurrence of any of the following:

- a new day begins (rollover at midnight)
- the log file size exceeds the maximum value
- the service starts

The SAS Micro Analytic Service log files are typically created in the following folder:

```
/opt/sas/viya/config/var/log/microanalyticservice/default
```
In the path above, *viya* is the deployment ID, and *default* is the instance ID.

The following table provides default logging information for each service:

<table>
<thead>
<tr>
<th>Service</th>
<th>Default naming convention</th>
<th>Default file rollover configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS Micro Analytic Service core</td>
<td>sas-microanalyticservice-core_yyyy-MM-dd_HH-mm-ss.log</td>
<td>When either of the following occur:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• the log file size exceeds 100K</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• the service starts</td>
</tr>
<tr>
<td>REST</td>
<td>sas-microanalyticservice_yyyy-MM-dd_HH-mm-ss.log</td>
<td>each day at midnight</td>
</tr>
</tbody>
</table>

For complete information about logging, see *SAS Viya Administration: Logging*.

**Loggers and Logging Levels**

Loggers are split into two groups: those that apply to SAS Micro Analytic core and those that apply to the REST service.

**SAS Micro Analytic Core**

Logger names that start with the following prefixes apply to the SAS Micro Analytic Service core: Admin, App, Audit, IOM, Perf, DataFlux.

SAS Micro Analytic core loggers are not typically designed to inherit levels. For example, setting App.TableServices.DS2 does not automatically set the same level for App.TableServices.DS2.Runtime.

Here are some important SAS Micro Analytic Service core loggers:

- **App.tk.MAS**: Logs start-up, shutdown, and method execution events. The default value is INFO.

- **App.tk.MAS.CodeGen**: Logs compilation and generation logging events such as compiler warning and errors. The default value is FATAL. Compile errors are returned to the user when submitting a module source code.

- **Audit.Table.Connection**: Logs database connection events from the DS2 packages.

- **DataFlux.Licensing**: Logs licensing messages. The default value is INFO.

Loggers that start with App.TableServices.DS2.Runtime. or App.TableServices.DS2.Config. can be used to diagnose DS2 problems. Module code might also use other loggers.

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

**Note:** When App.tk.MAS is set to TRACE, DEBUG, or INFO level, you see a message logging event that provides the SAS Micro Analytic Service version number in the log. Here is an example:
REST Service

Logger names that start with a Java domain, such as org or com, apply to the SAS Micro Analytic Service REST service. In addition, other loggers exist that log information about memory usage and profiling.

Some important SAS Micro Analytic Service REST service loggers include the following:

- **com.sas.mas.service.rest.controllers.ModulesController**: Logs all REST calls to SAS Micro Analytic Service.
- **com.sas.mas.impl.MASFactoryImpl**: Logs environment variables that are presented to SAS Micro Analytic Service core.
- **MAS_MEM.NATIVE**: Logs memory usage by SAS Micro Analytic Service core when executing module code.
- **MAS_MEM.CODE**: Logs memory usage by SAS Micro Analytic Service core when compiling module code.
- **MAS_PROFILE**: Logs sampled timings for compilation and execution of module code.

Logging Levels

The logging levels FATAL, ERROR, WARN, INFO, DEBUG, and TRACE are supported. Consider the following when you are setting logging levels:

- Normal operations, such as start-up and shutdown, are logged at the INFO level. Detailed information is logged at the DEBUG and TRACE level.
- For normal operations, it is recommended that you enable either the ERROR level or the WARN level.
- More verbose and frequent information, such as memory and profile logging, is logged only at the TRACE level.
- Enabling DEBUG and TRACE typically affects performance. Therefore, it is recommended that the DEBUG and TRACE levels are used only during system sizing, performance tuning, or when troubleshooting issues.

SAS Micro Analytic Service Security and Authorization

Access to the REST API endpoints in SAS Micro Analytic Service is determined by authorization rules. For information about how to modify authorization rules, see “Authorization” in the SAS Viya Administration Guide.

Upon installation, authorization rules are created for each SAS Micro Analytic Service endpoint. For each rule, permissions are granted to all authenticated users for specific operations, such as CREATE, READ, UPDATE, and DELETE.

It is recommended that your system administrator perform the following tasks to control access to the SAS Micro Analytic Service, according to your site requirements:

- create new groups of users or update existing groups
• modify the Principal value to assign the applicable group
• modify the Setting value (if necessary)

For example, the following settings show the default Create and Delete permissions for the SAS Micro Analytic Service:

<table>
<thead>
<tr>
<th>Object URI</th>
<th>Principal</th>
<th>Setting</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>/microanalyticScore/modules/*</td>
<td>Authenticated Users</td>
<td>Grant</td>
<td>Create, Delete</td>
</tr>
</tbody>
</table>

You can isolate Create and Delete access to the service by creating a new group and assigning that group as the Principal value. For example, the following settings show that a new group, called MAS Administrators, is the only group with Create and Delete permissions for the SAS Micro Analytic Service:

<table>
<thead>
<tr>
<th>Object URI</th>
<th>Principal</th>
<th>Setting</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>/microanalyticScore/modules/*</td>
<td>MAS Administrators</td>
<td>Grant</td>
<td>Create, Delete</td>
</tr>
</tbody>
</table>

For information about creating new user groups, see “Identity Management” in the SAS Viya Administration Guide.

### Secure DS2 HTTP Package Usage

The DS2 HTTP package supports HTTP and HTTPS endpoints. If the default list of trusted CA certificates does not enable access to all of the secure endpoints that you want to reach, refer to *Encryption in SAS Viya: Data in Motion* for more information. If environment variables such SSLCALISTLOC are needed, they can be added to the `/opt/sas/viya/config/etc/sysconfig/microanalyticservice.conf` file.

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—Basic and Negotiate. It does not support the Digest mechanism. Because Basic authentication 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 Viya: DS2 Programmer’s Guide*. 

