

# **SAS<sup>®</sup> Financial Management 5.3 Customization Guide**



The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2012. *SAS® Financial Management 5.3: Customization Guide*. Cary, NC: SAS Institute Inc.

**SAS® Financial Management 5.3: Customization Guide**

Copyright © 2012, SAS Institute Inc., Cary, NC, USA

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

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

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

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

**U.S. Government Restricted Rights Notice:** Use, duplication, or disclosure of this software and related documentation by the U.S. government is subject to the Agreement with SAS Institute and the restrictions set forth in FAR 52.227–19, Commercial Computer Software-Restricted Rights (June 1987).

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

1st electronic book, April 2012

SAS® Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at

[support.sas.com/publishing](http://support.sas.com/publishing) or call 1-800-727-3228.

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

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

---

# Contents

<b>Chapter 1 • About the Customization Guide</b>	<b>1</b>
What's in This Book	1
What's New in the Customization Guide	1
Required Skills	2
Documentation Conventions	3
Additional Documentation	3
<b>Chapter 2 • Content Administration</b>	<b>5</b>
Overview	5
The Reports Workspace	6
The SAS Information Delivery Portal	6
SAS Financial Management Stored Processes	8
<b>Chapter 3 • The SAS Financial Management Java API</b>	<b>11</b>
Using the SAS Financial Management Java API	12
Summary of Classes	16
The AdminQuery Class (Financial Planning and Reporting Only)	17
The AuditHistory Class	22
The BaseApi Class	24
The BaseQuery Class	26
The CellComments Class	28
The CycleQuery Class (Financial Cycles Only)	30
The Form Class (Financial Forms Only)	33
The FormSet Class	35
The Metadata Class	40
The Model Class (Financial Models Only)	41
Model Macros	44
Executing Queries with the %FMQUERY Macro	51
<b>Chapter 4 • Customizing a Workflow</b>	<b>63</b>
About Customizing a Workflow	63
Workflow Types	63
Adding Your Custom Code to a Workflow	64
Data Validation Example	68
<b>Chapter 5 • Creating a Custom Cell Action</b>	<b>73</b>
Overview	73
Write the Stored Process	74
Register the Stored Process	77
Update the Resource File	79
Select the Action	80
<b>Chapter 6 • Customizing the Form Status Dashboard</b>	<b>81</b>
The Form Status Dashboard	81
Indicators in the Form Status Dashboard	82
Customizing the Form Status Dashboard	83
<b>Chapter 7 • The SAS Financial Management Add-In API for Microsoft Excel</b>	<b>87</b>
Overview of Working with the SAS Financial Management Add-In API for Microsoft Excel	88
Setup for Using the API	88

General Usage Information . . . . .	89
Summary of Classes . . . . .	92
The FMAddIn Class . . . . .	93
The FMCollections Class . . . . .	97
The FMCrossing Class . . . . .	99
The FMCrossingsCollection Class . . . . .	100
The FMCube Class . . . . .	100
The FMCubesCollection Class . . . . .	105
The FMHierarchiesCollection Class . . . . .	105
The FMHierarchy Class . . . . .	105
The FMMember Class . . . . .	110
The FMMembersCollection Class . . . . .	113
The FMTable Class . . . . .	113
The FMTablesCollection Class . . . . .	121
The FMUser Class . . . . .	121
Including CDA Functions . . . . .	122
<b>Chapter 8 • Configuring Secure Socket Layer (SSL) . . . . .</b>	<b>123</b>
Introduction . . . . .	123
Overview of Configuring SSL for SAS Financial Management . . . . .	124
Configure the Web Application Server . . . . .	124
Modify the Remote Services . . . . .	125
Import the Certificate into Your Browser . . . . .	126
Edit the SAS Metadata . . . . .	126
Modify the SAS Environment Files . . . . .	128
Restart and Test . . . . .	129
Configuring Desktop Clients for Use with an SSL-Enabled Server . . . . .	129
<b>Index . . . . .</b>	<b>133</b>

## Chapter 1

# About the Customization Guide

---

<b>What's in This Book</b> .....	<b>1</b>
<b>What's New in the Customization Guide</b> .....	<b>1</b>
<b>Required Skills</b> .....	<b>2</b>
<b>Documentation Conventions</b> .....	<b>3</b>
<b>Additional Documentation</b> .....	<b>3</b>

---

## What's in This Book

This book includes the following topics:

- customizing SAS Financial Management content, including dashboard content, that is displayed in the SAS Information Delivery Portal
- using the SAS Financial Management APIs:
  - writing SAS code that uses the SAS Financial Management Java application programming interface (API)
  - writing Visual Basic Application (VBA) code in Microsoft Excel that interacts with SAS Financial Management objects
  - adding custom cell actions to Microsoft Excel
  - customizing a workflow
- configuring Secure Socket Layer (SSL)

*Note:* This book no longer contains information about alerts or directives, because those features are now part of the SAS Intelligence Platform or the Web Infrastructure Platform. For information about customizing themes, see the *SAS Intelligence Platform: Middle-Tier Administration Guide*.

---

## What's New in the Customization Guide

The customization guide includes the following new or updated information:

- SAS Financial Management Java API:

- The DATA step that includes a Javaobj declaration must include the following options in addition to the picklist option:

```
saveclassl='finbatch' getclassl='finbatch'
```

The saveclassl option loads and saves the classpath the first time it is called in a session. In subsequent DATA steps, the getclassl option reuses the already loaded classpath.

Do not call the logout method for a Java object.

- The %FMCELLC macro extracts cell comments from the SAS Financial Management data mart for a specific cycle. It writes those comments to a SAS data set.
- In the Model class, the updateTimeDefaultMembers method updates the default read and write members for the time hierarchy for a financial model.
- In the Model class, the constructor that specifies user ID and password is no longer supported. Call the METADATA\_PASSID function for authentication instead.
- The RUNASUSERID, TRUSTEDUSERNAME, TRUSTEDPASSWORD macro parameters are no longer supported.
- In the Form class, the constructor with a user ID and password as parameters is no longer supported. Use one of the other constructors instead.
- For custom cell actions, the output from the stored process should be written to the personal repository. Do not specify **New instance**.
- SAS Financial Management API for Microsoft Excel
  - Do not use **New** to instantiate a collection. If you need to create an empty FMMembersCollection object, use the GetNewMembersCollection method of the FMHierarchy class.
  - In the FMCube class's Crossing method: If a dimension or member in the array that represents the crossing is invalid, the method returns a null value. Your code should check for a null value before proceeding.
  - The FMHierarchy class has a new Boolean property, ShowAllMember, that applies to member property hierarchies (for example, Product.Color). It is used when you call a method such as SelectSlicerMember or ShowMemberSelectionDialog. If the property has a value of **True**, the dialog box includes the All member, which represents all members of the hierarchy.
- The customization guide contains instructions for customizing the SAS Financial Management form status dashboard.
- The customization guide contains information about managing SAS Financial Management content other than forms (for example, stored processes and alerts).

---

## Required Skills

To use the SAS Financial Management Java API, you must be familiar with both SAS and Java programming. To use the SAS Financial Management Add-In API for Microsoft Excel, you must have an understanding of Microsoft Excel and Visual Basic for Applications (VBA).

---

## Documentation Conventions

This book uses the following documentation conventions:

Convention	Description
<i>SAS-config-dir</i>	The path to the SAS configuration directory in the operating system. For example, <b>C:\SAS\Config</b> (Windows) or <b>/usr/local/SAS/Config</b> (UNIX).
<i>MySQL-install-dir</i>	The path to the MySQL installation directory. For example, <b>C:\MySQL\bin</b> (Windows) or <b>/usr/local/mysql</b> (UNIX)

*Note:*

- Your site might have a different configuration directory name or a different level number.
- File system pathnames are typically shown with Windows separators (\); for UNIX, substitute a forward slash.

---

## Additional Documentation

For additional information about SAS Financial Management, see the following books:

- *SAS Financial Management: Process Administrator's Guide*
- *SAS Financial Management: System Administrator's Guide*
- *SAS Financial Management: Migration Guide*
- *SAS Financial Management: Data Administrator's Guide*
- *SAS Financial Management: Data Model Reference*
- *SAS Financial Management: Formula Guide*
- *SAS Financial Management: Performance Guide*

They are available at <http://support.sas.com/documentation/onlinedoc/fm/index.html>.

*Note:* This site is password-restricted. You can find the user name and password in the preinstallation checklist or by contacting SAS Technical Support at <http://support.sas.com/techsup/contact>.

For information about the SAS Intelligence Platform, see <http://support.sas.com/93administration>.

For information about developing a stored process, see the *SAS Stored Processes: Developer's Guide*, which is available at <http://support.sas.com/documentation/onlinedoc/inttech/index.html>.

For information about administering third-party software, such as the Web application servers, see <http://support.sas.com/resources/thirdpartysupport/v93>.



## Chapter 2

# Content Administration

---

<b>Overview</b> .....	<b>5</b>
<b>The Reports Workspace</b> .....	<b>6</b>
<b>The SAS Information Delivery Portal</b> .....	<b>6</b>
Overview .....	6
Creating Page Templates for the Portal .....	7
Alerts Portlets .....	7
Link to SAS Financial Management .....	7
<b>SAS Financial Management Stored Processes</b> .....	<b>8</b>
Overview .....	8
Administrative Stored Processes .....	8
Standard Reports .....	8

---

## Overview

This chapter describes ways to manage the content (other than forms) that is displayed when a user logs on to SAS Financial Management or the SAS Information Delivery Portal in a Web browser:

- the Reports workspace of SAS Financial Management
- content that is available in the SAS Information Delivery Portal:
  - alerts
  - stored processes
  - SAS reports
  - information maps
  - a link to SAS Financial Management

For information about the Form Status dashboard, which is available in a SAS BI Dashboard portlet or in the SAS BI Dashboard viewer, see [Chapter 6, “Customizing the Form Status Dashboard,” on page 81](#).

---

## The Reports Workspace

When a user logs on to SAS Financial Management in a Web browser, the Reports workspace displays SAS Financial Management content. The following content types are supported in the Reports workspace:

- **Microsoft Excel spreadsheet:** a dynamic SAS Financial Management report. A dynamic report is a fully functional Excel binary file. It can be modified and its data can be refreshed.
- **Financial management report for Microsoft Excel:** a static SAS Financial Management report. A static report cannot be modified and its data cannot be refreshed.
- **PDF file:** a PDF document that can be opened in Adobe Acrobat.

Reports are opened in the appropriate application (Microsoft Excel or Adobe Acrobat).

For more information about the Reports workspace, see the *SAS Financial Management: User's Guide*, available from the Help menu of SAS Financial Management on the Web.

---

## The SAS Information Delivery Portal

### Overview

From the SAS Information Delivery Portal, users can access the following SAS Financial Management content:

- **Stored processes.** The standard SAS Financial Management reports are available in the `/Products/SAS Financial Management/5.3 Standard Reports` folder.

In the portal, stored processes can be added to a Collection portlet.

Alternatively, a single stored process can be added to a SAS Stored Process portlet. In that case, the stored process is executed when a user opens the portal page or refreshes the stored process.

For more information, see [“SAS Financial Management Stored Processes” on page 8](#).

- **SAS reports.** SAS reports are generated by the Publish wizard in the SAS Financial Management Add-In for Microsoft Excel. These static reports can be opened in SAS Web Report Studio. They display current data but cannot be modified.

In the portal, SAS reports can be added to a Collection portlet. Clicking a report opens it in SAS Web Report Studio.

- **Information maps.** These information maps are generated by the Information Map wizard in the SAS Financial Management Add-In for Microsoft Excel. They can be opened in SAS Web Report Studio as dynamic reports.

In the portal, information maps can be added to a Collection portlet. Clicking an information map opens it in SAS Web Report Studio.

*Note:* Information maps that are created in the SAS Financial Management Add-In for Microsoft Excel cannot be opened in SAS Information Map Studio.

- **Alerts.** Alerts from workflow events and forecasting can be viewed in an Alerts portlet. See “[Alerts Portlets](#)” on page 7.
- **Link to SAS Financial Management.** For more information, see “[Link to SAS Financial Management](#)” on page 7.
- the form status dashboard. For more information, see “[The Form Status Dashboard](#)” on page 81.

Users must have the necessary capabilities and security permissions to perform these tasks and view the data. For more information about capabilities, see “Assigning Groups and Roles” in the *SAS Financial Management: System Administrator’s Guide*,

## Creating Page Templates for the Portal

If you are a portal content administrator, you can design page templates for the portal that contain Collection portlets, Alerts portlets, and BI Dashboard portlets. You can share these templates with specific groups, and users in those groups can then add those pages to their own portal pages.

For information about becoming a portal content administrator and designing page templates, see the *SAS Intelligence Platform: Web Application Administration Guide*, available at <http://support.sas.com/93administration>.

The *SAS Financial Management: User’s Guide* also explains how users can add portlets to their own portal pages.

## Alerts Portlets

An alert is a notification of an event that the user might need to respond to.

Workflow alerts are notifications of tasks that the user has to perform, such as approving a budget form. On the Preferences window of the portal, users can specify how they want to receive alert notifications. One method is via an e-mail message. Another method is an Alerts portlet.

To add an Alerts portlet to a page, follow these steps:

1. From the **Customize** menu, select **Edit Page** ⇒ **Edit Page Content**.
2. On the Edit Page Content page, select **Add Portlets** and add a Shared Alerts portlet.

For more information about adding portlets to a page, see the online Help for the portal.