Part 4

Appendixes

Appendix 1
  SAS Micro Analytic Service Return Codes  81

Appendix 2
  REST Server Error Messages and Resolutions  93

Appendix 3
  Table Service Driver Reference  97
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.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Hexadecimal Code</th>
<th>#define Symbol</th>
<th>Message or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1958744063</td>
<td>0x8b3ff001U</td>
<td>MASBadArgs</td>
<td>Invalid arguments.</td>
</tr>
<tr>
<td>-1958744062</td>
<td>0x8b3ff002U</td>
<td>MASInternalError</td>
<td>Internal error.</td>
</tr>
<tr>
<td>-1958744061</td>
<td>0x8b3ff003U</td>
<td>MASFailure</td>
<td>SAS Micro Analytic Service encountered a failure.</td>
</tr>
<tr>
<td>-1958744060</td>
<td>0x8b3ff004U</td>
<td>MASFail</td>
<td>%s encountered a failure.</td>
</tr>
<tr>
<td>-1958744059</td>
<td>0x8b3ff005U</td>
<td>MASUnexFail</td>
<td>%s encountered an unexpected failure.</td>
</tr>
<tr>
<td>-1958744058</td>
<td>0x8b3ff006U</td>
<td>MASUnexInternal</td>
<td>%s encountered an unexpected internal failure.</td>
</tr>
<tr>
<td>-1958744057</td>
<td>0x8b3ff007U</td>
<td>MASUnexFailIn</td>
<td>%s encountered an unexpected failure in %s.</td>
</tr>
<tr>
<td>-1958744056</td>
<td>0x8b3ff008U</td>
<td>MASFailIn</td>
<td>%s encountered a failure in %s.</td>
</tr>
<tr>
<td>-1958744055</td>
<td>0x8b3ff009U</td>
<td>MASFailWithText</td>
<td>%s encountered a failure in %s: %s.</td>
</tr>
<tr>
<td>-1958744054</td>
<td>0x8b3ff00aU</td>
<td>MASSFGCBLock</td>
<td>Failed to obtain the SFGCB lock.</td>
</tr>
<tr>
<td>-1958744053</td>
<td>0x8b3ff00bU</td>
<td>MASEXeLock</td>
<td>Failed to obtain the .exe lock.</td>
</tr>
<tr>
<td>-1958744052</td>
<td>0x8b3ff00cU</td>
<td>MASLockCreate</td>
<td>Failed to create the %s lock.</td>
</tr>
<tr>
<td>-1958744051</td>
<td>0x8b3ff00dU</td>
<td>MASEventCreate</td>
<td>Failed to create the %s event for thread %d.</td>
</tr>
<tr>
<td>-1958744050</td>
<td>0x8b3ff00eU</td>
<td>MASThreadCreate</td>
<td>Failed to create SAS Micro Analytic Service worker thread %d of %d.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Hexadecimal Code</td>
<td>#define Symbol</td>
<td>Message or Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
<td>---------------------</td>
<td>----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-1958744049</td>
<td>0x8b3ff000U</td>
<td>MASCPUcount</td>
<td>Failed to determine the number of CPUs. Setting the number of worker threads to %d.</td>
</tr>
<tr>
<td>-1958744048</td>
<td>0x8b3ff010U</td>
<td>MASThreadCount</td>
<td>The number of threads requested, %d, exceeds the limit. The maximum allowable threads = %d times the number of CPUs = %d.</td>
</tr>
<tr>
<td>-1958744047</td>
<td>0x8b3ff011U</td>
<td>MASThreadPoolSize</td>
<td>Worker thread pool size is set to: %d.</td>
</tr>
<tr>
<td>-1958744046</td>
<td>0x8b3ff012U</td>
<td>MASInitAlready</td>
<td>SAS Micro Analytic Service was already initialized.</td>
</tr>
<tr>
<td>-1958744045</td>
<td>0x8b3ff013U</td>
<td>MASInitFailed</td>
<td>SAS Micro Analytic Service failed to initialize.</td>
</tr>
<tr>
<td>-1958744044</td>
<td>0x8b3ff014U</td>
<td>MASNotLicensed</td>
<td>SAS Micro Analytic Service is not licensed.</td>
</tr>
<tr>
<td>-1958744043</td>
<td>0x8b3ff015U</td>
<td>MASLicSvcInitFailed</td>
<td>License service failed to initialize.</td>
</tr>
<tr>
<td>-1958744042</td>
<td>0x8b3ff016U</td>
<td>MASNotInitialized</td>
<td>SAS Micro Analytic Service is not initialized.</td>
</tr>
<tr>
<td>-1958744041</td>
<td>0x8b3ff017U</td>
<td>MASTermFailed</td>
<td>SAS Micro Analytic Service failed to terminate successfully.</td>
</tr>
<tr>
<td>-1958744040</td>
<td>0x8b3ff018U</td>
<td>MASArgTrunc</td>
<td>The maximum size of parameter %d in the %s call is not large enough, and the value has been truncated at %d characters.</td>
</tr>
<tr>
<td>-1958744039</td>
<td>0x8b3ff019U</td>
<td>MASCompStatus</td>
<td>Compiler encountered status 0x%X.</td>
</tr>
<tr>
<td>-1958744038</td>
<td>0x8b3ff01aU</td>
<td>MASUnsupportedType</td>
<td>Unsupported type.</td>
</tr>
<tr>
<td>-1958744037</td>
<td>0x8b3ff01bU</td>
<td>MASUnknownType</td>
<td>Unknown type.</td>
</tr>
<tr>
<td>-1958744036</td>
<td>0x8b3ff01cU</td>
<td>MASNoSuchPackage</td>
<td>Package not found.</td>
</tr>
<tr>
<td>-1958744035</td>
<td>0x8b3ff01dU</td>
<td>MASNoSuchMethod</td>
<td>Method not found.</td>
</tr>
<tr>
<td>-1958744034</td>
<td>0x8b3ff01eU</td>
<td>MASNoSuchRevision</td>
<td>Revision not found.</td>
</tr>
<tr>
<td>-1958744033</td>
<td>0x8b3ff01fU</td>
<td>MASRevisionGet</td>
<td>Failed to get revision.</td>
</tr>
<tr>
<td>-1958744032</td>
<td>0x8b3ff020U</td>
<td>MASNoSuchModule</td>
<td>Module not found.</td>
</tr>
<tr>
<td>-1958744031</td>
<td>0x8b3ff021U</td>
<td>MASNoSuchUserContext</td>
<td>User context not found.</td>
</tr>
<tr>
<td>-1958744030</td>
<td>0x8b3ff022U</td>
<td>MASModuleCtxCreate</td>
<td>Failed to create module context.</td>
</tr>
<tr>
<td>-1958744029</td>
<td>0x8b3ff023U</td>
<td>MASUserCtxCreate</td>
<td>Failed to create user context.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Hexadecimal Code</td>
<td>#define Symbol</td>
<td>Message or Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------</td>
<td>---------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>-1958744028</td>
<td>0x8b3f024U</td>
<td>MASArgTypeMismatch</td>
<td>Argument type mismatch.</td>
</tr>
<tr>
<td>-1958744027</td>
<td>0x8b3f025U</td>
<td>MASArgCoutMismatch</td>
<td>Argument count mismatch.</td>
</tr>
<tr>
<td>-1958744026</td>
<td>0x8b3f026U</td>
<td>MASClientCodegenError</td>
<td>Code generation error.</td>
</tr>
<tr>
<td>-1958744025</td>
<td>0x8b3f027U</td>
<td>MASDS2CompileError</td>
<td>DS2 compilation error.</td>
</tr>
<tr>
<td>-1958744024</td>
<td>0x8b3f028U</td>
<td>MASDS2RuntimeError</td>
<td>DS2 run-time error.</td>
</tr>
<tr>
<td>-1958744023</td>
<td>0x8b3f029U</td>
<td>MASTKGNoEntryPoint</td>
<td>Code generation did not find an entry point.</td>
</tr>
<tr>
<td>-1958744022</td>
<td>0x8b3f02aU</td>
<td>MASTKGGenericError</td>
<td>Code generation generic error.</td>
</tr>
<tr>
<td>-1958744021</td>
<td>0x8b3f02bU</td>
<td>MASInvalidRequest</td>
<td>Invalid request.</td>
</tr>
<tr>
<td>-1958744020</td>
<td>0x8b3f02cU</td>
<td>MASMissingEntryPoints</td>
<td>Missing entry points.</td>
</tr>
<tr>
<td>-1958744019</td>
<td>0x8b3f02dU</td>
<td>MASUnassignedInput</td>
<td>Unassigned input.</td>
</tr>
<tr>
<td>-1958744018</td>
<td>0x8b3f02eU</td>
<td>MASInternalOnly</td>
<td>Internal only.</td>
</tr>
<tr>
<td>-1958744017</td>
<td>0x8b3f02fU</td>
<td>MASOnlyValidForDS2</td>
<td>Valid only for DS2 code.</td>
</tr>
<tr>
<td>-1958744016</td>
<td>0x8b3f030U</td>
<td>MASOnlyValidForC</td>
<td>Valid only for C code.</td>
</tr>
<tr>
<td>-1958744015</td>
<td>0x8b3f031U</td>
<td>MASExecutionException</td>
<td>Exception occurred during execution.</td>
</tr>
<tr>
<td>-1958744014</td>
<td>0x8b3f032U</td>
<td>MASCompilationException</td>
<td>Exception occurred during compilation.</td>
</tr>
<tr>
<td>-1958744013</td>
<td>0x8b3f033U</td>
<td>MASDS2ThreadUnsupported</td>
<td>DS2 thread unsupported.</td>
</tr>
<tr>
<td>-1958744012</td>
<td>0x8b3f034U</td>
<td>MASTKEDSError</td>
<td>DS2 error.</td>
</tr>
<tr>
<td>-1958744011</td>
<td>0x8b3f035U</td>
<td>MASUnrecognizedLanguage</td>
<td>Unrecognized language.</td>
</tr>
<tr>
<td>-1958744010</td>
<td>0x8b3f036U</td>
<td>MASUnspecifiedDataType</td>
<td>Unspecified data type.</td>
</tr>
<tr>
<td>-1958744009</td>
<td>0x8b3f037U</td>
<td>MASTKThreadingError</td>
<td>Threading error.</td>
</tr>
<tr>
<td>-1958744008</td>
<td>0x8b3f038U</td>
<td>MASFatalProgRepoLost</td>
<td>Program repository lost.</td>
</tr>
<tr>
<td>-1958744007</td>
<td>0x8b3f039U</td>
<td>MASSaveToRepo</td>
<td>Failed to save to repository.</td>
</tr>
<tr>
<td>-1958744006</td>
<td>0x8b3f03aU</td>
<td>MASLog4SASCfgFailed</td>
<td>Logging configuration failed.</td>
</tr>
<tr>
<td>-1958744005</td>
<td>0x8b3f03bU</td>
<td>MASDS2CompileStart</td>
<td>User context '%s' compiling module '%s' on thread '%d'.</td>
</tr>
</tbody>
</table>
# Appendix 1 • SAS Micro Analytic Service Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Hexadecimal Code</th>
<th>#define Symbol</th>
<th>Message or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1958744004</td>
<td>0x8b3ff03cU</td>
<td>MASDS2CompileFinish</td>
<td>User context '%s' module '%s' thread %d compilation succeeded.</td>
</tr>
<tr>
<td>-1958744003</td>
<td>0x8b3ff03dU</td>
<td>MASDS2CompileFailed</td>
<td>User context '%s' module '%s' thread %d new revision failed, RC = %d.</td>
</tr>
<tr>
<td>-1958744002</td>
<td>0x8b3ff03eU</td>
<td>MASStartup</td>
<td>*** SAS Micro Analytic Service Started ***</td>
</tr>
<tr>
<td>-1958744001</td>
<td>0x8b3ff03fU</td>
<td>MASShutdown</td>
<td>*** Micro Analytic Service Shutting Down ***</td>
</tr>
<tr>
<td>-1958744000</td>
<td>0x8b3ff040U</td>
<td>MASAsyncException</td>
<td>SAS Micro Analytic Service received async exception code %d.</td>
</tr>
<tr>
<td>-1958743999</td>
<td>0x8b3ff041U</td>
<td>MASAsyncInitFailed</td>
<td>SAS Micro Analytic Service failed to install async exception handler.</td>
</tr>
<tr>
<td>-1958743998</td>
<td>0x8b3ff042U</td>
<td>MASShutdownJNI</td>
<td>SAS Micro Analytic Service calling JVM System.exit(0).</td>
</tr>
<tr>
<td>-1958743997</td>
<td>0x8b3ff043U</td>
<td>MASExecDeletePending</td>
<td>Attempt to execute method %s while deletion pending for module context %s revision %d.</td>
</tr>
<tr>
<td>-1958743996</td>
<td>0x8b3ff044U</td>
<td>MASMTXDeletePending</td>
<td>Attempt to add module context %s while deletion pending for user context %s.</td>
</tr>
<tr>
<td>-1958743995</td>
<td>0x8b3ff045U</td>
<td>MASRevDeletePending</td>
<td>Attempt to create revision while deletion pending for module context %s.</td>
</tr>
<tr>
<td>-1958743994</td>
<td>0x8b3ff046U</td>
<td>MASRevDelDeletePending</td>
<td>Attempt to delete revision while deletion pending for module context %s.</td>
</tr>
<tr>
<td>-1958743993</td>
<td>0x8b3ff047U</td>
<td>MASRevDelRefCount</td>
<td>Pending delete called for module context %s with ref count %d.</td>
</tr>
<tr>
<td>-1958743992</td>
<td>0x8b3ff048U</td>
<td>MASRevDelRefCountError</td>
<td>Delete called for module context %s with ref count %d.</td>
</tr>
<tr>
<td>-1958743991</td>
<td>0x8b3ff049U</td>
<td>MASMTXDelete</td>
<td>Garbage collection is deleting module context %s.</td>
</tr>
<tr>
<td>-1958743990</td>
<td>0x8b3ff04aU</td>
<td>MASCTXDeletePending</td>
<td>Attempt to delete user context %s while being deleted by another thread.</td>
</tr>
<tr>
<td>-1958743989</td>
<td>0x8b3ff04bU</td>
<td>MASCTXGetCDTDelPending</td>
<td>Attempt to retrieve creation time from user context %s while deletion pending.</td>
</tr>
<tr>
<td>-1958743988</td>
<td>0x8b3ff04cU</td>
<td>MASCTXGetMDTDelPending</td>
<td>Attempt to retrieve modified time from user context %s while deletion pending.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Hexadecimal Code</td>
<td><code>#define</code> Symbol</td>
<td>Message or Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-1958743987</td>
<td>0x8b3ff04dU</td>
<td>MASMTXGetCDTDelPending</td>
<td>Attempt to retrieve creation time from module context %s while deletion pending.</td>
</tr>
<tr>
<td>-1958743986</td>
<td>0x8b3ff04eU</td>
<td>MASMTXGetMDTDelPending</td>
<td>Attempt to retrieve modified time from module context %s while deletion pending.</td>
</tr>
<tr>
<td>-1958743985</td>
<td>0x8b3ff04fU</td>
<td>MASMTXGetRevDelPending</td>
<td>Attempt to retrieve highest revision from module context %s while deletion pending.</td>
</tr>
<tr>
<td>-1958743984</td>
<td>0x8b3ff050U</td>
<td>MASMTXGetIUODelPending</td>
<td>Attempt to retrieve internal use flag from module context %s while deletion pending.</td>
</tr>
<tr>
<td>-1958743983</td>
<td>0x8b3ff051U</td>
<td>MASRevGetCDTDelPending</td>
<td>Attempt to retrieve revision %d creation time from module context %s while deletion pending.</td>
</tr>
<tr>
<td>-1958743982</td>
<td>0x8b3ff052U</td>
<td>MASMTXGetMsgDelPending</td>
<td>Attempt to retrieve compilation messages from module context %s while deletion pending.</td>
</tr>
<tr>
<td>-1958743981</td>
<td>0x8b3ff053U</td>
<td>MASMTXRegDeletePending</td>
<td>Attempt to register name while deletion pending for module context %s.</td>
</tr>
<tr>
<td>-1958743980</td>
<td>0x8b3ff054U</td>
<td>MASMTXLangDelPending</td>
<td>Attempt to retrieve language of module context %s while deletion pending.</td>
</tr>
<tr>
<td>-1958743979</td>
<td>0x8b3ff055U</td>
<td>MASMTXGetDispDelPending</td>
<td>Attempt to retrieve display name from module context %s while deletion pending.</td>
</tr>
<tr>
<td>-1958743978</td>
<td>0x8b3ff056U</td>
<td>MASMTXGetCSrcDelPending</td>
<td>Attempt to retrieve C source code from module context %s revision %d while deletion pending.</td>
</tr>
<tr>
<td>-1958743977</td>
<td>0x8b3ff057U</td>
<td>MASCTXGetPkgsDelPending</td>
<td>Attempt to retrieve packages from user context %s while deletion pending.</td>
</tr>
<tr>
<td>-1958743976</td>
<td>0x8b3ff058U</td>
<td>MASMTXGetMthsDelPending</td>
<td>Attempt to retrieve methods from module context %s while deletion pending.</td>
</tr>
<tr>
<td>-1958743975</td>
<td>0x8b3ff059U</td>
<td>MASNoSuchEntryPoint</td>
<td>Entry point not found.</td>
</tr>
<tr>
<td>-1958743974</td>
<td>0x8b3ff05aU</td>
<td>MASMTXGetSigDelPending</td>
<td>Attempt to retrieve method %s signature from module context %s while deletion pending.</td>
</tr>
<tr>
<td>-1958743973</td>
<td>0x8b3ff05bU</td>
<td>MASCTXLdOOTBDelPending</td>
<td>Private load out-of-the-box packages for user context %s while deletion pending.</td>
</tr>
<tr>
<td>-1958743972</td>
<td>0x8b3ff05cU</td>
<td>MASCTXRegIntDelPending</td>
<td>Attempt to publish internal package %s to user context %s while deletion pending.</td>
</tr>
<tr>
<td>-1958743971</td>
<td>0x8b3ff05dU</td>
<td>MASCTXRemIntDelPending</td>
<td>Attempt to remove internal package %s from user context %s while deletion pending.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Hexadecimal Code</td>
<td>#define Symbol</td>
<td>Message or Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------</td>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-1958743970</td>
<td>0x8b3ff05fU</td>
<td>MASCreateGCAFailed</td>
<td>Attempt to create garbage collection control structures failed.</td>
</tr>
<tr>
<td>-1958743969</td>
<td>0x8b3ff05fU</td>
<td>MASGarbageCollection</td>
<td>Garbage collection interval.</td>
</tr>
<tr>
<td>-1958743968</td>
<td>0x8b3ff060U</td>
<td>MASGarbageCollectionDel</td>
<td>Garbage collection found assets ready to delete.</td>
</tr>
<tr>
<td>-1958743967</td>
<td>0x8b3ff061U</td>
<td>MASGCEXception</td>
<td>Exception occurred during garbage collection run.</td>
</tr>
<tr>
<td>-1958743966</td>
<td>0x8b3ff062U</td>
<td>MASProgRepoUpdateError</td>
<td>Error obtaining exclusive lock to update DS2 program repository.</td>
</tr>
<tr>
<td>-1958743965</td>
<td>0x8b3ff063U</td>
<td>MASCTXDelete</td>
<td>Garbage collection is deleting user context %s.</td>
</tr>
<tr>
<td>-1958743964</td>
<td>0x8b3ff064U</td>
<td>MASRevDelete</td>
<td>Garbage collection is deleting module context %s revision %d.</td>
</tr>
<tr>
<td>-1958743963</td>
<td>0x8b3ff065U</td>
<td>MASDS2Fatal</td>
<td>Module context %s revision %d generated fatal run-time exception. Deleting revision.</td>
</tr>
<tr>
<td>-1958743962</td>