## Link to SAS Financial Management

To add a link to SAS Financial Management from the portal, follow these steps:

1. In a Collection portlet, click the **Edit Content** button.
2. In the Edit Portlet Content window, click **Add Items**.
3. In the Add Items to Portlet window, click the **Search** tab.
4. In the **Content Types** list, select **Application**.
5. Enter **Forms** as a keyword and click **Search**.

6. In the search results, select the SAS Financial Management application and add it to the portal page.

---

## SAS Financial Management Stored Processes

### Overview

The `/Products/SAS Financial Management/5.3 Standard Reports` folder contains a number of stored processes:

- two stored processes that are available for administrators (Import Users and Groups and ETL Job Status)
- several standard reports, such as Audit and Facts, that can be made available to end users
- a stored process (Form status) that is used only to generate output for the Form Process Status dashboard. It cannot be executed from the portal directly.

*Note:* If you open the stored process properties in SAS Management Console, you see that in many cases the **Type** is **Stored process (9.2)**, indicating that they are compatible with SAS 9.2. If you want to modify one of these stored processes and use SAS 9.3 stored process features, you must first upgrade the stored process. For more information, see “Making Stored Processes Compatible with 9.2 and Upgrading Stored Processes” in the *SAS Stored Processes: Developer's Guide*.

### Administrative Stored Processes

The `/Products/SAS Financial Management/5.3 Standard Reports` folder contains two stored processes that are typically made available only to administrators. To run these stored processes, administrators need ReadMetadata access to the stored process and membership in the SASSDM MySQL Users group.

- **Import Users and Groups**

This stored process loads information about user and group membership to the SAS Financial Management Data Mart. It is equivalent to running the `solnsvc_1300_load_users`, `solnsvc_1400_load_groups`, and `solnsvc_1500_load_user_x_group` jobs in SAS Data Integration Studio.

For more information, see “Assigning Groups and Roles” in the *SAS Financial Management: System Administrator's Guide*.

- **ETL Job Status**

This stored process displays the status of jobs that are executed in SAS Data Integration Studio.

### Standard Reports

The `/Products/SAS Financial Management/5.3 Standard Reports` folder also contains a set of standard SAS Financial Management reports that you might want to make available to other users. To execute these stored processes, users need ReadMetadata permission for the folder or for the individual stored processes.

*Note:* Without ReadMetadata permission on a folder, users cannot navigate to items beneath that folder. For more information, see “Best Practices for Managing SAS Folders” in the *SAS Intelligence Platform: System Administration Guide*, available at <http://support.sas.com/93administration>.

**Table 2.1** SAS Financial Management Standard Reports

Report	Description
<b>Audit</b>	<p>Lists actions that have been completed. You can limit an Audit report to an object type, a user, or a range of dates. By default this stored process returns a maximum of 10,000 rows. For information about modifying the stored process, see the description of the AuditHistory class in the <i>SAS Financial Management: Customization Guide</i>.</p> <p>This stored process applies to both financial and operational cycles. The other reports in this list apply only to financial cycles.</p>
<b>Data Entry</b>	Lists data records that were entered through financial forms using a specified financial model. You can limit a Data Entry report to a time period, an organization, an analysis member, or a financial form set.
<b>Eliminations</b>	Lists, for a specified financial model, data records for all accounts that have the Intercompany attribute but that are not specified in any intercompany balancing rule or net intercompany balancing rule. There should not be any such accounts, so this report should not list any data records. If the report does list data records, then you need to edit the rules that look for imbalances in intercompany accounts, or add more such rules. You can limit an Eliminations report to a time period, an organization, or an analysis member.
<b>ETL Facts</b>	Lists data records that have been loaded from SAS Data Integration Studio to a specified time period and analysis member within a specified financial cycle. You can further limit an ETL Facts report to a specified organization.
<b>Facts</b>	<p>Lists data records that are associated with a specified financial model.</p> <p>You can limit a Facts report to a time period or an analysis member, and in several other ways.</p>
<b>ICAccounts</b>	Lists, for a specified financial model, accounts that have the Intercompany attribute but that are not specified in any intercompany balancing rule or net intercompany balancing rule. You can limit an ICAccounts report to accounts that belong to a particular account type or accounts that have a particular balance type.
<b>Intercompany</b>	Lists, for a specified financial model, data records in which the account member has the Intercompany attribute and the trader member is either EXT or identical to the organization member. No records should satisfy this condition. You can limit an Intercompany report to a time period, an organization, or an analysis member.
<b>Manual Adjustments</b>	Lists all the currently posted manual adjustments for a specified financial model. You can limit a Manual Adjustments report to a time period, an organization, or an analysis member. You can also limit the report to a range of adjustment amounts.
<b>Non Intercompany</b>	Lists, for a specified financial model, data records in which the account member does not have the Intercompany attribute and the trader member is neither EXT nor identical to the organization member. No records should satisfy this condition. You can limit a Non Intercompany report to a time period, an organization, or an analysis member.

Report	Description
<b>Non Leaf</b>	Lists all the non-leaf data records (also known as virtual-child data records) for a specified financial model. You can limit a Non Leaf report to a time period, an organization, or an analysis member.
<b>Ownership Adjustments</b>	Lists all the adjustments that are generated by the ownership rule for a specified financial model. You can limit an Ownership Adjustments report to a time period, an analysis member, a holding organization, or a held organization.
<b>Ownership Methods</b>	Lists all the ownership relations that are specified in the ownership rule for a specified financial model, showing the consolidation method for each relation. You can limit an Ownership Methods report to a holding organization, a held organization, or a consolidation method.
<b>Ownership Transactions</b>	Lists all the asset purchases and asset sales that are specified in the ownership rule for a specified financial model. You can limit an Ownership Transactions report to a holding organization, a held organization, or a transaction type.
<b>Rule</b>	Lists all the adjustments that are generated by a specified adjustment rule within a specified financial model. You can limit a Rule report to a time period or an analysis member.
<b>Rules Facts</b>	Lists all the adjustments that are generated by all the adjustment rules that are part of a specified financial model. You can limit a Rules Facts report to a time period, an organization, or an analysis member. You can also limit the report to a range of adjustment amounts.
<b>Trial Balance</b>	Lists data records that are associated with a specified financial model and that were loaded from the SAS Financial Management staging area. You can limit a Trial Balance report to a time period, an organization, or an analysis member.

The Form status stored process, also in this folder, is used only to generate output for the Form Process Status dashboard. It cannot be directly executed, the way the other stored processes can. For more information, see [Chapter 6, “Customizing the Form Status Dashboard,”](#) on page 81.

## Chapter 3

# The SAS Financial Management Java API

---

<b>Using the SAS Financial Management Java API</b> . . . . .	<b>12</b>
About the SAS Financial Management Java API . . . . .	12
Declaring the Javaobj . . . . .	13
Authenticating the User . . . . .	13
Calling an Object's Methods . . . . .	14
Deleting the Javaobj . . . . .	14
Retrieving Error Messages . . . . .	15
Configuring a Log File . . . . .	15
Handling Exceptions . . . . .	15
A Simple Example . . . . .	15
<b>Summary of Classes</b> . . . . .	<b>16</b>
<b>The AdminQuery Class (Financial Planning and Reporting Only)</b> . . . . .	<b>17</b>
Overview . . . . .	17
Method Summary . . . . .	17
<b>The AuditHistory Class</b> . . . . .	<b>22</b>
Overview . . . . .	22
Method Summary . . . . .	23
<b>The BaseApi Class</b> . . . . .	<b>24</b>
Overview . . . . .	24
Method Summary . . . . .	24
<b>The BaseQuery Class</b> . . . . .	<b>26</b>
Overview . . . . .	26
Method Summary . . . . .	26
<b>The CellComments Class</b> . . . . .	<b>28</b>
Overview . . . . .	28
The %FMCELLC Macro . . . . .	28
Syntax . . . . .	29
Examples . . . . .	30
<b>The CycleQuery Class (Financial Cycles Only)</b> . . . . .	<b>30</b>
Overview . . . . .	30
Method Summary . . . . .	31
<b>The Form Class (Financial Forms Only)</b> . . . . .	<b>33</b>
Overview . . . . .	33
Method Summary . . . . .	34
<b>The FormSet Class</b> . . . . .	<b>35</b>
Overview . . . . .	35
Method Summary . . . . .	36

<b>The Metadata Class</b> .....	<b>40</b>
Overview .....	40
Method Summary .....	40
<b>The Model Class (Financial Models Only)</b> .....	<b>41</b>
Overview .....	41
Method Summary .....	41
<b>Model Macros</b> .....	<b>44</b>
Overview .....	44
The %GETALLMODELS Macro .....	45
The %GETFORMS Macro .....	46
The %GETFORMSETS Macro .....	47
The %GETMODELHIERARCHIES Macro .....	48
The %GETMODELMEMBERS Macro .....	49
The %GETMODELPROPERTIES Macro .....	50
<b>Executing Queries with the %FMQUERY Macro</b> .....	<b>51</b>
Overview .....	51
Query Types .....	52
Syntax .....	52
The Query Data Set .....	53
%FMQUERY Example (Non-MDX) .....	54
%FMQUERY Example with MDX String .....	54
Copying an MDX String .....	55
MDX Reference for SAS Financial Management .....	55

---

## Using the SAS Financial Management Java API

### *About the SAS Financial Management Java API*

The SAS Financial Management application programming interface (API) includes a set of Java classes and a set of SAS macros that are available for accessing SAS Financial Management data. Among other tasks, the API can be used for the following purposes:

- to execute a custom query against SAS Financial Management data
- to get a list of models or information about the properties, members, hierarchies, forms, or form sets that are associated with a specified model
- to reset or publish a form set
- to post adjustments for a model

Most of the classes apply only to financial planning and reporting. However, the macros (other than %FMQUERY) apply to both financial and operational models. The AuditHistory and Metadata classes can also be used for both financial and reporting and for operational planning.

The API can be used in a stored process or in code that is called by a SAS Data Integration Studio job. It can also be used in an interactive SAS session.

For information about creating and registering a stored process, see the *SAS Stored Processes: Developer's Guide*. For information about security for stored processes, see the *SAS Intelligence Platform: Security Administration Guide*.



## Declaring the Javaobj

The API uses the Javaobj interface, a mechanism that is similar to the Java Native Interface (JNI) for instantiating Java classes and accessing their methods and fields. The DATA step that includes a Javaobj declaration must include the following options:

```
/picklist='finance/finance.txt' saveclassl='finbatch' getclassl='finbatch';
```

The picklist option enables the Javaobj to access the necessary JAR files.

The saveclassl option loads and saves the classpath the first time it is called in a session. In subsequent DATA steps, the getclassl option reuses the already-loaded classpath.

*Note:* Do not call the logout method when you delete the object.

You declare a Javaobj object using the following syntax:

```
dcl javaobj object-name (classname, constructor-arguments);
```

Parameters are as follows:

### *object-name*

The handle to the Java object that is returned. You use this handle to access the object's methods.

### *classname*

A string that contains the fully qualified name of the Java class that you are instantiating, such as “com/sas/solutions/finance/api/Form”.

### *constructor-arguments*

Any arguments that are required by the constructor.

## Authenticating the User

### **Authentication Using the METADATA\_PASSID Function**

In order to access SAS Financial Management data, the user must be authenticated on the middle tier. The recommended approach is to call the object's setEnvironment method and then call the METADATA\_PASSID function in the DATA step. For example:

```
data _null_ /picklist='finance/finance.txt' saveclassl="finbatch"
    getclassl="finbatch";
dcl javaobj j("com/sas/solutions/finance/api/AuditHistory");
j.ExceptionDescribe(1);
j.callVoidMethod("setEnvironment", "default");
call METADATA_PASSID("j", "");
```

This function creates a one-time user-password combination and authenticates the user on the middle tier.

In a stored process, the METADATA\_PASSID function has access to the user ID and password. In an interactive SAS session, the user is asked for the user ID and password to be used for authentication on the middle tier. If the authentication fails, check the stored process log or the SAS log.

For an interactive SAS session, you also need the configuration and autoexec options that are specific to SAS Financial Management. One way to invoke those options is by starting an interactive SAS session from the *SAS-config-dir\Lev1\SASApp* folder.

**Authentication from a Workflow**

For a stored process that is called from a workflow, you must get the session context from the FM\_SP\_SECKEY variable and pass it to the constructor for a Java class. For more information about using a stored process in a workflow, see [“Data Validation Example” on page 68](#).

**Specifying the SAS Financial Management Environment**

The environment argument is the name of an environment that is defined in the EnvironmentFactory.xml file; for example, “default,” “dev,” or “prod.” It should be the same value that a user would specify when logging on to the middle tier from Microsoft Excel.

If it was not added at installation time, add the JREOPTIONS option to the sasV9\_usermods.cfg file located in each SAS Application Server context directory that you use. The env.factory.location argument should point to a network-accessible copy of the EnvironmentFactory.xml file. By default, this file is made available as follows:

```
-JREOPTIONS=(-Denv.factory.location=
    http://hostname:port/SASConfig/EnvironmentFactory.xml)
```

*Note:* Line break inserted for readability. *hostname* is the name of the host machine for the middle tier, and *port* is the port number of the managed server to which you deployed SAS Financial Management.