<td>0x8b3ff066U</td>
<td>MASGarbageCollectionTerm</td>
<td>Garbage collection is freeing control assets during shut down.</td>
</tr>
<tr>
<td>-1958743961</td>
<td>0x8b3ff067U</td>
<td>MASShutdownHang</td>
<td>Worker thread did not interrupt after %d seconds during shutdown.</td>
</tr>
<tr>
<td>-1958743960</td>
<td>0x8b3ff068U</td>
<td>MASGCInvalidIntervalHigh</td>
<td>Specifies that the garbage collection interval is above the maximum. Setting to default value.</td>
</tr>
<tr>
<td>-1958743959</td>
<td>0x8b3ff069U</td>
<td>MASGCInvalidIntervalLow</td>
<td>Specifies that the garbage collection interval is below the minimum. Setting to default value.</td>
</tr>
<tr>
<td>-1958743958</td>
<td>0x8b3ff06aU</td>
<td>MASGCInvalidGraceHigh</td>
<td>Specifies that the grace period is above the maximum. Setting to default value.</td>
</tr>
<tr>
<td>-1958743957</td>
<td>0x8b3ff06bU</td>
<td>MASGCInvalidGraceLow</td>
<td>Specifies that the grace period is below the minimum. Setting to default value.</td>
</tr>
<tr>
<td>-1958743956</td>
<td>0x8b3ff06cU</td>
<td>MASGCMissingInterval</td>
<td>Garbage collection interval is not specified. Setting to default value.</td>
</tr>
<tr>
<td>-1958743955</td>
<td>0x8b3ff06dU</td>
<td>MASGCMissingGracePeriod</td>
<td>Grace period is not specified. Setting to default value.</td>
</tr>
<tr>
<td>-1958743954</td>
<td>0x8b3ff06eU</td>
<td>MASModuleStats</td>
<td>Check the log for module statistics.</td>
</tr>
<tr>
<td>-1958743953</td>
<td>0x8b3ff06fU</td>
<td>MASInvalidDS2Connection</td>
<td>Attempt to create TKTS driver connection failed.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Hexadecimal Code</td>
<td>#define Symbol</td>
<td>Message or Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------</td>
<td>---------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>-1958743952</td>
<td>0x8b3ff070U</td>
<td>MASDS2FatalRecompiled</td>
<td>DS2 package fatal error. Auto-recompile succeeded.</td>
</tr>
<tr>
<td>-1958743950</td>
<td>0x8b3ff072U</td>
<td>MASDS2RevisionEjected</td>
<td>DS2 package fatal error. Max retry exceeded. Ejecting revision. Correct and republish.</td>
</tr>
<tr>
<td>-1958743949</td>
<td>0x8b3ff073U</td>
<td>MASDBConnLost</td>
<td>Connection to the database lost. Check the log for details.</td>
</tr>
<tr>
<td>-1958743948</td>
<td>0x8b3ff074U</td>
<td>MASDBConnReestablished</td>
<td>Lost connection reestablished for user context.</td>
</tr>
<tr>
<td>-1958743947</td>
<td>0x8b3ff075U</td>
<td>MASDBConnRetryLimit</td>
<td>Maximum connection retry attempts exceeded for user context.</td>
</tr>
<tr>
<td>-1958743946</td>
<td>0x8b3ff076U</td>
<td>MASDBConnDoesNotExist</td>
<td>Attempt to execute SQLSTMT, when no connection exists.</td>
</tr>
<tr>
<td>-1958743945</td>
<td>0x8b3ff077U</td>
<td>MASDBConnRetryThreadErr</td>
<td>Error while creating database connection retry thread.</td>
</tr>
<tr>
<td>-1958743944</td>
<td>0x8b3ff078U</td>
<td>MASDBConnRetryAttempt</td>
<td>Connection retry attempt unsuccessful.</td>
</tr>
<tr>
<td>-1958743943</td>
<td>0x8b3ff079U</td>
<td>MASNameRegisterFailed</td>
<td>Unable to register tkmas in the threaded kernel named registry. DS2 programs that call Python scripts will not function.</td>
</tr>
<tr>
<td>-1958743942</td>
<td>0x8b3ff07aU</td>
<td>MASDS2PythonNameRequired</td>
<td>AS DS2 Python constructor missing Python module name.</td>
</tr>
<tr>
<td>-1958743941</td>
<td>0x8b3ff07bU</td>
<td>MASDS2PythonCreateError</td>
<td>Unable to create SAS Micro Analytic Service DS2 Python package.</td>
</tr>
<tr>
<td>-1958743940</td>
<td>0x8b3ff07cU</td>
<td>MASDS2PythonInitError</td>
<td>Unable to initialize support for SAS Micro Analytic Service DS2 Python package.</td>
</tr>
<tr>
<td>-1958743939</td>
<td>0x8b3ff07dU</td>
<td>MASUnsupportedFunction</td>
<td>Unsupported function.</td>
</tr>
<tr>
<td>-1958743938</td>
<td>0x8b3ff07eU</td>
<td>MASDS2NotInitialized</td>
<td>Attempt to perform action on uninitialized SAS Micro Analytic Service DS2 Python package.</td>
</tr>
<tr>
<td>-1958743937</td>
<td>0x8b3ff07fU</td>
<td>MASDS2PythonParmError</td>
<td>SAS Micro Analytic Service DS2 Python package parameter mismatch.</td>
</tr>
<tr>
<td>-1958743936</td>
<td>0x8b3ff080U</td>
<td>MASDS2PythonArgNameReqd</td>
<td>SAS Micro Analytic Service DS2 Python missing argument name.</td>
</tr>
<tr>
<td>-1958743935</td>
<td>0x8b3ff081U</td>
<td>MASDS2PythonArgValueReqd</td>
<td>AS DS2 Python missing argument value.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Hexadecimal Code</td>
<td>#define Symbol</td>
<td>Message or Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------</td>
<td>--------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-1958743934</td>
<td>0x8b3ff082U</td>
<td>MASDS2PythonArgInvalid</td>
<td>SAS Micro Analytic Service DS2 Python invalid argument value.</td>
</tr>
<tr>
<td>-1958743933</td>
<td>0x8b3ff083U</td>
<td>MASDS2PythonThreadError</td>
<td>Invalid operation: DS2 callback into SAS Micro Analytic Service received an unrecognized thread.</td>
</tr>
<tr>
<td>-1958743932</td>
<td>0x8b3ff084U</td>
<td>MASP\textbackslash{\texttt{Python_Compile_Ex}}</td>
<td>Exception thrown while initializing Python or compiling Python script.</td>
</tr>
<tr>
<td>-1958743931</td>
<td>0x8b3ff085U</td>
<td>MASDS2InvalidMaxRecomp</td>
<td>Invalid maximum DS2 recompile count given. Setting to default value.</td>
</tr>
<tr>
<td>-1958743930</td>
<td>0x8b3ff086U</td>
<td>MASDBInvalidIntervalHigh</td>
<td>Specified DBMS connection retry interval is above the maximum. Setting to default value.</td>
</tr>
<tr>
<td>-1958743929</td>
<td>0x8b3ff087U</td>
<td>MASDBInvalidIntervalLow</td>
<td>Specified DBMS connection retry interval is below the minimum. Setting to default value.</td>
</tr>
<tr>
<td>-1958743928</td>
<td>0x8b3ff088U</td>
<td>MASDBInvalidMaxRetry</td>
<td>Invalid setting for maximum DBMS reconnection attempts. Setting to default value.</td>
</tr>
<tr>
<td>-1958743927</td>
<td>0x8b3ff089U</td>
<td>MASC\textbackslash{\texttt{Connect_Err}}</td>
<td>SAS Micro Analytic Service failed to create a connection.</td>
</tr>
<tr>
<td>-1958743926</td>
<td>0x8b3ff08aU</td>
<td>MASC\textbackslash{\texttt{Connect}}</td>
<td>SAS Micro Analytic Service created a connection.</td>
</tr>
<tr>
<td>-1958743925</td>
<td>0x8b3ff08bU</td>
<td>MASC\textbackslash{\texttt{GC_CanBe_Deleted}}</td>
<td>Garbage collection is checking module context for deletion pending.</td>
</tr>
<tr>
<td>-1958743924</td>
<td>0x8b3ff08cU</td>
<td>MASC\textbackslash{\texttt{Repo_Lock_Remove_Priv}}</td>
<td>Locking program repository to remove internal package.</td>
</tr>
<tr>
<td>-1958743923</td>
<td>0x8b3ff08dU</td>
<td>MASC\textbackslash{\texttt{Unlock_Remove_Priv}}</td>
<td>Released program repository lock after removing internal package.</td>
</tr>
<tr>
<td>-1958743922</td>
<td>0x8b3ff08eU</td>
<td>MASC\textbackslash{\texttt{Repo_Lock_Remove_Rev}}</td>
<td>Locking program repository to remove module context.</td>
</tr>
<tr>
<td>-1958743921</td>
<td>0x8b3ff08fU</td>
<td>MASC\textbackslash{\texttt{Unlock_Remove_Rev}}</td>
<td>Released program repository lock, after removing module context.</td>
</tr>
<tr>
<td>-1958743920</td>
<td>0x8b3ff090U</td>
<td>MASC\textbackslash{\texttt{Lock_Create}}</td>
<td>Creating a lock for user context.</td>
</tr>
<tr>
<td>-1958743919</td>
<td>0x8b3ff091U</td>
<td>MASC\textbackslash{\texttt{Lock_Destroy}}</td>
<td>Destroying a lock for user context.</td>
</tr>
<tr>
<td>-1958743918</td>
<td>0x8b3ff092U</td>
<td>MASC\textbackslash{\texttt{Lock_Package_Comp}}</td>
<td>Locking program repository during compilation of package.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Hexadecimal Code</td>
<td>#define Symbol</td>
<td>Message or Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
<td>--------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>-1958743917</td>
<td>0x8b3ff093U</td>
<td>MASRepoUnlockPackageComp</td>
<td>Released program repository lock after compilation of package.</td>
</tr>
<tr>
<td>-1958743916</td>
<td>0x8b3ff094U</td>
<td>MASRepoUnlockCompCrash</td>
<td>Released program repository lock due to DS2 compiler crash while compiling package.</td>
</tr>
<tr>
<td>-1958743915</td>
<td>0x8b3ff095U</td>
<td>MASRepoLockPackageSave</td>
<td>Locking program repository to save package after successful compilation.</td>
</tr>
<tr>
<td>-1958743914</td>
<td>0x8b3ff096U</td>
<td>MASRepoUnlockPackageSave</td>
<td>Released program repository after saving package.</td>
</tr>
<tr>
<td>-1958743913</td>
<td>0x8b3ff097U</td>
<td>MASRepoLockPackagePriv</td>
<td>Locking program repository to save internal package.</td>
</tr>
<tr>
<td>-1958743912</td>
<td>0x8b3ff098U</td>
<td>MASRepoUnlockPackagePriv</td>
<td>Released program repository after saving internal package.</td>
</tr>
<tr>
<td>-1958743911</td>
<td>0x8b3ff099U</td>
<td>MASPythonNotLoaded</td>
<td>Python extension not loaded. Python must be installed in order to execute Python within SAS Micro Analytic Service.</td>
</tr>
<tr>
<td>-1958743910</td>
<td>0x8b3ff09aU</td>
<td>MASTKTSConnHndlFail</td>
<td>Failed to create a table services connection handle.</td>
</tr>
<tr>
<td>-1958743909</td>
<td>0x8b3ff09bU</td>
<td>MASDBDisconnected</td>
<td>SAS Micro Analytic Service disconnected database from user context.</td>
</tr>
<tr>
<td>-1958743908</td>
<td>0x8b3ff09cU</td>
<td>MASDBDisconnect</td>
<td>SAS Micro Analytic Service encountered a failure when attempting to disconnect the database from the user context.</td>
</tr>
<tr>
<td>-1958743907</td>
<td>0x8b3ff09dU</td>
<td>MASPercentS</td>
<td>Internal error. Check the SAS Micro Analytic Service Core log.</td>
</tr>
<tr>
<td>-1958743906</td>
<td>0x8b3ff09eU</td>
<td>MASPythonCompileErr</td>
<td>Error compiling the Python script for the module.</td>
</tr>
<tr>
<td>-1958743905</td>
<td>0x8b3ff09fU</td>
<td>MASDS2MissingArray</td>
<td>A missing array argument is not supported with DS2.</td>
</tr>
<tr>
<td>-1958743904</td>
<td>0x8b3ff0a0U</td>
<td>MASDS2EmptyArray</td>
<td>An empty array argument is not supported with DS2.</td>
</tr>
<tr>
<td>-1958743903</td>
<td>0x8b3ff0a1U</td>
<td>MASDS2ArrayReplaced</td>
<td>Missing or insufficiently sized DS2 array argument has been replaced with new array of size %d.</td>
</tr>
<tr>
<td>-1958743902</td>
<td>0x8b3ff0a2U</td>
<td>MASDS2OutputTransError</td>
<td>Error %d when converting CHAR string of length %d to TKChar string.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Hexadecimal Code</td>
<td>#define Symbol</td>
<td>Message or Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------</td>
<td>---------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>-1958743901</td>
<td>0x8b3ff0a3U</td>
<td>MASDS2InputTransError</td>
<td>Error %d when converting TKChar string of length %d to CHAR string.</td>
</tr>
<tr>
<td>-1958743900</td>
<td>0x8b3ff0a4U</td>
<td>MASDS2PythonOutputTrans</td>
<td>Error %d when converting Python CHAR string of length %d to TKChar string.</td>
</tr>
<tr>
<td>-1958743899</td>
<td>0x8b3ff0a5U</td>
<td>MASDS2PythonInputTrans</td>
<td>Error %d when converting TKChar string of length %d to CHAR string for Python.</td>
</tr>
<tr>
<td>-1958743898</td>
<td>0x8b3ff0a6U</td>
<td>MASDBCr8ConnNoSub</td>
<td>SAS Micro Analytic Service created a default data source connection.</td>
</tr>
<tr>
<td>-1958743897</td>
<td>0x8b3ff0a7U</td>
<td>MASDBCr8ConnErrNoSub</td>
<td>SAS Micro Analytic Service failed to create a default data source connection.</td>
</tr>
<tr>
<td>1958743896</td>
<td>0x8b3ff0a8U</td>
<td>MASBDisconnNoSub</td>
<td>SAS Micro Analytic Service disconnected from the default data source.</td>
</tr>
<tr>
<td>1958743895</td>
<td>0x8b3ff0a9U</td>
<td>MASBDisconnErrNoSub</td>
<td>SAS Micro Analytic Service encountered a failure when attempting to disconnect from the default data source.</td>
</tr>
<tr>
<td>1958743894</td>
<td>0x8b3ff0aaU</td>
<td>MASDS2ScanError</td>
<td>Out of memory or malformed DS2 encountered while scanning the package %s source code prior to dictionary generation.</td>
</tr>
<tr>
<td>1958743893</td>
<td>0x8b3ff0abU</td>
<td>MASDS2ParseError</td>
<td>Out of memory or malformed DS2 encountered while parsing the package %s method %s during dictionary generation.</td>
</tr>
<tr>
<td>1958743892</td>
<td>0x8b3ff0acU</td>
<td>MASMTXGetDictDelPending</td>
<td>Attempt to retrieve the dictionary from module context %s revision %d while deletion pending.</td>
</tr>
<tr>
<td>1958743892</td>
<td>0x8b3ff0adU</td>
<td>MASCFuncProtoNotSupp</td>
<td>Part of the C function prototype is not supported.</td>
</tr>
<tr>
<td>1958743890</td>
<td>0x8b3ff0aeU</td>
<td>M ASDupModuleName</td>
<td>Module name %s already exists. Module name must be unique within the user context.</td>
</tr>
<tr>
<td>1958743889</td>
<td>0x8b3ff0afU</td>
<td>M ASDupDS2Package</td>
<td>The DS2 package name %s is already bound to module %s. Separate modules cannot represent the same DS2 package.</td>
</tr>
<tr>
<td>1958743888</td>
<td>0x8b3ff0b0U</td>
<td>M ASDIndexOutOfRangeSet</td>
<td>The index is out of range while setting an argument. Argument %d specified when number of arguments is %d.</td>
</tr>
<tr>
<td>1958743887</td>
<td>0x8b3ff0b1U</td>
<td>M ASDIndexOutOfRangeGet</td>
<td>The index is out of range while retrieving an argument. Argument %d specified when number of arguments is %d.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Hexadecimal Code</td>
<td>#define Symbol</td>
<td>Message or Description</td>
</tr>
<tr>
<td>------------</td>
<td>------------------</td>
<td>------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1958743886</td>
<td>0x8b3ff0b2U</td>
<td>MASIntTypeExpected</td>
<td>The argument <code>%d</code> in method <code>%Us</code> should be an integral type used to specify the length of the previous argument, which is an array.</td>
</tr>
<tr>
<td>1958743885</td>
<td>0x8b3ff0b3U</td>
<td>MASOutArgExpected</td>
<td>The argument <code>%d</code> in method <code>%Us</code> should be an output argument. All input arguments must precede output arguments.</td>
</tr>
<tr>
<td>1958743884</td>
<td>0x8b3ff0b4U</td>
<td>MASDS2pymas</td>
<td>DS2 PyMAS package encountered a failure.</td>
</tr>
<tr>
<td>1958743883</td>
<td>0x8b3ff0b5U</td>
<td>MASDS2pymasFailIn</td>
<td>DS2 PyMAS package encountered a failure in <code>%Us</code>.</td>
</tr>
<tr>
<td>1958743882</td>
<td>0x8b3ff0b6U</td>
<td>MASDS2pymasPubUTF8</td>
<td>DS2 PyMAS package failed to publish module <code>%Us</code>.</td>
</tr>
<tr>
<td>1958743881</td>
<td>0x8b3ff0b7U</td>
<td>MASDS2pymasPubTK</td>
<td>DS2 PyMAS package failed to publish module <code>%s</code>.</td>
</tr>
<tr>
<td>1958743880</td>
<td>0x8b3ff0b8U</td>
<td>MASDS2pymasUsed</td>
<td>The DS2 PyMAS package's use method has already been called on this package instance. Create a separate PyMAS instances for each method that is used.</td>
</tr>
<tr>
<td>1958743879</td>
<td>0x8b3ff0b9U</td>
<td>MASThrdPoolSizeDiff</td>
<td>SAS Micro Analytic Service has already been initialized with a worker thread pool size of <code>%d</code>.</td>
</tr>
<tr>
<td>1958743878</td>
<td>0x8b3ff0baU</td>
<td>MASSymbolTableCreateFailed</td>
<td>SAS Micro Analytic Service failed to create a symbol table.</td>
</tr>
<tr>
<td>1958743877</td>
<td>0x8b3ff0bbU</td>
<td>MASMethodExecutionFailed</td>
<td>SAS Micro Analytic Service failed to execute a method.</td>
</tr>
</tbody>
</table>
## 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.

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Error Message</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| 400       | Module module\_ID not found. | A module with the specified ID does not exist. This typically occurs when attempting to do one of the following for the specified module:  
- view, update, delete the module  
- execute a step of the module  
- validate data input for the execution of a step of the module | Specify a module ID that exists in the system. |
| 404       | Step step\_ID for module module\_ID not found. | The specified module exists, but the referenced step was not found. This typically occurs when attempting to do one of the following:  
- access the specified module to execute a step  
- validate data input for the execution of a step | Specify a step ID that exists in the specified module ID. |

### Errors related to missing resources

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Error Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Cannot create or update module. Media type type_name is not supported. type_name is the only supported media type.</td>
<td>The media type must be text/vnd.sas.source.ds2 or text/application.source.ds2.</td>
</tr>
<tr>
<td>400</td>
<td>Cannot create or update module. DS2 is the only supported language at this time.</td>
<td>The programming language is not DS2.</td>
</tr>
<tr>
<td>HTTP Code</td>
<td>Error Message</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>400</td>
<td>Cannot create or update module. The supplied module ID may not be empty or missing.</td>
<td>Under certain circumstances, it is possible to specify the module ID as part of the module definition. However, when doing so the module specification must not be an empty string or missing.</td>
</tr>
<tr>
<td>400</td>
<td>Cannot create or update module. Unable to parse package name.</td>
<td>The DS2 package must be properly formed so that the package name can be parsed.</td>
</tr>
<tr>
<td>400</td>
<td>Cannot create or update module. Value of field <em>field_name</em> may not be empty or missing.</td>
<td>When creating or updating a module, the payload contains certain mandatory fields, for example, scope, type, and code. Mandatory fields cannot be missing and the associated content must not be null or an empty string.</td>
</tr>
<tr>
<td>400</td>
<td>Cannot create or update module. Field <em>field_name</em> may not be repeated.</td>
<td>When creating or updating a module, the payload can contain multiple name value pairs called properties. Property names must be unique.</td>
</tr>
<tr>
<td>400</td>
<td>Cannot create or update module. Unexpected end of source while parsing comment.</td>
<td>DS2 code that contains comments must be properly delimited.</td>
</tr>
<tr>
<td>400</td>
<td>Cannot create or update module. Unmatched single quotes.</td>
<td>If strings are defined in the DS2 program, they must be enclosed in single quotation marks.</td>
</tr>
<tr>
<td>400</td>
<td>Cannot create or update module. Unmatched double quotes.</td>
<td>If delimited variables names are defined in the DS2 program, they must be enclosed in matching double quotation marks.</td>
</tr>
<tr>
<td>400</td>
<td>Cannot create or update module. No DS2 package found in source.</td>
<td>DS2 source was not found.</td>
</tr>
<tr>
<td>400</td>
<td>Cannot create or update module. More than one DS2 package found in source.</td>
<td>Only a single DS2 package is accepted to create a module.</td>
</tr>
<tr>
<td>HTTP Code</td>
<td>Error Message</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>400</td>
<td>Cannot create or update module module_ID. Compilation errors detected: error_messages.</td>
<td>The DS2 source code contains compilation errors.</td>
</tr>
<tr>
<td>400</td>
<td>Cannot create module. The supplied module ID module_ID is already in use.</td>
<td>When creating a module, the specified module ID must not already exist in the system.</td>
</tr>
<tr>
<td>400</td>
<td>Cannot execute step step_ID of module module_ID.</td>
<td>The input data supplied to execute a module step is incorrect. Subsequent error messages will provide further details.</td>
</tr>
<tr>
<td>403</td>
<td>Cannot access step(s) of PRIVATE module module_ID.</td>
<td>The steps of private modules are not accessible.</td>
</tr>
<tr>
<td>400</td>
<td>Expected integer_value, but received integer_value inputs.</td>
<td>An incorrect number of parameters were supplied to execute a module step.</td>
</tr>
<tr>
<td>400</td>
<td>Parameter parameter_name is not defined.</td>
<td>The named parameter is not defined for the module step.</td>
</tr>
<tr>
<td>400</td>
<td>Cannot assign specified_value to parameter parameter_name of type type.</td>
<td>An inappropriate value was received for the parameter, for example, supplying a string value to an integer parameter.</td>
</tr>
<tr>
<td>400</td>
<td>Cannot assign specified_value to array parameter parameter_name [element_number] of type type.</td>
<td>An inappropriate value was received for the array parameter, for example, supplying a string value to an integer array parameter.</td>
</tr>
<tr>
<td>400</td>
<td>Value of field field_name may not be empty or missing.</td>
<td>The name and value parts of the parameter must be available to execute the module step.</td>
</tr>
<tr>
<td>400</td>
<td>Field field_name may not be repeated.</td>
<td>You cannot specify more than one value for a parameter.</td>
</tr>
<tr>
<td>HTTP Code</td>
<td>Error Message</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>500</td>
<td>Error <code>error_message</code> executing step <code>step_ID</code> of module <code>module_ID</code>.</td>
<td>An error was returned by the SAS Micro Analytic Service core during execution of the modules.</td>
</tr>
</tbody>
</table>