**Calling an Object's Methods**

With a handle to the Java object, you can call its methods. This code calls the Form object's getState method, which returns a String value:

```
j.callStringMethod("getState", state);
```

In this example, *j* represents the handle to the Form object. The call statement matches the method's return type (for example, CALLSTRINGMETHOD, CALLDoubleMETHOD, CALLINTMETHOD, CALLBooleanMETHOD, or CALLVOIDMETHOD).

The first parameter is always the method name, and the last parameter always contains the return value (if any). The remaining parameters are the parameters that the Java method requires. In the example above, the getState method has no parameters.

*Note:* The value returned by a boolean method is 1 (**true**) or 0 (**false**).

**Deleting the Javaobj**

To avoid memory leaks, all instantiations of a Javaobj should be terminated by a call to the DELETE method, as in this example:

```
dcl javaobj j("com/sas/solutions/finance/api/AuditHistory");
j.ExceptionDescribe(1);
...
j.delete();
```

*Note:* Do not call the object's logout method.

## Retrieving Error Messages

Many methods return a Boolean value indicating whether the action was successful. Because SAS does not have a true Boolean type, the return code is either 0 (failure) or 1 (success). When the return code is 0, the `getErrorMessage` method can be used to retrieve the pertinent error message, as in this example:

```
if rc le 0 then do;
    j.callStringMethod("getErrorMessage", msg);
end;
```

## Configuring a Log File

In addition to calling `getErrorMessage`, you can generate a more detailed log by creating a `log4j.properties` file. For more information, see “Configure a Log File for the SAS Financial Management Reports” in the *SAS Financial Management: System Administrator's Guide*.

## Handling Exceptions

The `EXCEPTIONCHECK` method can be used to determine whether an exception has been thrown. The `EXCEPTIONCLEAR` method clears any existing exceptions. Here is an example:

```
/* clear any existing stored exception */
j.ExceptionClear();
j.callvoidmethod("getModelHierarchies", "Default_Model", "FMSData",
    "TstHierarchies");
/* check to see if an exception has been thrown */
rc = j.ExceptionCheck(exception);
if (exception) then
    put 'Exception occurred, Please check the log for more information';
```

*Note:* The `EXCEPTIONCHECK` method cannot be used to detect exceptions that are thrown when constructing an object.

## A Simple Example

This stored process example sets the default read and write members for the time hierarchy for a model:

```
*ProcessBody;
%stpbegin;

data _null_/picklist='finance/finance.txt' saveclassl="finbatch"
    getclassl="finbatch";
    length msg $ 1000;
    length ex 8;

    /* Instantiate the Javaobj */
    dcl javaobj oModel;
    oModel = _new_ javaobj("com/sas/solutions/finance/api/Model");
    oModel.ExceptionDescribe(1);
```

```

oModel.callVoidMethod("setEnvironment", "default");

/* Obtain a security context from the middle tier */
call METADATA_PASSID("oModel", "");

/* Set the model code. */
oModel.callVoidMethod("setCode", "MyModel");

/* Update the time dimension default read and write members */
oModel.callVoidMethod("updateTimeDefaultMembers",
    "Q1 2012", "Jan 2012");

/* Check for errors */
rc = oModel.ExceptionCheck(ex);
if (ex) then do;
    oModel.callStringMethod("getErrorMessage", msg);
    put msg;
    oModel.delete();
    abort;
end;

oModel.delete();
run;

%stpend;

```

*Note:* The SAS Financial Management standard reports call the %RPTINIT macro, which extends and replaces the %STPBEGIN macro. The %RPTINIT macro has the following syntax:

```
%RPTINIT ( [STYLE=] [, DEVICE=] )
```

- **STYLE** specifies the value to use in the ODS STYLE option. It defaults to sasweb.
- **DEVICE** specifies the option for generating graphical output. It defaults to ACTXIMG for Windows and GIF for UNIX.

For more information about these options, see the *SAS Stored Processes: Developer's Guide*.

---

## Summary of Classes

**Table 3.1** Summary of Classes. Each class is part of the *com.sas.solutions.finance.api* package.

Class	Description
AdminQuery	Contains methods for running queries on the Base Facts data of SAS Financial Management. Applies only to financial planning and reporting.
AuditHistory	Contains methods for running queries on the AuditHistory data of SAS Financial Management.

Class	Description
BaseApi	Serves as the base class for the SAS Financial Management Java API. This class is extended by the BaseQuery, Form, Metadata, and Model classes.
BaseQuery	Contains methods for running queries. This class is extended by the AuditHistory, AdminQuery, and CycleQuery classes.
CellComments	Contains a method for extracting cell comments.
CycleQuery	Contains methods for extracting facts from a cycle. Applies only to financial cycles.
Form	Contains methods for running queries on the properties of a planning form from SAS Financial Management. Applies only to financial forms.
Metadata	Contains methods for retrieving metadata about SAS Financial Management.
Model	<p>Contains methods for retrieving information about SAS Financial Management models, for running queries against a model, generating formula facts, posting adjustments, and updating the model's default read and write members for the time dimension. Applies only to financial models.</p> <p>Many of the methods have corresponding macros, which should be used instead of calling the methods directly.</p>

## The AdminQuery Class (Financial Planning and Reporting Only)

### Overview

The `com.sas.solutions.finance.api.AdminQuery` class contains methods for running queries on the Base Facts data. It extends the `com.sas.solutions.finance.api.BaseQuery` class. The `AdminQuery` class applies only to financial planning and reporting.

For an example of using the `AdminQuery` class, see the Facts stored process in the `!sasroot\finance\sasstp\facts.sas` directory (Windows) or the `!sasroot/sasstp/finance/facts.sas` directory (UNIX). This stored process lists data records that are associated with a specified financial model. You can limit a Facts report to a time period or an analysis member, and in several other ways.

### Method Summary

**Table 3.2** AdminQuery Class Method Summary

Method	Description
<code>AdminQuery()</code>	<p>Constructor.</p> <p>Throws: <code>java.lang.Exception</code></p>

Method	Description
boolean <b>executeQuery()</b>	<p>Executes the query using the filters and any other parameters that have been previously specified.</p> <p>Returns: <b>true</b> if the action succeeded; otherwise, <b>false</b></p> <p>Throws: java.lang.Exception</p>
java.lang.String <b>getQueryColNames</b> (java.lang.String queryType)	<p>Gets the list of column names for a specific query and model. This method can be executed before running a query. However, you must set the model to be used in the query before calling getQueryColNames.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>queryType: type of query to execute. For a list of possible values, see the definition of the setQueryType method.</li> </ul> <p>Returns: column names, separated by commas</p>
java.lang.String <b>getQueryColNames</b> (java.lang.String queryType, java.lang.String modelCode)	<p>Gets the list of column names for a specific query and model. This method can be executed before running a query.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>queryType: the type of query to execute. For a list of possible values, see the definition of the setQueryType method.</li> <li>modelCode: the model code to be used in the query.</li> </ul> <p>Returns: column names, separated by commas</p> <p>Throws: java.lang.Exception</p>
java.lang.String <b>getQueryColNamesWithSeparator</b> (java.lang.String queryType, java.lang.String modelCode, java.lang.String separator)	<p>Gets the list of column names for a specific query and model. This method can be executed before running the query.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>queryType: the type of query to execute. For a list of possible values, see the definition of the setQueryType method.</li> <li>modelCode: the model code to be used in the query.</li> <li>separator: the text (such as a comma) to be used to separate column names in the list that is returned.</li> </ul> <p>Returns: column names, separated by the separator text</p> <p>Throws: java.lang.Exception</p>
java.lang.String <b>getQuerySASNames</b> (java.lang.String queryType)	<p>Gets a list of column names (in SAS valid name format) for a specific query and model. This method can be executed before running a query. However, you must first set the model to be used in the query.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>queryType: the type of query to execute. For a list of possible values, see the definition of the setQueryType method.</li> </ul> <p>Returns: column names, separated by commas</p>

Method	Description
java.lang.String <b>getQuerySASNames</b> (java.lang.String queryType, java.lang.String modelCode)	<p>Gets a list of column names (in SAS valid name format) for a specific query and model. This method can be executed before running the query. It returns column names, separated by commas.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• queryType: the type of query to execute. For a list of possible values, see the definition of the setQueryType method.</li> <li>• modelCode: the model code to be used in the query.</li> </ul> <p>Returns: column names, separated by commas</p> <p>Throws: java.lang.Exception</p>
java.lang.String <b>getQuerySASNamesWithSeparator</b> (java.lang.String queryType, java.lang.String modelCode, java.lang.String separator)	<p>Gets the list of column names (in SAS valid name format) for a specific query and model. This method can be executed before running the query.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• queryType: the type of query to execute. For a list of possible values, see the definition of the setQueryType method.</li> <li>• modelCode: the model code to be used in the query.</li> <li>• separator: the text (such as a comma) to be used to separate column names in the list that is returned.</li> </ul> <p>Returns: column names, separated by the separator text</p> <p>Throws: java.lang.Exception</p>
java.lang.String <b>getReportingCurrency ()</b>	<p>Gets the reporting currency member code.</p> <p>Returns: If the reporting currency has been set (via the setReportingCurrency method), then that member code is returned. Otherwise, the default reporting currency is returned.</p> <p>Throws: java.lang.Exception</p>
boolean <b>setDimFilter</b> (java.lang.String code, java.lang.String value)	<p>Sets a filter on a dimension; to filter on multiple values, call the method for each value.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• code: the dimension code</li> <li>• value: the member code to be used as the filter value</li> </ul> <p>Returns: <b>true</b> if the action succeeded; otherwise, <b>false</b></p> <p>Throws: java.lang.Exception</p>

Method	Description
boolean <b>setDimFilterID</b> (java.lang.String dimID, java.lang.String memID)	<p>Sets a filter on a dimension; to filter on multiple values, call the method for each value.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• dimID: the dimension ID</li> <li>• memID: the member reference ID to be used in the filter</li> </ul> <p>Returns: <b>true</b> if the action succeeded; otherwise, <b>false</b></p> <p>Throws: java.lang.Exception</p>
boolean <b>setDimTypeFilter</b> (java.lang.String code, java.lang.String value)	<p>Sets a filter on a dimension type; to filter on multiple values, call the method for each value.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• code: the dimension type code.</li> <li>• value: the member code to be used in the filter.</li> </ul> <p>Returns: <b>true</b> if the action succeeded; otherwise, <b>false</b></p> <p>Throws: java.lang.Exception</p>
boolean <b>setFactsParms</b> (java.lang.String otid, java.lang.String oid, java.lang.String ssid)	<p>Deprecated. Use setParms instead.</p>
boolean <b>setFormSetID</b> (java.lang.String name)	<p>Sets the form set ID to be used in a query.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• name: the form set name.</li> </ul> <p>Returns: <b>true</b> if the action succeeded; otherwise, <b>false</b></p> <p>Throws: java.lang.Exception</p>
boolean <b>setModel</b> (java.lang.String name)	<p>Sets the model to be used in a query.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• name: the model name.</li> </ul> <p>Returns: <b>true</b> if the action succeeded; otherwise, <b>false</b></p> <p>Throws: java.lang.Exception</p>
boolean <b>setModelCode</b> (java.lang.String code)	<p>Sets the model to be used in a query.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• code: the model code.</li> </ul> <p>Returns: <b>true</b> if the action succeeded; otherwise, <b>false</b></p> <p>Throws: java.lang.Exception</p>
boolean <b>setModelID</b> (java.lang.String ID)	<p>Sets the model to be used in a query.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• ID: the model ID.</li> </ul> <p>Returns: <b>true</b> if the action succeeded; otherwise, <b>false</b></p> <p>Throws: java.lang.Exception</p>



Method	Description
boolean <b>setParms</b> (java.lang.String otid, java.lang.String oid, java.lang.String convert)	<p>Sets the parameters for the query.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• otid: the object type ID, which must be a string containing one of these values: <b>adjustmentsequence</b>, <b>attachment</b>, <b>cashinfusiontransaction</b>, <b>compositeresult</b>, <b>cycle</b>, <b>dataload</b>, <b>differentialwritedown</b>, <b>disposaltransaction</b>, <b>dividendtransaction</b>, <b>equityassignment</b>, <b>form</b>, <b>formset</b>, <b>formtemplate</b>, <b>holding</b>, <b>holdingmethodaccounts</b>, <b>lineitem</b>, <b>manualadjustment</b>, <b>measureexport</b>, <b>othercpolineitem</b>, <b>othercpotransaction</b>, <b>ownershipchangetransaction</b>, <b>period</b>, <b>poconsolidationmethod</b>, <b>pocholdingfact</b>, <b>purchaseadjustment</b>, <b>purchasedifferential</b>, <b>purchasetransaction</b>, <b>result</b>, <b>rule</b>, <b>standaloneparent</b>, or <b>balsheet_reversal</b>.</li> <li>• oid: the object ID. Typically, this value is an empty string ("").</li> <li>• convert: the currency conversion flag. A value of <b>Y</b> specifies that currency values should be converted from their functional currencies to a presentation currency. A value of <b>N</b> specifies that conversion should not take place.</li> </ul> <p>Returns: <b>true</b> if the parameter values are valid; otherwise, <b>false</b></p>
boolean <b>setQueryType</b> (java.lang.String queryType)	<p>Sets the type of query to execute.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• queryType: the type of query to execute, which must be a string containing one of these values: <b>ELIMINATIONS</b>, <b>NONLEAF</b>, <b>DATAENTRY</b>, <b>TRIALBALANCE</b>, <b>INTERCOMPANY</b>, <b>NONINTERCOMPANY</b>, <b>RULESFACTS</b>, <b>RULE</b>, <b>MANUALADJUSTMENTS</b>, <b>FACTS</b>, <b>OWNERSHIP</b>, <b>ICACCOUNTS</b>, <b>FACTSR</b>, <b>DETAILS</b>, <b>OWNERSHIPTRANSACTIONS</b>, or <b>OWNERSHIPMETHODS</b></li> </ul> <p>Returns: <b>true</b> if the query type value is valid; otherwise, <b>false</b></p>
boolean <b>setReportingCurrency</b> (java.lang.String code)	<p>Sets the currency to be used for reporting values.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• code: a currency code, such as <b>EUR</b>.</li> </ul> <p>Returns: <b>true</b> if the action succeeded; otherwise, <b>false</b></p> <p>Throws: java.lang.Exception</p>

Method	Description
boolean <b>setRule</b> (java.lang.String name)	<p>Sets the rule by name (required only by the RULE query).</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>name: the name of a rule.</li> </ul> <p>Returns: <b>true</b> if the action succeeded; otherwise, <b>false</b></p> <p>Throws: java.lang.Exception</p>
boolean <b>setRuleID</b> (java.lang.String id)	<p>Sets the rule by ID (required only by the RULE query).</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>id: the ID of a rule.</li> </ul> <p>Returns: <b>true</b> if the action succeeded; otherwise, <b>false</b></p> <p>Throws: java.lang.Exception</p>
boolean <b>setVCubeID</b> (java.lang.String ID)	<p>Sets the model using the ID of a virtual cube (vcube).</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>ID: the ID of a virtual cube.</li> </ul> <p>Returns: <b>true</b> if the action succeeded; otherwise, <b>false</b></p> <p>Throws: java.lang.Exception</p>

The following methods are inherited from class com.sas.solutions.finance.api.BaseQuery: getColumnName, getColumnSASName, getColumnType, getMaxRowsMessage, getNumberOfColumns, getNumericValue, getQueryColNames, getQueryColNamesWithSeparator, getQueryRecordsNumber, getQuerySASNames, getQuerySASNamesWithSeparator, getRecord, getValue, and setMaxRows.

The following methods are inherited from class com.sas.solutions.finance.api.BaseApi: authenticate, buildExceptionMessageString, getErrorMessage, getMessage, setEnvironment, setLocale, and trim.

The following methods are inherited from class java.lang.Object: equals, getClass, hashCode, notify, notifyAll, toString, and wait.

## The AuditHistory Class

### Overview

The com.sas.solutions.finance.api.AuditHistory class contains methods for running queries on AuditHistory data from SAS Financial Management. It extends the com.sas.solutions.finance.api.BaseQuery class.

For an example of using the AuditHistory class, see the Audit stored process (**!sasroot\finance\sasstp\audit.sas**). This stored process extracts audit and history data that is filtered by three optional parameters: a user, an action type, and a date range.

*Note:* By default, the Audit stored process returns a maximum of 10,000 records. It can be modified to return more records by adding the `setMaxRows` method to the stored process (keeping memory considerations in mind).

## Method Summary

**Table 3.3** AuditHistory Class Method Summary

Method	Description
<code>AuditHistory()</code>	<p>Constructor.</p> <p>Throws: <code>java.lang.Exception</code></p>
boolean <code>executeQuery()</code>	<p>Executes the query using the filters and any other parameter previously specified.</p> <p>The query generates records with the following columns (all are character data): <code>USERNAME</code>, <code>ACTION_TYPE_ID</code>, <code>TIMESTAMP_TS</code>, <code>OBJECT_CLASS_ID</code>, <code>OBJECT_ID</code>, <code>SOLUTION_ID</code>, <code>TRANSACTION_ID</code>, <code>AUDIT_ID</code>, <code>PROPERTY_NM</code>, <code>OLD_VALUE</code>, and <code>NEW_VALUE</code>. You can call the <code>getValue</code> method to retrieve these values.</p> <p>Returns: <b>true</b> if the action succeeded; otherwise, <b>false</b></p> <p>Throws: <code>java.lang.Exception</code></p>
java.lang.String <code>getQueryColNames()</code>	<p>Gets the list of column names that were returned by the AuditHistory query. This method can be called only after running the query.</p> <p>Returns: column names, separated by commas</p>
java.lang.String <code>getQueryColNames</code> (java.lang.String separator)	<p>Gets the list of column names that were returned by the AuditHistory query. This method can be called only after running the query.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>separator: the text (such as a comma) to be used to separate column names in the list that is returned.</li> </ul> <p>Returns: column names, separated by the separator text</p>
void <code>setDateFormat</code> (java.lang.String format)	<p>Sets the desired format for passing the dates when calling <code>setDateRange</code>.</p> <p>Throws: <code>java.lang.Exception</code></p>
boolean <code>setDateRange</code> (java.lang.String from, java.lang.String to)	<p>Sets a date range. Dates are expected to be in the format <i>mm/dd/yyyy</i> unless they are otherwise specified by a call to <code>setDateFormat</code>.</p> <p>Throws: <code>java.lang.Exception</code></p>

Method	Description
boolean <b>setFilter</b> (java.lang.String name, java.lang.String value)	<p>Sets a filter on a column. To filter on multiple values, call the method for each value.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>name: the column name. For a list of valid column names, see the description of the executeQuery method.</li> <li>value: the value for the filter. For example, if you wanted to see audit records for the sasdmo user, you would use a call like this: <pre>j.callBooleanMethod("setFilter", "username", "sasdmo", rc);</pre> </li> </ul> <p>Throws: java.lang.Exception</p>

The following methods are inherited from class com.sas.solutions.finance.api.BaseQuery: getColumnName, getColumnSASName, getColumnType, getMaxRowsMessage, getNumberOfColumns, getNumericValue, getQueryColNames, getQueryColNamesWithSeparator, getQueryRecordsNumber, getQuerySASNames, getQuerySASNamesWithSeparator, getRecord, getValue, and setMaxRows.

The following methods are inherited from class com.sas.solutions.finance.api.BaseApi: authenticate, buildExceptionMessageString, getErrorMessage, getMessage, setEnvironment, setLocale, and trim.

The following methods are inherited from class java.lang.Object: equals, getClass, hashCode, notify, notifyAll, toString, and wait.

## The BaseApi Class

### Overview

The com.sas.solutions.finance.api.BaseApi class is extended by the BaseQuery, Form, Metadata, and Model classes.

*Note:* BaseApi methods should be called only by one of its subclasses.

### Method Summary

**Table 3.4** BaseApi Class Method Summary

Method	Description
<b>BaseApi ()</b>	<p>Constructor.</p> <p>Throws: java.lang.Exception</p>

Method	Description
boolean <b>authenticate</b> (java.lang.String entityKey)	<p>Authenticates the user on the middle tier. This method can be called only from a stored process that is part of a workflow.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>entityKey: the security key that contains the session context information for the current user. See <a href="#">“Authentication from a Workflow” on page 14.</a></li> </ul> <p>Returns: <b>true</b> if the authentication succeeded; otherwise, <b>false</b></p> <p>Throws: java.lang.Exception</p>
java.lang.String <b>getErrorMessage</b> ()	<p>Gets the localized error message from the last action. If the setLocale method was called, the specified locale is used. Otherwise, the system default locale is used.</p> <p>Returns: a localized message string</p>
java.lang.String <b>getMessage</b> (java.lang.String message)	<p>Gets the localized message that corresponds to a message code. If the setLocale method was called, the specified locale is used. Otherwise, the system default locale is used.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>message: the identifier for a localized message string.</li> </ul> <p>For a list of valid message codes, see the Resources <i>_language-code.properties</i> files in the sas.solutions.finance.api.nls.jar file.</p> <p>To locate the correct JAR file, open the</p> <pre>!sasroot\picklist\finance\finance.txt</pre> <p>file and find the following name:</p> <pre>sas.solutions.finance.api.</pre> <p>Make a note of the version that corresponds to this name. The JAR file is in the <i>SAS-install-dir</i></p> <pre>\SASVersionedJarRepository\version</pre> <p>directory.</p> <p>Returns: a localized message string</p> <p>Example:</p> <pre>j.callStringMethod("getMessage",     "Api.QueryReturnedNoFacts.txt", msg); call symput('msg', msg);</pre>
boolean <b>setLocale</b> (java.lang.String l)	<p>Sets the locale. (The default locale is the system default locale.)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>l: a locale that is specified as <i>language-code_country-code</i>, such as <b>en_US</b> or <b>es_SP</b>. The <i>language-code</i> is a valid ISO language code in the form of a lowercase, two-character string, and the <i>country-code</i> is a valid ISO country code in the form of an uppercase, two-character string.</li> </ul> <p>Returns: <b>true</b> if the action succeeded; otherwise, <b>false</b></p>

Method	Description
java.lang.String <b>trim</b> (java.lang.String s)	Returns the value passed in, with trailing blanks removed.

The following methods are inherited from class java.lang.Object: equals, getClass, hashCode, notify, notifyAll, toString, and wait.

## The BaseQuery Class

### Overview

The com.sas.solutions.finance.api.BaseQuery class is extended by the AdminQuery, AuditHistory, and CycleQuery classes. It contains methods for retrieving the results of a query.

*Note:* The methods of the BaseQuery class should be called only from one of its subclasses.

### Method Summary

**Table 3.5** BaseQuery Class Method Summary

Method	Description
<b>BaseQuery ()</b>	Constructor. Throws: java.lang.Exception
java.lang.String <b>getColumnNName</b> (double n)	Gets the name of the <i>n</i> th column.
java.lang.String <b>getColumnSASName</b> (double n)	Gets the SAS name of the <i>n</i> th column.
java.lang.String <b>getColumnType</b> (double n)	Gets the column type (numeric or character) of the <i>n</i> th column.
java.lang.String <b>getMaxRowsMessage ()</b>	Gets the maximum number of rows that a query can return. If the query returns fewer than this maximum number of rows, the getMaxRowsMessage method returns an empty string. Otherwise, it returns a localized message with this string: <b>Showing the first n rows</b> , where <i>n</i> is the maximum number of rows that were requested.
int <b>getNumberOfColumns ()</b>	Gets the number of columns returned by the query. This method can be executed only after a query has run.
double <b>getNumericValue</b> (double n, double m)	Gets the numeric value of the <i>n</i> th column of the <i>m</i> th record.

Method	Description
java.lang.String <b>getQueryColNames</b> ()	Gets the list of column names that were returned by a query. This method can be executed only after the query has been run.  Returns: column names, separated by commas
java.lang.String <b>getQueryColNamesWithSeparator</b> (java.lang.String separator)	Gets the list of column names that were returned by a query. This method can be executed only after the query has been run.  Parameters: <ul style="list-style-type: none"> <li>separator: the text (such as a comma) to be used to separate column names in the list that is returned.</li> </ul> Returns: column names, separated by the separator text
int <b>getQueryRecordsNumber</b> ()	Gets the number of records (facts) that were returned by the query.
java.lang.String <b>getQuerySASNames</b> ()	Gets a list of column names (in SAS valid name format) that were returned by a query. This method can be executed only after the query has been run.  Returns: column names, separated by commas
java.lang.String <b>getQuerySASNamesWithSeparator</b> (java.lang.String separator)	Gets the list of column names (in SAS valid name format) that were returned by a query. This method can be executed only after the query has been run.  Parameters: <ul style="list-style-type: none"> <li>separator: the text (such as a comma) to be used to separate column names in the list that is returned.</li> </ul> Returns: column names, separated by the separator text
java.lang.String <b>getRecord</b> (double n)	Gets the <i>n</i> th record.  Parameters: <ul style="list-style-type: none"> <li>n: the index of a record in the query results.</li> </ul> Returns: record values, separated by commas
java.lang.String <b>getRecord</b> (double n, java.lang.String separator)	Gets the <i>n</i> th record.  Parameters: <ul style="list-style-type: none"> <li>n: the index of a record in the query results.</li> <li>separator: the text to be used as a separator, such as a comma.</li> </ul> Returns: record values, separated by the separator text
java.lang.String <b>getValue</b> (double n, double m)	Gets the value of the <i>n</i> th column of the <i>m</i> th record.

Method	Description
boolean <b>setMaxRows</b> (java.lang.String s)	<p>Sets the maximum number of records (or facts) a query can return. The default is 50,000,000 rows.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>s: the maximum number of rows. A value of <b>0</b> specifies no limit.</li> </ul> <p>Returns: <b>true</b> if the action succeeded; otherwise, <b>false</b></p>

The following methods are inherited from class `com.sas.solutions.finance.api.BaseApi`: `authenticate`, `buildExceptionMessageString`, `getErrorMessage`, `getMessage`, `setEnvironment`, `setLocale`, and `trim`.

The following methods are inherited from class `java.lang.Object`: `equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, and `wait`.

## The CellComments Class

### Overview

The `com.sas.solutions.finance.api.CellComments` class has a method for extracting cell comments from the SAS Financial Management data mart. Instead of calling this method directly, call the `%FMCELLC` macro.

### The %FMCELLC Macro

The `%FMCELLC` macro extracts cell comments from the SAS Financial Management data mart for a specific cycle. The macro returns all comments, both public and private, that match the selection criteria such as cycle name. It writes those comments to a data set with the following variables:

Variable	Description
COMMENT_ID	The ID of the comment in the database.
USER_ID	The user ID of the user who made the comment.
DOCUMENT_TYPE	One of the following: <b>FINANCIAL_FORM_SET</b> , <b>OPERATIONAL_FORM_SET</b> , or <b>FINANCIAL_REPORT</b> .
DOCUMENT_NAME	For a form, this variable contains the form set name. For a report, this variable contains the report ID, in the form <b>FINANCIAL_REPORT_report-ID</b> .
CYCLE_TYPE	The type of cycle: <b>FINANCIAL</b> or <b>OPERATIONAL</b> .
CYCLE_NAME	The name of the cycle.