**Internal Errors**

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Error Message</th>
<th>Explanation</th>
<th>Remedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>Internal Error: DS2 Compiler crashed while compiling module. Please check log files for further diagnosis.</td>
<td>In most cases, this error is caused by insufficient stack space for the SAS Micro Analytic Service core TK subsystem.</td>
<td>In SAS Environment Manager, increase the value of the configuration parameter <code>core.tktstacksizekbytes</code>, as appropriate.</td>
</tr>
<tr>
<td>500</td>
<td>An internal error occurred. Please check log files for further diagnosis.</td>
<td></td>
<td>Please contact SAS Technical Support.</td>
</tr>
</tbody>
</table>
# Appendix 3
## Table Service Driver Reference

<table>
<thead>
<tr>
<th>Driver Reference</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DB2 Driver Reference</strong></td>
<td>98</td>
</tr>
<tr>
<td>Understanding the Table Services Driver for DB2</td>
<td>98</td>
</tr>
<tr>
<td>Data Service Connection Options for DB2</td>
<td>99</td>
</tr>
<tr>
<td>DB2 Wire Protocol Driver Usage Notes</td>
<td>102</td>
</tr>
<tr>
<td><strong>FedSQL Driver Reference</strong></td>
<td>103</td>
</tr>
<tr>
<td>Overview</td>
<td>103</td>
</tr>
<tr>
<td>Connection Options</td>
<td>103</td>
</tr>
<tr>
<td><strong>Greenplum Driver Reference</strong></td>
<td>106</td>
</tr>
<tr>
<td>Understanding the Table Services Driver for Greenplum</td>
<td>106</td>
</tr>
<tr>
<td>Data Service Connection Options for Greenplum</td>
<td>106</td>
</tr>
<tr>
<td>Greenplum Wire Protocol Driver Usage Notes</td>
<td>110</td>
</tr>
<tr>
<td><strong>ODBC Driver Reference</strong></td>
<td>110</td>
</tr>
<tr>
<td>About ODBC</td>
<td>110</td>
</tr>
<tr>
<td>Understanding the Table Services Driver for ODBC</td>
<td>111</td>
</tr>
<tr>
<td>Data Service Connection Options for ODBC</td>
<td>111</td>
</tr>
<tr>
<td>Wire Protocol Driver Usage Notes</td>
<td>117</td>
</tr>
<tr>
<td><strong>Oracle Reference</strong></td>
<td>118</td>
</tr>
<tr>
<td>Understanding the Table Services Driver for Oracle</td>
<td>118</td>
</tr>
<tr>
<td>Data Service Connection Options for Oracle</td>
<td>118</td>
</tr>
<tr>
<td>Oracle Wire Protocol Driver Usage Notes</td>
<td>124</td>
</tr>
<tr>
<td><strong>PostgreSQL Driver Reference</strong></td>
<td>124</td>
</tr>
<tr>
<td>Understanding the SAS Federation Server Driver for PostgreSQL</td>
<td>124</td>
</tr>
<tr>
<td>Data Service Connection Options for PostgreSQL</td>
<td>125</td>
</tr>
<tr>
<td><strong>SAS Data Set Reference</strong></td>
<td>129</td>
</tr>
<tr>
<td>Overview</td>
<td>129</td>
</tr>
<tr>
<td>Understanding the Driver for Base SAS</td>
<td>129</td>
</tr>
<tr>
<td>Data Service Connection Options for SAS Data Sets</td>
<td>129</td>
</tr>
<tr>
<td><strong>Teradata Reference</strong></td>
<td>133</td>
</tr>
<tr>
<td>Understanding the Table Services Driver for Teradata</td>
<td>133</td>
</tr>
<tr>
<td>Data Service Connection Options for Teradata</td>
<td>133</td>
</tr>
</tbody>
</table>
Understanding the Table Services Driver for DB2

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

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

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

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

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

AIX
Bourne Shell
$ LIBPATH=$INSTHOME/lib:$LIBPATH
$ export LIBPATH
C Shell
$ setenv LIBPATH $INSTHOME/lib:$LIBPATH
HP-UX and HP-UX for the Itanium Processor
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 Viya: FedSQL Programming for SAS Cloud Analytic Services*.

**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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATALOG</td>
<td><code>CATALOG=catalog-identifier;</code></td>
</tr>
<tr>
<td></td>
<td>Specifies an arbitrary identifier for an SQL catalog, which groups logically related schemas. Any identifier is valid (for example, <code>catalog=DB2</code>). You must specify a catalog. For the DB2 database, this is a logical catalog name to use as an SQL catalog identifier.</td>
</tr>
<tr>
<td></td>
<td><em>Note:</em> The FedSQL language processor automatically quotes SQL identifiers that do not meet the regular naming convention as defined in <em>SAS Viya: FedSQL Programming for SAS Cloud Analytic Services</em>.</td>
</tr>
<tr>
<td>DATABASE</td>
<td><code>DATABASE=database-specification;</code></td>
</tr>
<tr>
<td></td>
<td>Specifies the name of the DB2 database (for example, <code>database=sample</code>, <code>DB=sample</code>).</td>
</tr>
<tr>
<td></td>
<td><em>Note:</em> You must specify a database name.</td>
</tr>
<tr>
<td>DRIVER</td>
<td><code>DRIVER=DB2;</code></td>
</tr>
<tr>
<td></td>
<td>Identifies the DB2 data source to which you want to connect.</td>
</tr>
<tr>
<td></td>
<td><em>Note:</em> You must specify the driver.</td>
</tr>
</tbody>
</table>

---

**DB2 Driver Reference**  
99
Advanced Connection Options

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLIENT_ENCODING=</td>
<td>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.</td>
</tr>
<tr>
<td>CT_PRESERVE</td>
<td>CT_PRESERVE=STRICT</td>
</tr>
</tbody>
</table>

Note: Setting the DB2CODEPAGE value or the CLIENT_ENCODING= value incorrectly can cause unpredictable results. You should set these values only when a situation such as the example above occurs.

Note: You can specify any valid encoding value for CLIENT_ENCODING=option.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| **DEFAULT_ATTR** | **DEFAULT_ATTR=(attr=value;...)**  
Used to specify connection handle or statement handle attributes that are supported for initial connect-time configuration, where \texttt{attr=value} corresponds to any of the following options:  
* **CURSORS=n** - Connection handle option. This option controls the driver’s use of client-side, result set cursors. The possible values are 0, 1, or 2.  
  0 Causes the driver to use client-side static cursor emulation if a scrollable cursor is requested but the database server cannot provide one.  
  1 Causes the driver to always use client-side static cursor emulation if a scrollable cursor is requested. The database server’s native cursor is not used.  
  2 (Default) Causes the driver to never use client-side static cursor emulation if a scrollable cursor is requested. The database server’s native cursor is used if available. Otherwise, the cursor is forward-only.  
Example: DEFAULT\_ATTR=(CURSORS=2)  
* **USE\_EVP=n** - Statement handle option. This option optimizes the driver for large result sets. The possible values are 0 (OFF) or 1 (ON), which is the default. Example: DEFAULT\_ATTR=(USE\_EVP=0)  
* **XCODE\_WARN=n** - Statement handle option. Used to warn about possible character transcoding errors that occur during row input or output operations. Possible values are 0 (returns an error), 1 (returns a warning), or 2 (ignore transaction errors). 0 is the default. Example: DEFAULT\_ATTR=(XCODE\_WARN=1) |

| **DRIVER\_TRACE** | **DRIVER\_TRACE='API | SQL | ALL';**  
Requests tracing information, which logs transaction records to an external file that can be used for debugging purposes. The driver writes a record of each command that is sent to the database to the trace log based on the specified tracing level, which determines the type of tracing information. Here are the tracing levels:  
* **API** Specifies that API method calls be sent to the trace log. This option is most useful if you are having a problem and need to send a trace log to SAS Technical Support for troubleshooting.  
* **SQL** Specifies that SQL statements that are sent to the database management system (DBMS) be sent to the trace log. Tracing information is DBMS specific, but most table services drivers log SQL statements such as SELECT and COMMIT.  
* **ALL** Activates all trace levels.  
* **DRIVER** Specifies that driver-specific information be sent to the trace log.  
Default: Tracing is not activated.  
Note: If you activate tracing, you must also specify the location of the trace log with DRIVER\_TRACEFILE=. Note that DRIVER\_TRACEFILE= is resolved against the TRACEFILEPATH set in ALTER SERVER. TRACEFILEPATH is relative to the server’s content root location.  
(Optional) You can control trace log formatting with DRIVER\_TRACEOPTIONS=.  
Interaction: You can specify one trace level, or you can concatenate more than one by including the | (OR) symbol. For example, driver\_trace='api|sql' generates tracing information for API calls and SQL statements. |
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRIVER_TRACEFILE</td>
<td>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’).</td>
</tr>
<tr>
<td></td>
<td><strong>Default:</strong> The default TRACEFILE location applies to a relative filename, and it is placed relative to TRACEFILEPATH.</td>
</tr>
<tr>
<td></td>
<td><strong>Requirement:</strong> DRIVER_TRACEFILE is required when activating tracing using DRIVER_TRACE.</td>
</tr>
<tr>
<td></td>
<td><strong>Interaction:</strong> (Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=.</td>
</tr>
<tr>
<td>DRIVER_TRACEOPTIONS</td>
<td>DRIVER_TRACEOPTIONS=APPEND</td>
</tr>
<tr>
<td></td>
<td>• APPEND Adds trace information to the end of an existing trace log. The contents of the file are not overwritten.</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMP Prepends each line of the trace log with a time stamp.</td>
</tr>
<tr>
<td></td>
<td>• THREADSTAMP Prepends each line of the trace log with a thread identification.</td>
</tr>
<tr>
<td></td>
<td><strong>Default:</strong> The trace log is overwritten with no thread identification or time stamp.</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>PWD=password</td>
</tr>
<tr>
<td></td>
<td>Specifies the password for DB2.</td>
</tr>
<tr>
<td>UID</td>
<td>UID=user-id;</td>
</tr>
<tr>
<td></td>
<td>Specifies the DB2 login user ID.</td>
</tr>
</tbody>
</table>

**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:
FedSQL Driver Reference

Overview

The FedSQL language driver supports the FedSQL dialect, as documented in *SAS Viya: FedSQL Programming for SAS Cloud Analytic Services*. 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 that 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:

```bash
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:

```bash
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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATALOG</td>
<td>CATALOG=catalog-identifier;</td>
</tr>
</tbody>
</table>

Specifies an arbitrary identifier for an SQL catalog, which groups logically related schemas. Any identifier is valid (for example, catalog=gps_test). You must specify a catalog. For the Greenplum database, this is a logical catalog name to use as an SQL catalog identifier.