Variable	Description
PARENT_COMMENT_ID	If the comment is a reply to a previous comment, this variable contains the ID of the previous comment. Otherwise, the variable contains the ID of the current comment.
PRIVACY	One of the following: <b>PUBLIC</b> or <b>PRIVATE</b> . <i>Note:</i> Private comments are not filtered by user.
COMMENT_TEXT	The text of the comment.
CREATED_DATE_TIME	The date and time the comment was created.
VISIBILITY	One of the following: <b>FORM_SET</b> (visible within the form set or report in which the comment was made) or <b>CYCLE</b> (visible within the cycle).
MODEL_CODE	The code for the model that is associated with the comment.
<i>dimension_CODE</i>	Codes for dimensions that make up the crossing for the comment.

*Note:* The code uses a delimiter of “|” when creating the result rows. If any of the fields in the output (for example, the cycle name or the comment text) contain this character, an underscore is substituted in the output. For example, a cell comment of “a|b|c” would become “a\_b\_c”.

## Syntax

```
%FMCELLC (
  CYCLENAME=""
  [, CYCLETYPE="FINANCIAL"
  [, DOCUMENTNAME=""
  [, DOCUMENTTYPE="FINANCIAL_FORM_SET"
  [, STARTDATETIME=""
  [, ENDDATETIME=""
  [, LOCALSASLIBNAME="Work"
  [, RESULTDATASETNAME="CellCommentDataSet"
  [, ENVIRONMENT="default"
)
```

### CYCLENAME

The name of the cycle for the query. Required.

### CYCLETYPE

The type of cycle, either FINANCIAL (the default) or OPERATIONAL.

### DOCUMENTNAME

If you want to restrict the query to comments that were added to a specific form set, specify this parameter with the form set name.

If you specify this parameter, you must also specify the form set type (unless you use the default).

**DOCUMENTTYPE**

The type of form set, either “FINANCIAL\_FORM\_SET” (the default) or “OPERATIONAL\_FORM\_SET”.

If you omit the DOCUMENTNAME parameter, the DOCUMENTTYPE parameter is ignored.

**STARTDATETIME, ENDDATETIME**

Starting and ending dates and times for the query, in the form *yyyy-mm-dd hh:mm:ss*.

**LOCALSASLIBNAME**

The library in which to create the output data set. The default is the WORK library.

**RESULTDATASETNAME**

The name of the result data set. The default name is “CellCommentDataSet”.

**ENVIRONMENT**

An environment (such as “default”, “dev”, or “prod”) refers to an installation of SAS Financial Management. If you omit this parameter, “default” is used.

The environment value is site-specific. For more information, see [“Specifying the SAS Financial Management Environment” on page 14](#).

*Note:* These parameters are not case sensitive.

## Examples

Get all cell comments for the specified financial cycle. Comments are returned in WORK.CellCommentDataSet:

```
%FMCELLC(CYCLENAM="MyCycle");
```

Get all cell comments for the specified financial cycle that were made since September 1, 2011. Display only comments that were added to reports:

```
%FMCELLC(CYCLENAM="MyCycle", STARTDATETIME="2011-09-01 00:00:00");
PROC PRINT DATA=CellCommentDataSet NOOBS;
    WHERE DOCUMENT_TYPE="FINANCIAL_REPORT";
run;
```

Get all cell comments that were added to the specified form set. Display only public comments:

```
%FMCELLC(CYCLENAM="MyCycle", DOCUMENTNAME="MyFormset");
proc print data=CellCommentDataSet NOOBS;
    where PRIVACY="PUBLIC";
run;
```

For more information about cell comments, see the *SAS Financial Management Process Administrator’s Guide*.

---

## The CycleQuery Class (Financial Cycles Only)

### Overview

The com.sas.solutions.finance.api.CycleQuery class contains methods for extracting facts from a cycle. It extends the com.sas.solutions.finance.api.BaseQuery class.

This class applies only to financial cycles.

The CycleQuery class is similar to the AdminQuery class. For an example of its use, see the ETL Facts stored process (`!sasroot\finance\sasstp\etlfacts.sas`). This stored process lists data records that have been loaded from SAS Data Integration Studio to a specified time period and analysis member within a specified financial cycle, and perhaps a specified organization.

## Method Summary

**Table 3.6** CycleQuery Class Method Summary

Method	Description
<code>CycleQuery()</code>	Constructor. Throws: <code>java.lang.Exception</code>
boolean <code>getETLFacts()</code>	Gets the ETL facts for the specified cycle and filters. Returns: <b>true</b> if the action succeeded; otherwise, <b>false</b> Throws: <code>java.lang.Exception</code>
java.lang.String <code>getQueryColNames</code> (java.lang.String cycleName, java.lang.String separator)	Gets the list of column names for the query. This method can be called before running the query. Parameters: <ul style="list-style-type: none"> <li>• <code>cycleName</code>: the name of a cycle</li> <li>• <code>separator</code>: the text, such as a comma, to be used as a separator</li> </ul> Returns: a list of column names, separated by the separator text. Throws: <code>java.lang.Exception</code>
java.lang.String <code>getQueryColNamesByID</code> (java.lang.String cycleID, java.lang.String separator)	Gets the list of column names for the query. This method can be called before running the query. Parameters: <ul style="list-style-type: none"> <li>• <code>cycleID</code>: the ID of a cycle.</li> <li>• <code>separator</code>: the text, such as a comma, to be used as a separator.</li> </ul> Returns: a list of column names, separated by the separator text Throws: <code>java.lang.Exception</code>
java.lang.String <code>getQuerySASNames</code> (java.lang.String cycleName, java.lang.String separator)	Gets a list of column names (in SAS valid name format) for a specific query and cycle. This method can be executed before running the query. Parameters: <ul style="list-style-type: none"> <li>• <code>cycleName</code>: the cycle name.</li> <li>• <code>separator</code>: the text, such as a comma, to be used as a separator.</li> </ul> Returns: a list of column names, separated by the separator text Throws: <code>java.lang.Exception</code>

Method	Description
java.lang.String <b>getQuerySASNamesByID</b> (java.lang.String cycleID, java.lang.String separator)	<p>Gets a list of column names (in SAS valid name format) for a specific query and cycle. This method can be executed before running the query (after setting the cycle to be used in the query).</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• cycleID: the cycle ID.</li> <li>• separator: the text, such as a comma, to be used as a separator.</li> </ul> <p>Returns: a list of column names, separated by the separator text</p> <p>Throws: java.lang.Exception</p>
boolean <b>setCycleByID</b> (java.lang.String ID)	<p>Sets the cycle for the query by ID.</p> <p>Returns: <b>true</b> if the action succeeded; otherwise, <b>false</b></p> <p>Throws: java.lang.Exception</p>
boolean <b>setCycleByName</b> (java.lang.String name)	<p>Sets the cycle for the query by name.</p> <p>Returns: <b>true</b> if the action succeeded; otherwise, <b>false</b></p> <p>Throws: java.lang.Exception</p>
boolean <b>setDimTypeFilter</b> (java.lang.String code, java.lang.String value)	<p>Sets a filter on a dimension type; to filter on multiple values, call the method for each value.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• code: the dimension type code.</li> <li>• value: the member code to be used in the filter.</li> </ul> <p>Returns: <b>true</b> if the action succeeded; otherwise, <b>false</b></p> <p>Throws: java.lang.Exception</p>

Method	Description
boolean <b>setParms</b> (java.lang.String otid, java.lang.String oid)	<p>Sets the parameters for the query.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• otid: the object type ID. Possible values are <b>adjustmentsequence</b>, <b>attachment</b>, <b>cashinfusiontransaction</b>, <b>compositeresult</b>, <b>cycle</b>, <b>dataload</b>, <b>differentialwritedown</b>, <b>disposaltransaction</b>, <b>dividendtransaction</b>, <b>equityassignment</b>, <b>form</b>, <b>formset</b>, <b>formtemplate</b>, <b>holding</b>, <b>holdingmethodaccounts</b>, <b>lineitem</b>, <b>manualadjustment</b>, <b>measureexport</b>, <b>othercpolineitem</b>, <b>othercpotransaction</b>, <b>ownershipchangetransaction</b>, <b>period</b>, <b>pocconsolidationmethod</b>, <b>pocholdingfact</b>, <b>purchaseadjustment</b>, <b>purchasedifferential</b>, <b>purchasetransaction</b>, <b>result</b>, <b>rule</b>, <b>standaloneparent</b>, or <b>balsheet_reversal</b>.</li> <li>• oid: the object ID.</li> </ul> <p>Returns: <b>true</b> if the parameter values are valid; otherwise, <b>false</b></p> <p>Throws: java.lang.Exception</p>

The following methods are inherited from class com.sas.solutions.finance.api.BaseQuery: getColumnName, getColumnSASName, getColumnType, getMaxRowsMessage, getNumberOfColumns, getNumericValue, getQueryColNames, getQueryColNamesWithSeparator, getQueryRecordsNumber, getQuerySASNames, getQuerySASNamesWithSeparator, getRecord, getValue, and setMaxRows.

The following methods are inherited from class com.sas.solutions.finance.api.BaseApi: authenticate, buildExceptionMessageString, getErrorMessage, getMessage, setEnvironment, setLocale, and trim.

The following methods are inherited from class java.lang.Object: equals, getClass, hashCode, notify, notifyAll, toString, and wait.

## The Form Class (Financial Forms Only)

### Overview

The com.sas.solutions.finance.api.Form class contains methods for running queries on the properties of a planning form from SAS Financial Management. It extends the com.sas.solutions.finance.api.BaseApi class. This class applies only to financial forms.

For an example of using the Form class, see [“Data Validation Example” on page 68](#).

## Method Summary

**Table 3.7** Form Class Method Summary

Method	Description
<b>Form</b> (java.lang.String sFormId, java.lang.String entityKey)	<p>Constructor.</p> <p>This constructor can be used only in a stored process that is used in a workflow. Both the form ID and the security key (entityKey) are available as environment variables that are set by the workflow.</p> <p>Throws: java.lang.Exception</p>
java.lang.String <b>getAuthors</b> (java.lang.String delimiter)	<p>Returns the user IDs of all authors of a specified form, separated by the <i>delimiter</i> text if more than one author was found.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>delimiter: the text (such as a space or semi-colon) that is used to separate author names in the return string.</li> </ul> <p>Throws: java.lang.Exception</p>
java.lang.String <b>getDescription</b> ()	Returns the form description.
java.lang.String <b>getDueDate</b> ()	Returns the due date of the form.
java.lang.String <b>getFormSetDescription</b> ()	Returns the description of the form set to which the form belongs.
int <b>getFormSetId</b> ()	Returns the ID of the form set to which the form belongs.
java.lang.String <b>getFormSetName</b> ()	Returns the name of the form set to which the form belongs.
java.lang.String <b>getId</b> ()	Returns the form ID as a string.
java.lang.String <b>getInfo</b> ()	Returns a formatted string with key information about the form.
java.lang.String <b>getName</b> ()	Returns the form name.
java.lang.String <b>getPlanningAdministrators</b> (java.lang.String delimiter)	<p>Returns a list of users with the role of Finance Process Administrator.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>delimiter: the text that is used to separate names in the return string.</li> </ul> <p>Throws: java.lang.Exception</p>
java.lang.String <b>getReviewers</b> (java.lang.String delimiter)	<p>Returns all reviewers of a specified form. The reviewers are separated by the <i>delimiter</i> text if more than one reviewer was found.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>delimiter: the text that is used to separate names in the return string.</li> </ul> <p>Throws: java.lang.Exception</p>

Method	Description
java.lang.String <b>getState()</b>	Returns the form state.
java.lang.String <b>getTargetDimensionCode()</b>	Returns the code of the target dimension of the form set to which the form belongs.
java.lang.String <b>getTargetDimensionDescription()</b>	Returns the description of the target dimension of the form set to which the form belongs.
java.lang.String <b>getTargetDimensionName()</b>	Returns the name of the target dimension of the form set to which the form belongs.
java.lang.String <b>getTargetMemberCode()</b>	Returns the target member code of the form.
java.lang.String <b>getTargetMemberDescription()</b>	Returns the description of the target member of the form.
int <b>getTargetMemberId()</b>	Returns the target member ID of the form.
java.lang.String <b>getTargetMemberName()</b>	Returns the name of the target member of the form.
boolean <b>isLocked()</b>	Returns <b>true</b> if the form is locked by some process.

*Note:* The constructor that took a user ID and password as parameters is no longer supported. Use one of the other constructors instead.

The following methods are inherited from class `com.sas.solutions.finance.api.BaseApi`: `authenticate`, `buildExceptionMessageString`, `getErrorMessage`, `getMessage`, `setEnvironment`, `setLocale`, and `trim`.

The following methods are inherited from class `java.lang.Object`: `equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, and `wait`.

## The FormSet Class

### Overview

Using methods of the `com.sas.solutions.finance.api.FormSet` class, you can manage form sets from SAS code. For example, you can reset a form set or you can write a batch script to publish a form set.

This class applies to both financial and operational planning, with the restrictions noted for specific methods.