*Note:* SAS Federation Server automatically quotes SQL identifiers that do not meet the regular naming convention as defined in *SAS Viya: FedSQL Programming for SAS Cloud Analytic Services*. 
### 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_identifier;`  
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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 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. |
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_ATTR</td>
<td>DEFAULT_ATTR=(attr=value,...)</td>
</tr>
<tr>
<td></td>
<td>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:</td>
</tr>
<tr>
<td>CURSORS=n</td>
<td>Connection handle option. This option controls the driver’s use of client-side, result set cursors. The possible values are 0, 1, or 2.</td>
</tr>
<tr>
<td>0</td>
<td>Causes the driver to use client-side static cursor emulation if a scrollable cursor is requested but the database server cannot provide one.</td>
</tr>
<tr>
<td>1</td>
<td>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.</td>
</tr>
<tr>
<td>2</td>
<td>(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.</td>
</tr>
<tr>
<td>USE_EVP=n</td>
<td>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)</td>
</tr>
<tr>
<td>XCODE_WARN=n</td>
<td>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)</td>
</tr>
<tr>
<td>DRIVER_TRACE</td>
<td>DRIVER_TRACE='API</td>
</tr>
<tr>
<td></td>
<td>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:</td>
</tr>
<tr>
<td>ALL</td>
<td>Activates all trace levels.</td>
</tr>
<tr>
<td>API</td>
<td>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.</td>
</tr>
<tr>
<td>DRIVER</td>
<td>Specifies that driver-specific information be sent to the trace log.</td>
</tr>
<tr>
<td>SQL</td>
<td>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.</td>
</tr>
<tr>
<td>Default:</td>
<td>Tracing is not activated.</td>
</tr>
<tr>
<td>Note:</td>
<td>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.</td>
</tr>
<tr>
<td>(Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=.</td>
<td></td>
</tr>
<tr>
<td>Interaction:</td>
<td>You can specify one trace level, or you can concatenate more than one by including the</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DRIVER_TRACEFILE</td>
<td><code>DRIVER_TRACEFILE='filename'</code>; 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, <code>driver_tracefile='mytrace.log'</code>).</td>
</tr>
<tr>
<td></td>
<td><strong>Default</strong>: The default TRACEFILE location applies to a relative filename, and it is placed relative to TRACEFILEPATH. <strong>Requirement</strong>: DRIVER_TRACEFILE is required when activating tracing using DRIVER_TRACE. <strong>Interaction</strong>: (Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=.</td>
</tr>
<tr>
<td>DRIVER_TRACEOPTIONS</td>
<td>`DRIVER_TRACEOPTIONS=APPEND</td>
</tr>
<tr>
<td></td>
<td>• <strong>APPEND</strong> Adds trace information to the end of an existing trace log. The contents of the file are not overwritten.</td>
</tr>
<tr>
<td></td>
<td>• <strong>THREADSTAMP</strong> Prepends each line of the trace log with a thread identification.</td>
</tr>
<tr>
<td></td>
<td>• <strong>TIMESTAMP</strong> Prepends each line of the trace log with a time stamp.</td>
</tr>
<tr>
<td></td>
<td><strong>Default</strong>: The trace log is overwritten with no thread identification or time stamp.</td>
</tr>
<tr>
<td>MAX_BINARY_LEN</td>
<td><code>MAX_BINARY_LEN=value;</code> 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.</td>
</tr>
<tr>
<td>MAX_CHAR_LEN</td>
<td><code>MAX_CHAR_LEN=value;</code> 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.</td>
</tr>
<tr>
<td>MAX_TEXT_LEN</td>
<td><code>MAX_TEXT_LEN=value;</code> 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.</td>
</tr>
<tr>
<td>NUMBYTESPERCHAR</td>
<td><code>NUMBYTESPERCHAR=value;</code> Specifies the default number of bytes per character.</td>
</tr>
</tbody>
</table>
| PASSWORD          | `PASSWORD=password;` Specifies a password for the ID passed through the USER= option. The alias is PWD=.
|                  | **Note**: You must specify the PASSWORD= option. |
| SCHEMA            | `SCHEMA=value;` Specifies the default schema for the connection. If the option is not specified, the schema (or list of schemas) is determined based on the value of the schema search path defined on the database server. |
| STRIP_BLANKS      | `STRIP_BLANKS=value;` Specifies whether to strip blanks from character fields. |
Option | Description
--- | ---
USER | USER=user-id;

Specifies a Greenplum user ID. If the ID contains blanks or national characters, enclose it in quotation marks. The alias is UID=.

*Note:* You must specify the USER= option.

**Greenplum Wire Protocol Driver Usage Notes**

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

### Windows
The options are located on the **Advanced** or **Performance** tabs in the ODBC Administrator.

### UNIX
The options are available when configuring data sources using the ODBC Administrator tool. Values can also be set by editing the `odbc.ini` file in which their data sources are defined.

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

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

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

---

**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.

```bash
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:

```bash
driver=sql;conopts=(driver=odbc;
catalog=acat;conopts=(dsn=ODBCPgresDD;pwd=Tester2))
```
The driver for ODBC supports the following connection options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CATALOG</strong></td>
<td><code>CATALOG=catalog-identifier;</code>&lt;br&gt;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, <code>catalog=myodbc</code>). For databases like Microsoft SQL Server that do support native catalogs, <code>CATALOG=</code> is not required. The connection defaults to <code>CATALOG=*</code> 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 <code>mycat</code> to the native catalog named <code>newusers</code>, use the following command: <code>catalog=(mycat=newusers);</code>. Catalog name maps can be used only with FedSQL. They are not valid with native SQL.&lt;br&gt;&lt;br&gt;Note: The FedSQL language processor automatically quotes SQL identifiers that do not meet the regular naming convention as defined in <em>SAS Viya: FedSQL Programming for SAS Cloud Analytic Services</em>.</td>
</tr>
</tbody>
</table>
| **CONOPTS** | `CONOPTS=(ODBC–compliant database connection string);`<br>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 `FILESN=` 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.<br><br>Here is an example string using the `CONOPTS` option:<br><br>`driver=sql;conopts=((driver=odbc;catalog=acat;conopts=(dsn=ODBCPgresDD;pwd=Tester2));
(driver=postgres;catalog=bcat;uid=myuid;pwd='123pass';
server=sv.abc.123.com;port=5432;DB=mydb;schema=public))` |
| **DRIVER**  | `DRIVER=ODBC;`<br>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.<br><br>Note: `DRIVER` is a required option. You must specify the driver. |
| **ODBC_DSN** | `ODBC_DSN=odbc dsn name`<br>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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT_PRESERVE</td>
<td>**CT_PRESERVE = STRICT</td>
</tr>
<tr>
<td>ENABLE_MARS</td>
<td>**ENABLE_MARS = NO</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| 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)** |
<p>|              | • USE_EVP=<em>n</em> - 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: <strong>DEFAULT_ATTR=(USE_EVP=0)</strong> |
|              | • XCODE_WARN=<em>n</em> - 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: <strong>DEFAULT_ATTR=(XCODE_WARN=1)</strong> |</p>
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_CURSOR_TYPE=FORWARD_ONLY</td>
<td>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.</td>
</tr>
<tr>
<td>KEYSET_DRIVEN</td>
<td>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.</td>
</tr>
<tr>
<td>DYNAMIC</td>
<td>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.</td>
</tr>
<tr>
<td>STATIC</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRIVER_TRACE</td>
<td>\texttt{DRIVER_TRACE='API</td>
</tr>
<tr>
<td></td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>• \texttt{ALL} Activates all trace levels.</td>
</tr>
<tr>
<td></td>
<td>• \texttt{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.</td>
</tr>
<tr>
<td></td>
<td>• \texttt{DRIVER} Specifies that driver-specific information be sent to the trace log.</td>
</tr>
<tr>
<td></td>
<td>• \texttt{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.</td>
</tr>
<tr>
<td></td>
<td>\textbf{Default:} Tracing is not activated.</td>
</tr>
<tr>
<td></td>
<td>\textit{Note:} If you activate tracing, you must also specify the location of the trace log with \texttt{DRIVER_TRACEFILE=}. Note that \texttt{DRIVER_TRACEFILE=} is resolved against the TRACFILEPATH set in ALTER SERVER. TRACFILEPATH is relative to the server's content root location.</td>
</tr>
<tr>
<td></td>
<td>(Optional) You can control trace log formatting with \texttt{DRIVER_TRACEOPTIONS=}.</td>
</tr>
<tr>
<td></td>
<td>\textbf{Interaction:} You can specify one trace level, or you can concatenate more than one by including the</td>
</tr>
<tr>
<td>DRIVER_TRACEFILE</td>
<td>\texttt{DRIVER_TRACEFILE='filename';}</td>
</tr>
<tr>
<td></td>
<td>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, \texttt{driver_tracefile='\mytrace.log'}).</td>
</tr>
<tr>
<td></td>
<td>\textbf{Default:} The default TRACEFILE location applies to a relative filename, and it is placed relative to TRACFILEPATH.</td>
</tr>
<tr>
<td></td>
<td>\textbf{Requirement:} DRIVER_TRACEFILE is required when activating tracing using DRIVER_TRACE.</td>
</tr>
<tr>
<td></td>
<td>\textbf{Interaction:} (Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=.</td>
</tr>
<tr>
<td>DRIVER_TRACEOPTIONS</td>
<td>\texttt{DRIVER_TRACEOPTIONS=APPEND</td>
</tr>
<tr>
<td></td>
<td>Specifies options in order to control formatting and other properties for the trace log:</td>
</tr>
<tr>
<td></td>
<td>• \texttt{APPEND} Adds trace information to the end of an existing trace log. The contents of the file are not overwritten.</td>
</tr>
<tr>
<td></td>
<td>• \texttt{THREADSTAMP} Prepends each line of the trace log with a thread identification.</td>
</tr>
<tr>
<td></td>
<td>• \texttt{TIMESTAMP} Prepends each line of the trace log with a time stamp.</td>
</tr>
<tr>
<td></td>
<td>\textbf{Default:} The trace log is overwritten with no thread identification or time stamp.</td>
</tr>
<tr>
<td>USER</td>
<td>\texttt{USER=user-ID;}</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
|                  | \textit{Note:} The alias is \texttt{UID=}.

---

116 Appendix 3 • Table Service Driver Reference
### 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:

```plaintext
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=5412;DB=mydb;schema=public))
```

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

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

---

### Wire Protocol Driver Usage Notes

#### Overview

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

**Windows**  
The options are located on the Advanced or Performance tabs in the ODBC Administrator window.

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

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

#### SQL Server and SQL Server Legacy

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

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

*Note:*

1. Significant performance improvements have been realized when using the SQL Server Legacy Wire Protocol driver, as compared to the SQL Server Wire Protocol driver.
2. The SQL Server Legacy Wire Protocol driver does not support transactions when it is used with FedSQL enabled because the driver allows only a single statement per connection while FedSQL requires multiple statements per connection when using transactions.

---

**Oracle Reference**

**Understanding the Table Services Driver for Oracle**

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

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

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

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

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

**Data Service Connection Options for Oracle**

**Overview**

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

**Connection Options**

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

```bash
driver=sql;conopts=(driver=oracle;
catalog=acat;uid=myuid;pwd=myPass9;
path=oraclev11.abc.123.com:1521/ORACLE)
```
The driver for Oracle supports the following connection options.

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT_PRESERVE</td>
<td>CT_PRESERVE = STRICT</td>
</tr>
<tr>
<td></td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>• STRICT The requested type must exist in the target database. No type promotion occurs. If the type does not exist, an error is returned.</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEFAULT_ATTR</th>
<th>DEFAULT_ATTR=(attr=value;...)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td>0 Causes the driver to use client-side static cursor emulation if a scrollable cursor is requested but the database server cannot provide one.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Example: DEFAULT_ATTR=(CURSORS=2)</td>
</tr>
<tr>
<td></td>
<td>• 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)</td>
</tr>
<tr>
<td></td>
<td>• 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)</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DRIVER_TRACE</td>
<td>DRIVER_TRACE='API</td>
</tr>
</tbody>
</table>
|                   | Requests tracing information, which logs transaction records to an external file that can be used for debugging purposes. The driver writes a record of each command that is sent to the database to the trace log based on the specified tracing level, which determines the type of tracing information. Here are the tracing levels:  
• **ALL** Activates all trace levels.  
• **API** Specifies that API method calls be sent to the trace log. This option is most useful if you are having a problem and need to send a trace log to SAS Technical Support for troubleshooting.  
• **DRIVER** Specifies that driver-specific information be sent to the trace log.  
• **SQL** Specifies that SQL statements that are sent to the database management system (DBMS) be sent to the trace log. Tracing information is DBMS specific, but most table services drivers log SQL statements such as SELECT and COMMIT.  
**Default:** Tracing is not activated.  
**Note:** If you activate tracing, you must also specify the location of the trace log with DRIVER_TRACEFILE=. Note that DRIVER_TRACEFILE= is resolved against the TRACEFILEPATH set in ALTER SERVER. TRACEFILEPATH is relative to the server's content root location.  
(Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=.  
**Interaction:** You can specify one trace level, or you can concatenate more than one by including the | (OR) symbol. For example, `driver_trace='api|sql'` generates tracing information for API calls and SQL statements. |
| DRIVER_TRACEFILE  | DRIVER_TRACEFILE='filename';                                                 |
|                   | Used to specify the name of the text file for the trace log. Include the filename and extension in single or double quotation marks (for example, `driver_tracefile='\mytrace.log'`).  
**Default:** The default TRACEFILE location applies to a relative filename, and it is placed relative to TRACEFILEPATH.  
**Requirement:** DRIVER_TRACEFILE is required when activating tracing using DRIVER_TRACE.  
**Interaction:** (Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=. |
| DRIVER_TRACEOPTIONS| DRIVER_TRACEOPTIONS=APPEND | THREADSTAMP | TIMESTAMP;  
|                   | Specifies options in order to control formatting and other properties for the trace log:  
• **APPEND** Adds trace information to the end of an existing trace log. The contents of the file are not overwritten.  
• **THREADSTAMP** Prepends each line of the trace log with a thread identification.  
• **TIMESTAMP** Prepends each line of the trace log with a time stamp.  
**Default:** The trace log is overwritten with no thread identification or time stamp. |
<p>| ORA_ENCODING      | ORA_ENCODING=UNICODE;                                                        |
|                   | Specifies that the Oracle data be returned in Unicode to table services. <strong>UNICODE</strong> is the default setting and is independent of the <strong>NLS_LANG</strong> environment variable setting. |</p>
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORNUMERIC</td>
<td>**ORNUMERIC=NO</td>
</tr>
<tr>
<td>USE_CACHED_CATALOG</td>
<td>**USE_CACHED_CATALOG=YES</td>
</tr>
</tbody>
</table>
Creating a Materialized View (USE_CACHED_CATALOG)
The following example shows you how to create a materialized view. Use this script if USE_CACHED_CATALOG is set to YES above.

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

Create materialized view SAS_CACHED_FK_CATALOG_MATVIEW REFRESH ON DEMAND as SELECT PKAC.OWNER as PKTABLE_SCHEM,
PKAC.TABLE_NAME as PKTABLE_NAME,
PKACC.COLUMN_NAME as PKCOLUMN_NAME,
FKAC.OWNER as FKTABLE_SCHEM,
FKAC.TABLE_NAME as FKTABLE_NAME,
FKACC.COLUMN_NAME as FKCOLUMN_NAME,
FKACC.POSITION as KEY_SEQ,
FKAC.CONSTRAINT_NAME as FK_NAME,
PKAC.CONSTRAINT_NAME as PK_NAME
from sys.all_constraints PKAC, sys.all_constraints FKAC,
sys.all_cons_columns PKACC, sys.all_cons_columns FKACC
where
PKAC.r_constraint_name=PKAC.constraint_name and
PKAC.constraint_name=PKACC.constraint_name and
FKAC.constraint_name=FKACC.constraint_name and FKAC.constraint_type='P' and
FKAC.constraint_type='R' and FKAC.owner=FKACC.owner and PKAC.owner=PKACC.owner
and PKAC.table_name=PKACC.table_name and FKAC.table_name=FKACC.table_name and
FKACC.position = PKACC.position ;

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

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

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

<table>
<thead>
<tr>
<th>Windows</th>
<th>The options are located on the Advanced or Performance tabs in the ODBC Administrator.</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

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

2. Verify that the Enable Bulk Load option is active in the ODBC DSN for databases that support this option. The Enable Bulk Load option is not enabled by default in the newer wire protocol drivers. As a result, insert performance suffers.

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

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

PostgreSQL Driver Reference

Understanding the SAS Federation Server Driver for PostgreSQL

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

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

The driver for PostgreSQL is a remote driver, which means that it connects to a server process in order to access data. The process might be running on the same machine as the table services, or it might be running on another machine in the network.
The table services driver for PostgreSQL uses shared libraries that are referenced as shared objects in UNIX. You must add the location of the shared libraries to one of the system environment variables, and set any other environment variables required by the PostgreSQL client libraries. The following Korn shell commands provide an example:

```bash
LD_LIBRARY_PATH=/dbi/odbc/unixodbc2310/lib:/dbi/postgres/9.03.04/lib:
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:

```plaintext
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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATALOG</td>
<td><code>CATALOG=catalog-identifier;</code> Specifies an arbitrary identifier for an SQL catalog, which groups schemas that are logically related (for example, <code>catalog=ptgtest</code>). Note: The FedSQL language processor automatically quotes SQL identifiers that do not meet the regular naming convention as defined in SAS Viya: FedSQL Programming for SAS Cloud Analytic Services.</td>
</tr>
<tr>
<td>CONOPTS</td>
<td><code>CONOPTS=(ODBC-compliant database connection string);</code> 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:</td>
</tr>
</tbody>
</table>

```plaintext
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))}
```
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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_ENCODING    | CLIENT_ENCODING=cei  
Used to specify encoding for the client. |
### CT_PRESERVE

**CT_PRESERVE=STRICT | SAFE | FORCE | FORCE_COL_SIZE**

Enables users to control how data types are mapped. Note that data type mapping is disabled when CT_PRESERVE is set to STRICT. If the requested type does not exist on the target database, an error is returned. Here are the options:

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

### DEFAULT_ATTR

**DEFAULT_ATTR=(attr=value;...)**

Used to specify connection handle or statement handle attributes supported for initial connect-time configuration, where `attr=value` corresponds to any of the following options:

- **CURSORS=n** - Connection handle option. This option controls the driver’s use of client-side, result set cursors. The possible values are 0, 1, or 2.
  - 0 Causes the driver to use client-side static cursor emulation if a scrollable cursor is requested but the database server cannot provide one.
  - 1 Causes the driver to always use client-side static cursor emulation if a scrollable cursor is requested. The database server’s native cursor is not used.
  - 2 (Default) Causes the driver to never use client-side static cursor emulation if a scrollable cursor is requested. The database server’s native cursor is used if available. Otherwise, the cursor is forward-only.

  Example: **DEFAULT_ATTR=(CURSORS=2)**

- **USE_EVP=n** - Statement handle option. This option optimizes the driver for large result sets. The possible values are 0 (OFF) or 1 (ON), which is the default. Example:  
  **DEFAULT_ATTR=(USE_EVP=0)**

- **XCODE_WARN=n** - Statement handle option. Used to warn about possible character transoding 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: DRIVER_TRACE

**DRIVER_TRACE='API | SQL | ALL';**

Requests tracing information, which logs transaction records to an external file that can be used for debugging purposes. The driver writes a record of each command that is sent to the database to the trace log based on the specified tracing level, which determines the type of tracing information. Here are the tracing levels:

- **ALL** Activates all trace levels.
- **API** Specifies that API method calls be sent to the trace log. This option is most useful if you are having a problem and need to send a trace log to SAS Technical Support for troubleshooting.
- **DRIVER** Specifies that driver-specific information be sent to the trace log.
- **SQL** Specifies that SQL statements that are sent to the database management system (DBMS) be sent to the trace log. Tracing information is DBMS specific, but most table services drivers log SQL statements such as SELECT and COMMIT.

**Default:** Tracing is not activated.

**Note:** If you activate tracing, you must also specify the location of the trace log with DRIVER_TRACEFILE=_. Note that DRIVER_TRACEFILE= is resolved against the TRACEFILEPATH set in ALTER SERVER. TRACEFILEPATH is relative to the server's content root location.

(Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=.

**Interaction:** You can specify one trace level, or you can concatenate more than one by including the | (OR) symbol. For example, `driver_trace='api|sql'` generates tracing information for API calls and SQL statements.

## Option: DRIVER_TRACEFILE

**DRIVER_TRACEFILE='filename';**

Used to specify the name of the text file for the trace log. Include the filename and extension in single or double quotation marks (for example, `driver_tracefile='\mytrace.log'`).

**Default:** The default TRACEFILE location applies to a relative filename, and it is placed relative to TRACEFILEPATH.

**Requirement:** DRIVER_TRACEFILE is required when activating tracing using DRIVER_TRACE.

**Interaction:** (Optional) You can control trace log formatting with DRIVER_TRACEOPTIONS=.

## Option: 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: MAX_BINARY_LEN

**MAX_BINARY_LEN=value;**

Specifies a value, in bytes, that limits the length of long binary fields (LONG VARCHAR). Unlike other databases, PostgreSQL does not have a size limit for long binary fields. The default is 1048576.
### Option Description

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_CHAR_LEN</td>
<td><code>MAX_CHAR_LEN=value;</code></td>
</tr>
<tr>
<td></td>
<td>Specifies a value that limits the length of character fields (CHAR and VARCHAR). The default is 2000.</td>
</tr>
<tr>
<td>MAX_TEXT_LEN</td>
<td><code>MAX_TEXT_LEN=value;</code></td>
</tr>
<tr>
<td></td>
<td>Specifies a value that limits the length of long character fields (LONG VARCHAR). The default is 409500.</td>
</tr>
<tr>
<td>SCHEMA</td>
<td><code>SCHEMA=value;</code></td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>STRIP_BLANKS</td>
<td>`STRIP_BLANKS=YES</td>
</tr>
<tr>
<td></td>
<td>Specifies whether to strip blanks from character fields.</td>
</tr>
</tbody>
</table>

### 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:
The following connection options are supported for SAS data sets:

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

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS</td>
<td>ACCESS=READONLY</td>
</tr>
<tr>
<td></td>
<td>• READONLY Assigns a read-only attribute to the schema. You cannot open a SAS data set to update or write new information.</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td>T I P</td>
<td>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.</td>
</tr>
<tr>
<td>CT_PRESERVE</td>
<td>CT_PRESERVE = STRICT</td>
</tr>
<tr>
<td>CT_PRESERVE</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>• STRICT The requested type must exist in the target database. No type promotion occurs. If the type does not exist, an error is returned.</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td>COMPRESS</td>
<td>COMPRESS=NO</td>
</tr>
<tr>
<td>COMPRESS</td>
<td>Controls the compression of rows in created SAS data sets.</td>
</tr>
<tr>
<td></td>
<td>• NO Specifies that the rows in a newly created SAS data set are uncompressed (fixed-length records). This setting is the default.</td>
</tr>
<tr>
<td></td>
<td>• YES</td>
</tr>
<tr>
<td></td>
<td>T I P Use this compression algorithm for character data.</td>
</tr>
<tr>
<td>COMPRESS</td>
<td>• 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.</td>
</tr>
<tr>
<td>T I P</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Option Description

#### DEFAULT_ATTR

**DEFAULT_ATTR=(attr=value;...)**

Used to specify connection handle or statement handle attributes that are supported for initial connect-time configuration, where *attr=value* corresponds to any of the following options:

- **CURSORS=n** - Connection handle option. This option controls the driver’s use of client-side, result set cursors. The possible values are 0, 1, or 2.
  - 0: Causes the driver to use client-side static cursor emulation if a scrollable cursor is requested but the database server cannot provide one.
  - 1: Causes the driver to always use client-side static cursor emulation if a scrollable cursor is requested. The database server’s native cursor is not used.
  - 2: (Default) Causes the driver to never use client-side static cursor emulation if a scrollable cursor is requested. The database server’s native cursor is used if available. Otherwise, the cursor is forward-only.

Example: `DEFAULT_ATTR=(CURSORS=2)`

- **USE_EVP=n** - Statement handle option. This option optimizes the driver for large result sets. The possible values are 0 (OFF) or 1 (ON), which is the default. Example: `DEFAULT_ATTR=(USE_EVP=0)`

- **XCODE_WARN=n** - Statement handle option. Used to warn about possible character transcoding errors that occur during row input or output operations. Possible values are 0 (returns an error), 1 (returns a warning), or 2 (ignore transaction errors). 0 is the default. Example: `DEFAULT_ATTR=(XCODE_WARN=1)`

#### ENCODING

**ENCODING=encoding-value;**

Overwrites and transcodes the encoding for input or output processing of SAS data sets.

*Note:* The default value is the current operating system setting.

#### LOCKTABLE

**LOCKTABLE=SHARED|EXCLUSIVE**

Places exclusive or shared locks on SAS data sets. You can lock tables only if you are the owner or have been granted the necessary privilege. The default value for the table services is SHARED.

- **SHARED** Locks tables in shared mode, allowing other users or processes to read data from the tables, but preventing other users from updating.
- **EXCLUSIVE** Locks tables exclusively, preventing other users from accessing any table that you open.

#### PATH_BIND

**PATH_BIND=CONNECT|ACCESS**

Specifies when and how schemas are validated during connection. CONNECT validates the entire connection string at the time of connection and returns an error if one or more schemas is invalid. ACCESS validates schemas when they are accessed so that processing continues regardless of errors in the schema portion of the connection string. ACCESS is the default for the table services.
**Teradata Reference**

**Understanding the Table Services Driver for Teradata**

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

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

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

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