This example resets two form sets and extends their deadlines:

#### **Example Code 3.1** Using Methods of the FormSet Class

```
dcl javaobj formset('com/sas/solutions/finance/api/FormSet');
formset.ExceptionDescribe(1);
call METADATA_PASSID('formset','');
formset.callVoidMethod('setCycle','FM','MyCycleName');
```

```

formset.callVoidMethod('setFormSet','MyFormSetName');
formset.callVoidMethod('enableUserNotification');
formset.callVoidMethod('enableCommentRetirement');
formset.callVoidMethod('setComment','My reset comment');
formset.callVoidMethod('reset');
formset.callVoidMethod('setDeadline','Apr 10, 2010 9:00 AM');
formset.callVoidMethod('setComment','My publish comment');
formset.callVoidMethod('publish');
formset.callVoidMethod('setFormSet','MyOtherFormSetName');
formset.callVoidMethod('reset');
formset.callVoidMethod('moveDeadlineByCalendarMonth',1);
formset.callVoidMethod('moveDeadlineByDaysAndHours',0,1);
formset.callVoidMethod('publish');
formset.delete();

```

*Note:* When a date string is specified as a parameter, use a format such as the following: “Apr 10, 2010 3:00 PM” or “10 avr 2010 15:00:00”. Returned date strings have the same format. The system default locale is used.

## Method Summary

The following methods can be called for the FormSet class.

**Table 3.8** FormSet Class Method Summary

Method	Description
<b>FormSet ()</b>	Constructor. Throws: java.lang.Exception
void <b>disableCommentRetirement ()</b>	Disables comment retirement for the publish action. With comment retirement disabled, when you publish the form set, comments that were associated with the form set are retained.
void <b>disableDeletionOfDataEntryFacts ()</b>	Disables the deletion of data entry facts for the publish action. This method is valid only when the cycle type is “OP”.
void <b>disableUserNotification ()</b>	Disables user notification for the publish and reset actions.
void <b>enableCommentRetirement ()</b>	Enables comment retirement for the publish action. This is the default behavior. When you publish the form set, comments that were associated with the form set are retired.
void <b>enableDeletionOfDataEntryFacts ()</b>	Enables deletion of data entry facts for the publish action. This method is valid only when the cycle type is “OP”. This is the default behavior. When you publish the form set, data entry facts that were previously associated with the form set are deleted.
void <b>enableUserNotification ()</b>	Enables user notification for the publish and reset actions.
java.lang.String <b>getComment ()</b>	Returns the comment to be used for publish and reset actions.



Method	Description
java.lang.String <b>getCycle ()</b>	Returns the cycle name.
java.lang.String <b>getCycleType ()</b>	Returns the cycle type: "FM" (for financial cycles) or "OP" (for operational cycles).
java.lang.String <b>getDeadline ()</b>	Returns the due date for the form set. Throws: FinanceClientException
java.lang.String <b>getFormSet ()</b>	Returns the form set name.
java.lang.String <b>getTargetHierarchyAsOfDate ()</b>	Returns the as-of date for the target hierarchy. Throws: FinanceClientException
boolean <b>isCommentRetirementEnabled ()</b>	Returns the current comment retirement behavior for publication actions: <b>true</b> if comments are retired at publish time; otherwise <b>false</b> .
boolean <b>isDeletionOfDataEntryFactsEnabled ()</b>	Returns the current deletion behavior of data entry facts: <b>true</b> if data entry facts are deleted when a form set is published; otherwise <b>false</b> .  This method is valid only when the cycle type is "OP".
boolean <b>isLocked ()</b>	Returns the lock status of the form set: <b>true</b> if the form set is locked; otherwise <b>false</b> . Throws: FinanceClientException
boolean <b>isUserNotificationEnabled ()</b>	Returns the current user notification behavior for publish and reset: <b>true</b> if user notification is enabled; otherwise <b>false</b> .
void <b>lock ()</b>	Locks the form set. Throws: FinanceClientException
void <b>moveDeadlineByCalendarMonth</b> (double months)	Moves the current deadline by the specified number of calendar months. Parameters: <ul style="list-style-type: none"> <li>months: the number of months (positive or negative) by which to move the form set deadline.</li> </ul> Throws: FinanceClientException
void <b>moveDeadlineByDaysAndHours</b> (double days, double hours)	Moves the current deadline by the specified number of days and hours. Parameters: <ul style="list-style-type: none"> <li>days: the number of days (positive or negative) by which to move the form set deadline.</li> <li>hours: the number of hours (positive or negative) by which to move the form set deadline.</li> </ul> Throws: FinanceClientException

Method	Description
<code>void <b>publish</b> ()</code>	<p>Publishes the form set. The form set must be in Draft status, it cannot be locked, and it must have a valid template.</p> <p>Throws: <code>FinanceClientException</code></p>
<code>void <b>publish</b> (double max_seconds_to_wait)</code>	<p>Initiates the publish action and waits for its completion. The form set must be in Draft status, it cannot be locked, and it must have a valid template.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><code>max_seconds_to_wait</code>: the maximum time (in seconds) to wait before issuing a time-out exception.</li> </ul> <p>The <code>publish()</code> method (without a parameter) initiates the publish action and returns.</p> <p>The <code>publish(max_seconds_to_wait)</code> method initiates the publish operation and then checks the status of the publish activity for at most <code>max_seconds_to_wait</code> seconds. When the publish completes, the method returns. If the publish activity has not completed within the specified time, the method throws a time-out exception. However, it does not cancel the publish action.</p> <p>Throws: <code>FinanceClientException</code></p>
<code>void <b>reset</b> ()</code>	<p>Resets the form set. This operation is not possible if the form set is locked.</p> <p>If comment text has been defined, the comment is applied to the operation.</p> <p>Throws: <code>FinanceClientException</code></p>
<code>void <b>restoreDefaults</b> ()</code>	<p>Restores the following default attributes of the <code>FormSet</code> object:</p> <ul style="list-style-type: none"> <li>User notification is enabled.</li> <li>Comments are deleted when the form set is published.</li> <li>(Operational form sets) Data entry facts are not deleted when the form set is published.</li> <li>The comment text is set to an empty string.</li> <li>The locale is set to the system default locale.</li> </ul> <p>Calling this method does not affect the form set name, cycle name, and cycle type that are associated with the <code>FormSet</code> object.</p>
<code>void <b>setComment</b> (java.lang.String comment)</code>	<p>Defines the comment text to be used for publish and reset actions. By default, this text is an empty string.</p>
<code>void <b>setCurrentUntilPublished</b> ()</code>	<p>Sets the due date of the target hierarchy to "current until published". This method is valid only when the cycle type is "FM".</p> <p>Throws: <code>FinanceClientException</code></p>

Method	Description
void <b>setCycle</b> (java.lang.String type, java.lang.String name)	<p>Associates a cycle name and type with the FormSet object. You must call <code>setCycle</code> before calling any of the following methods: <code>publish</code>, <code>reset</code>, <code>setDeadline</code>, <code>getDeadline</code>, <code>moveDeadlineByCalendarMonth</code>, <code>moveDeadlineByDaysAndHours</code>, <code>setTargetHierarchyAsOfDate</code>, <code>getTargetHierarchyAsOfDate</code>, <code>setCurrentUntilPublished</code>, <code>lock</code>, <code>unlock</code>, and <code>isLocked</code>.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>type: “OP” for operational cycles; “FM” for financial cycles.</li> <li>name: the name of the cycle for this form set.</li> </ul>
void <b>setDeadline</b> (java.lang.String dateString)	<p>Sets the due date and time for the form set.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>dateString: a string that contains the date and time of the deadline.</li> </ul> <p>Throws: <code>FinanceClientException</code></p>
void <b>setFormSet</b> (java.lang.String name)	<p>Associates a form set with the FormSet object.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>name: the name of the form set.</li> </ul>
void <b>setTargetHierarchyAsOfDate</b> (java.lang.String dateString)	<p>Sets the as-of date for the target hierarchy. This method is valid only when the cycle type is “FM” and the form set is in Draft state.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>dateString: the date and time to use for the target hierarchy's as-of date.</li> </ul> <p>Throws: <code>FinanceClientException</code></p>
void <b>unlock</b> (double unlockForms)	<p>Unlocks the form set, with the option of also unlocking its forms. If the form set is already unlocked, you can still call this method to unlock its forms.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>unlockForms: If the value is <b>0</b>, any forms that are explicitly locked remain locked, even though the form set itself is unlocked. If the value is <b>1</b>, any forms that are explicitly locked are unlocked.</li> </ul> <p>Throws: <code>FinanceClientException</code></p>

The following methods are inherited from class `com.sas.solutions.finance.api.BaseApi`: `authenticate`, `buildExceptionMessageString`, `getErrorMessage`, `getMessage`, `setEnvironment`, `setLocale`, and `trim`.

The following methods are inherited from class `java.lang.Object`: `equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, and `wait`.

## The Metadata Class

### Overview

The `com.sas.solutions.finance.api.Metadata` class contains methods for looking up SAS Financial Management metadata. It extends the `com.sas.solutions.finance.api.BaseApi` class.

This class can be used for both financial planning and reporting and for operational planning.

For an example of using the Metadata class, see [Chapter 5, “Creating a Custom Cell Action,”](#) on page 73.

### Method Summary

**Table 3.9** Metadata Class Method Summary

Method	Description
<b>Metadata</b> ()	Constructor. Throws: <code>java.lang.Exception</code>
<code>java.lang.String</code> <b>getDimensionCode</b> ( <code>java.lang.String dimID</code> )	Gets the dimension code. Parameters: <ul style="list-style-type: none"> <li><code>dimID</code>: the dimension ID.</li> </ul> Returns: the dimension code that corresponds to the dimension ID
<code>java.lang.String</code> <b>getMemberCode</b> ( <code>java.lang.String dimID</code> , <code>java.lang.String memID</code> )	Gets the member code. Parameters: <ul style="list-style-type: none"> <li><code>dimID</code>: the dimension ID.</li> <li><code>memID</code>: the member ID.</li> </ul> Returns: the member code that corresponds to the specified dimension ID and member ID

The following methods are inherited from class `com.sas.solutions.finance.api.BaseApi`: `authenticate`, `buildExceptionMessageString`, `getErrorMessage`, `getMessage`, `setEnvironment`, `setLocale`, and `trim`.

The following methods are inherited from class `java.lang.Object`: `equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, and `wait`.

## The Model Class (Financial Models Only)

### Overview

The `com.sas.solutions.finance.api.Model` class contains methods for retrieving information about SAS Financial Management models and for running queries against a model. It extends the `com.sas.solutions.finance.api.BaseModel` class (which in turn extends `com.sas.solutions.finance.api.BaseApi`). This class applies only to financial models.

Most methods of the Model class now have corresponding macros (as noted in the table below). Those methods are still supported for backward compatibility. However, we recommend using the macros instead. A few methods have no macro equivalents.

*Note:* The Model class is intended to be used by administrators and power users.

Security is applied for the user who is running the query. Keep that in mind if you make the query results available to other users.

### Method Summary

**Table 3.10** Model Class Method Summary

Method	Description
<b>Model</b>	Constructor. Throws: <code>java.lang.Exception</code>
<b>Model</b> ( <code>java.lang.String storedProcessEntityKey</code> )	Constructor. This constructor can be used only in a stored process that is part of a workflow. Parameters: <ul style="list-style-type: none"> <li><code>storedProcessEntityKey</code>: the security key that is passed from the workflow.</li> </ul> Throws: <code>java.lang.Exception</code>
<b>executeQuery</b>	Deprecated. Use the <code>%FMQUERY</code> macro instead. See <a href="#">“Executing Queries with the %FMQUERY Macro”</a> on page 51.