```bash
LD_LIBRARY_PATH=/opt/teradata/client/14.10/lib64:/opt/teradata/client/14.10/tbuild/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:

```bash
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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CATALOG</strong></td>
<td>CATALOG=catalog-identifier;</td>
</tr>
<tr>
<td></td>
<td>Specifies an arbitrary identifier for an SQL catalog, which groups logically</td>
</tr>
<tr>
<td></td>
<td>related schemas. Any identifier is valid (for example, catalog=tera).</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> You must specify a catalog.</td>
</tr>
<tr>
<td><strong>DATABASE</strong></td>
<td>DATABASE=database-name;</td>
</tr>
<tr>
<td></td>
<td>Specifies the Teradata database. If you do not specify DATABASE=, you</td>
</tr>
<tr>
<td></td>
<td>connect to the default Teradata database, which is often named the same as</td>
</tr>
<tr>
<td></td>
<td>your user ID. If the database value that you specify contains spaces or</td>
</tr>
<tr>
<td></td>
<td>non-alphanumeric characters, you must enclose it in quotation marks.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DRIVER</td>
<td><strong>DRIVER=TERA;</strong></td>
</tr>
<tr>
<td></td>
<td>Identifies the data service to which you want to connect, which is a Teradata database.</td>
</tr>
<tr>
<td></td>
<td><em>Note:</em> You must specify the driver.</td>
</tr>
<tr>
<td>SERVER</td>
<td><strong>SERVER=server-name;</strong></td>
</tr>
<tr>
<td></td>
<td>Specifies the Teradata server identifier.</td>
</tr>
</tbody>
</table>

**Advanced Connection Options**

The following advanced options are supported for Teradata database.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCOUNT</td>
<td><strong>ACCOUNT=account-ID;</strong></td>
</tr>
<tr>
<td></td>
<td>Specifies an optional account number that you want to charge for the Teradata session.</td>
</tr>
<tr>
<td>CLIENT_ENCODING</td>
<td><strong>CLIENT_ENCODING=encoding-value</strong></td>
</tr>
<tr>
<td></td>
<td>Used to specify the character set for the session. UTF8 is the default if encoding is not specified. These character sets are supported:</td>
</tr>
<tr>
<td></td>
<td>ASCII</td>
</tr>
<tr>
<td></td>
<td>EBCDIC</td>
</tr>
<tr>
<td></td>
<td>EBCDIC037_0E</td>
</tr>
<tr>
<td></td>
<td>KATAKANAEBCDIC</td>
</tr>
<tr>
<td></td>
<td>KANJIBEBCDIC035</td>
</tr>
<tr>
<td></td>
<td>LATIN9_0A</td>
</tr>
<tr>
<td></td>
<td>THAI874_4A0</td>
</tr>
<tr>
<td></td>
<td>LATIN1250_1A0</td>
</tr>
<tr>
<td></td>
<td>CYRILLIC1251_2A0</td>
</tr>
<tr>
<td></td>
<td>LATIN1254_7A0</td>
</tr>
<tr>
<td></td>
<td>HEBREW1255_5A0</td>
</tr>
<tr>
<td></td>
<td>ARABIC1256_6A0</td>
</tr>
<tr>
<td></td>
<td>LATIN1258_8A0</td>
</tr>
<tr>
<td></td>
<td>TCHBIG5_1R0</td>
</tr>
<tr>
<td></td>
<td>SCHINES936_6R0</td>
</tr>
<tr>
<td></td>
<td>KANJI932_1S0</td>
</tr>
<tr>
<td></td>
<td>HANGUL949_7R0</td>
</tr>
<tr>
<td></td>
<td>TCHINES950_8R0</td>
</tr>
<tr>
<td></td>
<td>LATIN1252_3A0</td>
</tr>
<tr>
<td></td>
<td>SCHEBCDIC935_21J</td>
</tr>
<tr>
<td></td>
<td>TCHEBBCD937_31B</td>
</tr>
<tr>
<td></td>
<td>HANGULEBCDIC933_111</td>
</tr>
<tr>
<td></td>
<td>EBBCDIC273_0E</td>
</tr>
<tr>
<td></td>
<td>EBBCDIC277_0E</td>
</tr>
<tr>
<td></td>
<td>KANJIEBCCB5035_0I</td>
</tr>
<tr>
<td></td>
<td>KANJIEBCCBE9026_0I</td>
</tr>
<tr>
<td></td>
<td>UTF8</td>
</tr>
<tr>
<td></td>
<td>UTF16</td>
</tr>
</tbody>
</table>
### 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

DEFAULTATTR=(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: DEFAULTATTR=(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: DEFAULTATTR=(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: DEFAULTATTR=(XCODE_WARN=1)
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| DRIVER_TRACE        | **DRIVER_TRACE='API | SQL | ALL';** Requests tracing information, which logs transaction records to an external file that can be used for debugging purposes. The driver writes a record of each command that is sent to the trace log based on the specified tracing level, which determines the type of tracing information. Here are the tracing levels:  
  - **ALL** Activates all trace levels.  
  - **API** Specifies that API method calls be sent to the trace log. This option is most useful if you are having a problem and need to send a trace log to SAS Technical Support for troubleshooting.  
  - **DRIVER** Specifies that driver-specific information be sent to the trace log.  
  - **SQL** Specifies that SQL statements that are sent to the database management system (DBMS) be sent to the trace log. Tracing information is DBMS specific, but most table services drivers log SQL statements such as SELECT and COMMIT.  
  
  **Default:** Tracing is not activated.  
  
  **Note:** If you activate tracing, you must also specify the location of the trace log with DRIVER_TRACEFILE=. Note that DRIVER_TRACEFILE= is resolved against the TRACEFILEPATH set in ALTER SERVER. TRACEFILEPATH is relative to the server's content root location.  
  
  *(Optional)* You can control trace log formatting with DRIVER_TRACEOPTIONS=.  
  
  **Interaction:** You can specify one trace level, or you can concatenate more than one by including the | (OR) symbol. For example, `driver_trace='api|sql'` generates tracing information for API calls and SQL statements. |
| DRIVER_TRACEFILE    | **DRIVER_TRACEFILE='filename';** Used to specify the name of the text file for the trace log. Include the filename and extension in single or double quotation marks (for example, `driver_tracefile='\mytrace.log'`).  
  
  **Default:** The default TRACEFILE location applies to a relative filename, and it is placed relative to TRACEFILEPATH.  
  
  **Requirement:** DRIVER_TRACEFILE is required when activating tracing using DRIVER_TRACE.  
  
  **Interaction:** *(Optional)* You can control trace log formatting with DRIVER_TRACEOPTIONS=. |
| DRIVER_TRACEOPTIONS | **DRIVER_TRACEOPTIONS=APPEND | THREADSTAMP | TIMESTAMP;** Specifies options in order to control formatting and other properties for the trace log:  
  - **APPEND** Adds trace information to the end of an existing trace log. The contents of the file are not overwritten.  
  - **THREADSTAMP** Prepends each line of the trace log with a thread identification.  
  - **TIMESTAMP** Prepends each line of the trace log with a time stamp.  
  
  **Default:** The trace log is overwritten with no thread identification or time stamp. |
| PASSWORD            | **PASSWORD=password;** Specifies a Teradata password. The password must match your USER= value. The alias is PWD=.  
  
  **Note:** You must specify the PASSWORD= option. |
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| ROLE   | **ROLE=security-role;**  
|        | Specifies a security role for the session. |
| USER   | **USER=user-id;**  
|        | Specifies a Teradata user ID. If the ID contains blanks or national characters, enclose it in quotation marks. The alias is UID=.  
|        | Note: You must specify the USER= option. |
Recommended Reading

- *SAS Decision Manager: User’s Guide*
- *SAS Decision Manager: Administrator’s Guide*
- *SAS Event Stream Processing: Overview*
- *SAS Event Stream Processing: Using SAS Event Stream Processing Analytics*
- *SAS Event Stream Processing: Advanced Topics*
- *SAS Event Stream Processing: Visualizing Event Streams with Streamviewer*
- *SAS Event Stream Processing: Connectors and Adapters*
- *SAS Event Stream Processing: Publish/Subscribe API*
- *SAS Event Stream Processing: Programming Reference*
- *SAS Event Stream Processing: XML Language Dictionary*
- *SAS Event Stream Processing: Creating and Using Windows*
- *SAS Event Stream Processing: Using the ESP Server*
- *SAS Event Stream Processing: Using SAS Event Stream Processing Studio*
- *SAS Event Stream Processing: Tutorials and Examples*
- *SAS Viya: DS2 Programmer’s Guide*
- *SAS Viya: DS2 Language Reference*
- *SAS Viya Administration: Tuning*
- *SAS Viya Administration: Logging*
- *Encryption in SAS Viya: Data in Motion*
- *Encryption in SAS Viya: Data at Rest*
- *SAS Viya: FedSQL Programming for SAS Cloud Analytic Services*

For a complete list of SAS publications, go to [sas.com/store/books](http://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](http://sas.com/store/books)
Index

A
architecture, SAS Micro Analytic Service 9
ASTORE models 33

C
component hierarchy 9
contexts 8

D
deploying SAS Micro Analytic Service
SAS Event Stream Processing environment 52
drivers
Base SAS 129
DB2 98
FedSQL 103
Greenplum 106
ODBC 110, 117
Oracle 118
PostgreSQL 124
Teradata 133
wire protocol 117
DS2
best practices 37, 65
character-to-numeric conversions 41, 69
collection strings 57
database drivers 58
global packages 38, 66
hash package 41, 69
HTTP package 78
I/O 57
invariant computations 42, 70
local packages 38, 66
methods and packages, SAS Decision
Manager environment 59
methods and packages, SAS Event
Stream Processing environment 29
passing character values to methods 41, 69
programming, SAS Decision Manager
environment 55
programming, SAS Event Stream
Processing environment 27
return results 37, 65
single computation 42, 70
DS2 argument types
SAS Decision Manager environment 62
SAS Event Stream Processing
environment 32

G
global packages, DS2 38, 66

L
local packages, DS2 38, 66
logging SAS Micro Analytic Service
SAS Decision Manager environment 75
SAS Event Stream Processing
environment 51

M
MAStkstacksize environment variable
52
microanalyticservice.conf file 58
modules 8

P
publishing DS2 source code
SAS Decision Manager environment 55
SAS Event Stream Processing
environment 28
Python
and SAS Micro Analytic Service 43
configuration helper scripts 48
configuring 46
public and private methods 44
REST server error messages 93
return codes 81
revisions 9

SAS Decision Manager environment
DS2 argument types 62
DS2 methods and packages 59
overview 4
SAS Environment Manager 72
SAS Event Stream Processing
environment
ASTORE models 33
C++ methods 14
data type mappings 20
DS2 argument types 32
DS2 methods and packages 29
elements 14
event operation codes and flags 25
generating multiple derived events 22
overview 4

SAS Micro Analytic Service
architecture 9
authorization 77
component hierarchy 9
concepts 7
configuration 72
contexts 8
logging, SAS Decision Manager
environment 75
logging, SAS Event Stream Processing
environment 51
modules 8
REST server error messages 93
return codes 81
revision, for a module 9
starting and stopping 71
SCAN, replacing in DS2 code 39, 67

TRANWRD, replacing in DS2 code 39, 67
Gain Greater Insight into Your SAS® Software with SAS Books.

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

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