Method	Description
int <b>generateFormulaFacts</b> (java.lang.String cycleName, java.lang.String formSetName)	<p>Computes and stores all driver formula output values for crossings in the selected form set. This method corresponds to the <b>Run driver formulas</b> option of SAS Financial Management Studio, which is used to make sure that the stored output values of all driver formulas are current.</p> <p>Before calling this method for an imported form set, save the form set template (in Microsoft Excel). Otherwise, the method returns an error. (This is also true when you select <b>Run driver formulas</b> in SAS Financial Management Studio.)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• cycleName: the name of the cycle to use.</li> <li>• formSetName: the name of the form set to use.</li> </ul> <p>Returns an integer containing the status code:</p> <ul style="list-style-type: none"> <li>• 0: SUCCESS</li> <li>• 1: OBJECT NOT FOUND</li> <li>• 2: FORM SET IS LOCKED</li> <li>• 3: GENERIC ERROR</li> </ul>
<b>getAllModels</b>	<p>Deprecated. Use the %GETALLMODELS macro instead. See <a href="#">“Model Macros” on page 44</a>.</p>
double <b>getCellValue</b> (java.lang.String resultCode, java.lang.String[] dimensionCodes, java.lang.String[] memberCodes)	<p>Gets the value of a crossing.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• resultCode: the code of the results model.</li> <li>• dimensionCodes: the list of dimension codes that define the crossing.</li> <li>• memberCodes: a matching list of member codes that define the crossing.</li> </ul> <p>If you omit a dimension, the default member for the model’s hierarchy in that dimension (at the model’s hierarchy as-of date) is used instead. For the Time dimension, this value is not necessarily the same as the default read member for the time hierarchy that can be set in the model. To avoid unexpected results, we recommend that you always include the Time dimension in your specification.</p> <p>Returns: the value of the specified crossing</p> <p>Throws: java.lang.Exception</p> <p>For an example, see <a href="#">Chapter 5, “Creating a Custom Cell Action,” on page 73</a>.</p>
<b>getModelHierarchies</b>	<p>Deprecated. Use the %GETMODELHIERARCHIES macro instead. See <a href="#">“Model Macros” on page 44</a>.</p>
<b>getModelMemberProperties</b> (	<p>Deprecated. Use the %GETMODELPROPERTIES macro instead. See <a href="#">“Model Macros” on page 44</a>.</p>

Method	Description
<b>getModelMembers</b>	Deprecated. Use the %GETMODELMEMBERS macro instead. See “ <a href="#">Model Macros</a> ” on page 44.
void <b>postAdjustments</b> (java.lang.String modelCode)	<p>Post adjustments for all time periods and analysis members of the specified model. Calling this method has the same effect as selecting <b>Post Adjustments</b> for a model in SAS Financial Management Studio.</p> <p>The postAdjustments method is useful for situations such as the following:</p> <ul style="list-style-type: none"> <li>• You need to post adjustments for a model with many adjustment rules.</li> <li>• You need to post adjustments for multiple models at a time.</li> <li>• You need to post adjustments for many time periods and analyses at a time.</li> </ul> <p>Parameters</p> <ul style="list-style-type: none"> <li>• modelCode: the model to be affected by the adjustments.</li> </ul> <p>Example:</p> <pre>j.callVoidMethod('postAdjustments',     'MyModel');</pre> <p>To confirm the completion of batch posting, check the posting status in SAS Financial Management Studio. In addition, if the posting is successful, the postAdjustments method logs an information message that begins “Posting adjustments completed successfully for model <i>model-name</i> ....”</p> <p>Throws: FinanceClientException</p>
void <b>postAdjustments</b> (java.lang.String modelCode, java.lang.String startTimeCode, java.lang.String endTimeCode, java.lang.String[] analysisCodes)	<p>Post adjustments for the specified model, time spans, and analysis members.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• modelCode: the code for the model to be affected.</li> <li>• startTimeCode: the dimension member code for the start time period.</li> <li>• endTimeCode: the dimension member code for the end time period.</li> <li>• analysisCodes: array of dimension member codes for the analysis members to be affected.</li> </ul> <p>Example:</p> <pre>array j[2] \$9 ("MyActual", "MyBudget"); j.callVoidMethod('postAdjustments', 'MyModel',     'Jan 2009', 'Feb 2009', j);</pre> <p>Throws: FinanceClientException</p>
void <b>setCode</b> (java.lang.String code)	<p>Select the model to be used by the updateTimeDefaultMembers method.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• code: the code for the model.</li> </ul>

Method	Description
<b>void updateTimeDefaultMembers</b> (java.lang.String timeDefaultReadMemberCode, java.lang.String timeDefaultWriteMemberCode)	<p>Update the default read and write members for the time hierarchy for the selected model.</p> <p>Before calling this method, you must first call the model's <code>setCode</code> method.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>timeDefaultReadMemberCode</code>: the member code for the default read member. This member must exist within the model's start and end time periods.</li> <li>• <code>timeDefaultWriteMemberCode</code>: the member code for the default write member. This member must be the same as the default read member or a descendant of the default read member.</li> </ul> <p>Example:</p> <pre>j.callVoidMethod("setCode", "Default_Model"); j.callVoidMethod("updateTimeDefaultMembers",     "Q1 2012", "Jan 2012");</pre>

*Note:* The constructor that specifies user ID and password is no longer available. Call the `METADATA_PASSID` function for authentication instead.

The following methods are inherited from class `com.sas.solutions.finance.api.BaseApi`: `authenticate`, `buildExceptionMessageString`, `getErrorMessage`, `getMessage`, `setEnvironment`, `setLocale`, and `trim`.

The following methods are inherited from class `java.lang.Object`: `equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, and `wait`.

## Model Macros

### Overview

Several macros are available for financial or operational models. These macros are intended for use only by administrators and power users. Security is applied for the user who is running the query. Keep that in mind if you make the results available to other users.

**Table 3.11** Summary of Macros

Macro	Description
<code>%GETALLMODELS</code>	Gets the available models of the specified model type.
<code>%GETFORMS</code>	Gets the forms in the specified form set.
<code>%GETFORMSETS</code>	Gets the form sets that use the specified model.
<code>%GETMODELHIERARCHIES</code>	Gets the hierarchies that are associated with the specified model.



Macro	Description
%GETMODELMEMBERS	Gets the members for the specified model.
%GETMODELPROPERTIES	Gets the member properties for the specified model.

To use these macros, first invoke the %MODEL macro. (You do not need to declare a Model object or authenticate the user; those functions are handled in the macro code.)

Here is an example:

```
%MODEL
%GETALLMODELS ('FM', 'WORK', 'ModelList')
```

*Note:* In these macros, the TRUSTEDUSERNAME and TRUSTEDPASSWORD parameters are no longer supported.

## The %GETALLMODELS Macro

### Overview

The %GETALLMODELS macro retrieves the available models of the specified model type and creates a result set with these columns: MODEL\_CD, MODEL\_NAME, and MODEL\_DESCRIPTION.

### Syntax

```
%GETALLMODELS (
    modelType
    , sasLibName
    , outputDataSetName
    [, ENVIRONMENT="default"]
    [, LOCALE="default"]
)
```

*modelType*

The type of model: 'FM' for financial models and 'OP' for operational models.

*Note:* The model type must be enclosed in single quotation marks.

*sasLibName*

The libref for the SAS library that holds the result set. This library must be defined during the current SAS session; typically, it is the WORK library.

*outputDataSetName*

The name of the result set.

ENVIRONMENT

An environment (such as “default”, “dev”, or “prod”) refers to an installation of SAS Financial Management. If you omit this parameter, “default” is used.

The environment value is site-specific. For more information, see [“Specifying the SAS Financial Management Environment” on page 14](#).

LOCALE

A locale that is specified as *language-code\_country-code*, such as **en\_US** or **es\_SP**. The *language-code* is a valid ISO language code in the form of a lowercase, two-character string, and the *country-code* is a valid ISO country code in the form of an

uppercase, two-character string. If you omit this parameter, the system default locale is used.

### Example

This example creates a result set, `ModelsOut`, containing information about the financial models that are available to this user. It uses default values for the optional parameters:

```
%MODEL
%GETALLMODELS('FM', 'Work', 'ModelsOut')
```

## The %GETFORMS Macro

### Overview

The %GETFORMS macro returns a data set with information about the forms in the specified form set. The data set contains the following columns: `FORM_ID`, `FORM_NAME`, `FORM_DESCRIPTION`, `FORM_AUTHOR`, `FORM_REVIEWER`, `FORM_DUE_DATE`, and `FORM_STATE`. The `FORM_STATE` column contains the status of the form, with a value such as `DRAFT`, `READY`, or `EDITED`.

### Syntax

```
%GETFORMS (
    modelType
    , sasLibName
    , outputDataSetName
    , modelCode
    , formSetName
    [, ENVIRONMENT=""]
    [, LOCALE="default"]
)
```

#### *modelType*

The type of model: 'FM' for financial models and 'OP' for operational models.

*Note:* The model type must be enclosed in single quotation marks.

#### *sasLibName*

The libref for the SAS library that holds the result set. This library must be defined during the current SAS session; typically, it is the WORK library.

#### *outputDataSetName*

The name of the result set.

#### *modelCode*

The code for a model that is associated with the form set. The model code is used only to retrieve the correct cycle. If the form set uses more than one model, select any one of the models.

#### *formSetName*

The name of the form set.

#### ENVIRONMENT

An environment (such as “default”, “dev”, or “prod”) refers to an installation of SAS Financial Management. If you omit this parameter, “default” is used.

The environment value is site-specific. For more information, see [“Specifying the SAS Financial Management Environment” on page 14](#).

**LOCALE**

A locale that is specified as *language-code\_country-code*, such as **en\_US** or **es\_SP**. The *language-code* is a valid ISO language code in the form of a lowercase, two-character string, and the *country-code* is a valid ISO country code in the form of an uppercase, two-character string. If you omit this parameter, the system default locale is used.

**Example**

This example creates a result set, FormsOut, containing information about the forms that are part of the MyFormSet form set.

```
%MODEL
%GETFORMS('FM', 'Work', 'FormsOut', 'MyModel', 'MyFormSet')
```

**The %GETFORMSETS Macro****Overview**

The %GETFORMSETS macro returns a data set with information about the form sets that use the specified model. The data set contains the following columns: FORMSET\_ID, FORMSET\_NAME, FORMSET\_DESCRIPTION, and FORMSET\_STATUS.

The FORMSET\_STATUS column can have one of the following values:

- 0: Processing
- 1: Draft
- 2: Published
- 4: Complete

**Syntax**

```
%GETFORMSETS (
    modelType
    , sasLibName
    , outputDataSetName
    , modelCode
    [, ENVIRONMENT=""]
    [, LOCALE="default"]
)
```

*modelType*

The type of model: 'FM' for financial models and 'OP' for operational models.

*Note:* The model type must be enclosed in single quotation marks.

*sasLibName*

The libref for the SAS library that holds the result set. This library must be defined during the current SAS session; typically, it is the WORK library.

*outputDataSetName*

The name of the result set.

*modelCode*

The code for the associated model.

**ENVIRONMENT**

An environment (such as “default”, “dev”, or “prod”) refers to an installation of SAS Financial Management. If you omit this parameter, “default” is used.

The environment value is site-specific. For more information, see [“Specifying the SAS Financial Management Environment” on page 14](#).

**LOCALE**

A locale that is specified as *language-code\_country-code*, such as **en\_US** or **es\_SP**. The *language-code* is a valid ISO language code in the form of a lowercase, two-character string, and the *country-code* is a valid ISO country code in the form of an uppercase, two-character string. If you omit this parameter, the system default locale is used.

**Example**

This example creates a result set, FormsetsOut, containing information about the form sets that use MyModel.

```
%MODEL
%GETFORMSETS('FM','Work','FormsetsOut','MyModel')
```

**The %GETMODELHIERARCHIES Macro****Overview**

The %GETMODELHIERARCHIES macro creates a result set with information about the hierarchies that are associated with the specified model. The result set contains these columns: DIMENSION\_TYPE\_CD, DIMENSION\_CD, DIMENSION\_NAME, DIMENSION\_DESCRIPTION, HIERARCHY\_CD, HIERARCHY\_NAME, and HIERARCHY\_DESCRIPTION.

**Syntax**

```
%GETMODELHIERARCHIES (
    modelType
    , sasLibName
    , outputDataSetName
    , modelCode
    [, ENVIRONMENT=""]
    [, LOCALE="default"]
)
```

*modelType*

The type of model: 'FM' for financial models and 'OP' for operational models.

*Note:* The model type must be enclosed in single quotation marks.

*sasLibName*

The libref for the SAS library that holds the result set. This library must be defined during the current SAS session; typically, it is the WORK library.

*outputDataSetName*

The name of the result set.

*modelCode*

The code for the associated model.

**ENVIRONMENT**

An environment (such as “default”, “dev”, or “prod”) refers to an installation of SAS Financial Management. If you omit this parameter, “default” is used.

The environment value is site-specific. For more information, see [“Specifying the SAS Financial Management Environment” on page 14](#).

**LOCALE**

A locale that is specified as *language-code\_country-code*, such as **en\_US** or **es\_SP**. The *language-code* is a valid ISO language code in the form of a lowercase, two-character string, and the *country-code* is a valid ISO country code in the form of an uppercase, two-character string. If you omit this parameter, the system default locale is used.

**Example**

This example creates a result set, HierarchiesOut, containing information about the hierarchies that are part of MyModel.

```
%MODEL
%GETMODELHIERARCHIES ('FM', 'Work', 'HierarchiesOut', 'MyModel')
```

**The %GETMODELMEMBERS Macro****Overview**

The %GETMODELMEMBERS macro retrieves a model's members and creates a result set with these columns: DIMENSION\_TYPE\_CD, HIERARCHY\_CD, MEMBER\_CD, MEMBER\_NAME, MEMBER\_DESCRIPTION, HIERARCHY\_LEVEL, HIERARCHY\_ORDER, PARENT\_CD, and IS\_LEAF. A value of 1 for IS\_LEAF signifies a leaf member. Otherwise, the value is 0.

**Syntax**

```
%GETMODELMEMBERS (
    modelType
    , sasLibName
    , outputDataSetName
    , modelCode
    [, ENVIRONMENT=""]
    [, LOCALE="default"]
)
```

*modelType*

The type of model: 'FM' for financial models and 'OP' for operational models.

*Note:* The model type must be enclosed in single quotation marks.

*sasLibName*

The libref for the SAS library that holds the result set. This library must be defined during the current SAS session; typically, it is the WORK library.

*outputDataSetName*

The name of the result set.

*modelCode*

The code for the associated model.

**ENVIRONMENT**

An environment (such as “default”, “dev”, or “prod”) refers to an installation of SAS Financial Management. If you omit this parameter, “default” is used.

The environment value is site-specific. For more information, see [“Specifying the SAS Financial Management Environment” on page 14](#).

**LOCALE**

A locale that is specified as *language-code\_country-code*, such as **en\_US** or **es\_SP**. The *language-code* is a valid ISO language code in the form of a lowercase, two-character string, and the *country-code* is a valid ISO country code in the form of an uppercase, two-character string. If you omit this parameter, the system default locale is used.

**Example**

This example creates a result set with information about the members that are part of MyModel:

```
%MODEL
%GETMODELMEMBERS ('FM', 'Work', 'ModelMembersOut', 'MyModel')
```

**The %GETMODELPROPERTIES Macro****Overview**

The %GETMODELPROPERTIES macro retrieves the member properties for the specified model and creates a result set with the following columns: DIMENSION\_TYPE\_CD, HIERARCHY\_CD, MEMBER\_CD, PROPERTY\_CD, PROPERTY\_NAME, and PROPERTY\_VALUE. You can limit the results by specifying dimension type codes, property codes, and/or member codes as parameters.

*Note:* The version of this macro that is in fmmodel.sas has been deprecated. Use this version instead.

**Syntax**

**%GETMODELPROPERTIES (**

```
    modelType
    , sasLibName
    , outputDataSetName
    , modelCode
    [, ENVIRONMENT=""]
    [, DIMTYPECODES=""]
    [, PROPERTYCODES=""]
    [, MEMBERCODES=""]
    [, LOCALE="default"]
    [, DELIM=';']
)
```

*modelType*

The type of model: 'FM' for financial models and 'OP' for operational models.

*Note:* The model type must be enclosed in single quotation marks.

*sasLibName*

The libref for the SAS library that holds the result set. This library must be defined during the current SAS session; typically, it is the WORK library.

*outputDataSetName*

The name of the result set.

*modelCode*

The code for the associated model.

ENVIRONMENT

An environment (such as “default”, “dev”, or “prod”) refers to an installation of SAS Financial Management. If you omit this parameter, “default” is used.

The environment value is site-specific. For more information, see [“Specifying the SAS Financial Management Environment” on page 14](#).

LOCALE

A locale that is specified as *language-code\_country-code*, such as **en\_US** or **es\_SP**.

The *language-code* is a valid ISO language code in the form of a lowercase, two-character string, and the *country-code* is a valid ISO country code in the form of an uppercase, two-character string. If you omit this parameter, the system default locale is used.

DIMTYPECODES

A delimited list of unquoted dimension type codes to use in the query. By default, all dimension type codes are used.

PROPERTYCODES

A delimited list of unquoted property codes to use in the query. By default, all properties are used.

MEMBERCODES

A delimited list of unquoted member codes to use in the query. By default, all members are used.

DELIM

The delimiter to be used to parse the input (dimension type codes, property codes, and member codes). By default, the macro expects a semicolon.

### **Example**

This example retrieves only the AccountType and AccountBehavior properties for the ACCOUNT dimension type:

```
%MODEL
%GETMODELPROPERTIES('FM', 'Work', 'PropertiesOut', 'MyModel',
  DIMTYPECODES='ACCOUNT', PROPERTYCODES='AccountType;AccountBehavior')
```

---

## Executing Queries with the %FMQUERY Macro

### **Overview**

The %FMQUERY macro executes a query against a model. The macro returns the same result as a query in Excel, including calculated members. Use this macro instead of the executeQuery method of the Model class.

*Note:* This macro applies only to financial models.

The %FMQUERY macro is intended to be used only by administrators and power users. Security is applied to a query for the user who is running the query. (Keep that in mind if you make the query results available to other users.)

*Note:* The RUNASUSERID, TRUSTEDUSERNAME, and TRUSTEDPASSWORD parameters are no longer supported.

## Query Types

The %FMQUERY macro supports two types of queries:

- **MDX queries:** Queries that use MDX syntax, which is similar to SQL syntax.
- **Non-MDX queries:** Queries that are based on a model code and a data set that contains query parameters.

## Syntax

### %FMQUERY (

```

    localSasLibName
    , resultDataSetName
    [, MDXSTRING=""]
    [, MODELCODE=""]
    [, SASLIBNAME=""]
    [, QUERYDATASETNAME=""]
    [, FILTEROPTS=0]
    [, MEMBEROPTS=0]
    [, ENVIRONMENT="default"]

```

)

#### *localSasLibName*

The libref for the SAS library that holds the result set. This library must be defined during the current SAS session; typically, it is the WORK library.

#### *resultDataSetName*

The name of the result set to be produced by the query. It contains the following columns:

DIMENSION\_TYPE\_CD: the member code for each dimension type. The calling routine must handle illegal characters in the member codes.

VALUE: the corresponding value. **NaN** is represented as a period (.).

Records are filtered according to the *filterOptions*.

If the query fails and the result data set already exists, the data set is not deleted.

#### MDXSTRING

The query to be executed. This parameter is required if you are performing an MDX-style query.

*Note:* For non-MDX-style queries, use the QUERYDATASETNAME parameter instead.

#### MODELCODE

The identifier for the results model to be used in the query.

*Note:* This parameter is not used for MDX-style queries.

#### SASLIBNAME

The libref for the SAS library that holds the query data set. This library must be registered in the metadata repository.



**QUERYDATASETNAME**

The name of the SAS table that contains the query. This table must exist before you call %FMQUERY. For details, see [“The Query Data Set” on page 53](#).

This parameter is not used for MDX-style queries. Instead, use the MDXSTRING parameter.

**FILTEROPTS**

A value that specifies filters to be applied to the result set. Valid options are:

- 0: include all crossings (default)
- 1: exclude missing values
- 2: exclude zero values
- 3: exclude missing and zero values

**MEMBEROPTS**

Additional member attributes, including hierarchical ordering, to be printed beside the member codes. The parameter can have any combination of these values:

- 0: include only member code (\_CD) columns.
- 1: include member name (\_NAME) columns.
- 2: include member description (\_DESC) columns.
- 4: include member hierarchy sort (\_SORT) columns. The sort values are represented as hierarchical child numbering of the member starting from the root of the hierarchy (such as 1.4.2.5).

Regardless of other options, the member code column is always printed. The options can be used in any combination. For example, a value of 5 includes the member code (always), the member name, and the member hierarchy sort columns.

**ENVIRONMENT**

An environment (such as **default**, **dev**, or **prod**) refers to an installation of SAS Financial Management.

The environment value is site-specific. For more information, see [“Specifying the SAS Financial Management Environment” on page 14](#).

If you omit a dimension type in your query, the default member for the model’s hierarchy in that dimension (at the model’s hierarchy as-of date) is used instead. For the Time dimension, this value is not necessarily the same as the default read member for the time hierarchy that can be set in the model. To avoid unexpected results, we recommend that you always include the Time dimension in your specification. This applies to both MDX and non-MDX queries.

## ***The Query Data Set***

For non-MDX queries, one parameter of the %FMQUERY macro is QUERYDATASETNAME, the name of a table that contains the query. This table must exist before you call the macro, and it must reside in the same library as the result set that is produced by the query.

*Note:* This parameter is not used for MDX queries.

The table has the following columns:

**Table 3.12** Contents of the Query Data Set

Column	Description	Data Type
DIMENSION_TYPE_CD	Dimension type code	character
MEMBER_CD	Member code. The dimension type code and member code pair define the root of the subtree to be queried.	character
INCLUDE_MEMBER	0: exclude the member 1: include the member	numeric
INCLUDE_LEAVES	0: exclude leaves 1: include first-level leaves 2: include all levels of leaves 3: include first-level leaves and virtual children 4: include all levels of leaves and virtual children	numeric
INCLUDE_ROLLUPS	0: exclude roll-ups 1: include first-level roll-ups 2: include all levels of roll-ups	numeric

**%FMQUERY Example (Non-MDX)**

This example executes a query against a fictitious model that is named TESTING18\_MODEL. The query data set name is QUERYPARAMETERS. In this example, the results are written to the NONMDXRESULTDATASETNAME data set in the WORK library.

**Example Code 3.2** Non-MDX Query

```
LIBNAME stagefm BASE "C:\SAS\Config\Levl\SASApp\Data\FinancialManagement\StageFM";

data stagefm.queryParameters;
    length DIMENSION_TYPE_CD MEMBER_CD $32;
    DIMENSION_TYPE_CD = "ACCOUNT"; MEMBER_CD = "A8420"; INCLUDE_MEMBER=1;
    INCLUDE_LEAVES=0; INCLUDE_ROLLUPS=0; output;
    DIMENSION_TYPE_CD = "TIME"; MEMBER_CD = "DEC1997"; INCLUDE_MEMBER=1;
    INCLUDE_LEAVES=0; INCLUDE_ROLLUPS=0; output;
    DIMENSION_TYPE_CD = "CURRENCY"; MEMBER_CD = "USD"; INCLUDE_MEMBER=1;
    INCLUDE_LEAVES=0; INCLUDE_ROLLUPS=0; output;
run;
%fmquery(modelCode="testing18_model",localSasLibName="Work", sasLibName="stagefm",
    queryDataSetName="queryParameters", resultDataSetName="NONMDXResultDataSetName",
    environment="default")
```

**%FMQUERY Example with MDX String**

Here is an example of calling %FMQUERY using an MDX string:

**Example Code 3.3** Query Using an MDX String

```
%fmquery("Work", "MDXResultDataSetName",
  mdxString="SELECT {ACCOUNT.A8420} on 0 FROM testing18_model WHERE (TIME.DEC1997, CURRENCY.USD)",
  environment="default")
```

*Note:* The mdxString cannot include a line break.

For one approach to creating an MDX string, see [“Copying an MDX String” on page 55](#).

For MDX reference information, see [“MDX Reference for SAS Financial Management” on page 55](#).

*Note:* Currently, MDX queries in SAS Financial Management do not support the equivalent of the INCLUDE\_MEMBER, INCLUDE\_LEAVES, or INCLUDE\_ROLLUPS options (that are available in non-MDX queries). In an MDX query, you must specify each member separately. To include leaves for one or more dimensions, specify those leaf members in the MDX string.

**Copying an MDX String**

To create an MDX string, one simple approach is to save the string that is created when you insert a Read-only table in Microsoft Excel. Follow these steps:

1. In Microsoft Excel, log on to the middle tier.
2. Insert a Read-only table.
3. Open the table properties.
4. Select the **Dimensions** tab.
5. Click **Query Diagnostics**.
6. Click **Copy ODCS MDX String to Clipboard**.

The MDX string for the Read-only table is available on the Windows clipboard.

**MDX Reference for SAS Financial Management****Overview**

Via ODCS, SAS Financial Management supports simple MDX queries that extend the capabilities that are available with the standard query parameters.

Previously, complex queries required exploding the cube or running multiple, smaller queries. By stacking multiple dimensions on an axis, MDX allows clients to express the specific query that they need.

Only a subset of MDX functionality is currently supported in ODCS:

- basic queries: **SELECT ... FROM ... WHERE ...**
- basic member functions

More sophisticated features are not currently supported. For example, these features are not currently supported:

- creating or manipulating metadata
- defining calculated members
- more advanced functions, such as filter, aggregate, and non-empty

- anything that is defined on a WITH clause

### Members

A member is represented as *DimensionTypeCode.MemberCode*. For example:

- `CURRENCY.USD`
- `TIME.Jan2001`
- `INTORG.Legal`

*Note:* Standard MDX and OLAP do not have the concept of dimension types. Instead, they use dimension codes to define members. ODCS uses dimension types, because they make it easier to reuse queries between virtual cubes (vcubes). In this MDX reference, references to dimensions and dimension types are interchangeable.

All codes in ODCS are case sensitive. If a dimension type code or member code includes a non-alphanumeric character, the code must be wrapped in square brackets, as in these examples:

- `INTORG.[R&D]`
- `ANALYSIS.[My Analysis]`
- `PRODUCT.[Hershey's Kisses]`
- `[CUSTOM TYPE].[My Member]`

A member function can be appended to a member using the following syntax:

*DimensionTypeCode.MemberCode.Function*

An example is the VC function, a SAS Financial Management function that returns the virtual child of the member:

- `INTORG.Legal.VC`
- `PRODUCT.[Hershey's Kisses].VC`

(In MDX, the virtual child is known as a DataMember.)

### Tuples

A tuple is a combination of members from one or more dimensions, with only one member from each dimension. You can think of it as a multidimensional member. The simplest example of a tuple has one member, such as `INTORG.Legal`.

When there are multiple members on a tuple, the members are separated by commas and the entire tuple is wrapped in parentheses, as in these examples:

- `(INTORG.Legal, TIME.Jan2001)`
- `(INTORG.Legal, TIME.Jan2001, PRODUCT.[Hershey's Kisses])`
- `(INTORG.Legal, TIME.Jan2001, PRODUCT.[Hershey's Kisses], CURRENCY.USD, ANALYSIS.Actuals)`

It is important to remember that tuples can have only one member from each dimension. The following tuples are invalid because they have multiple members from the same dimension:

- `(INTORG.Legal, TIME.Jan2001, TIME.Feb2001)`  
Invalid: two members from the TIME dimension.
- `(INTORG.Legal, TIME.Jan2001, INTORG.[R&D])`

Invalid: two members from the INTORG dimension.

### **Tuple Sets: {}s**

A tuple set is an ordered collection of tuples. A tuple set can have one tuple, multiple tuples, or even zero tuples. Within a set, tuples can be repeated.

*Note:* This definition differs from the mathematical definition of a set or the Set data structures in Java.)

The tuples in a set can have one or more members. A set is wrapped in braces, and the tuples are separated by commas. Here are some examples:

- { INTORG.Legal, INTORG.[R&D] }  
Set with two tuples, each containing one member.
- { (INTORG.Legal, TIME.Jan2001) }  
Set with one tuple (wrapped in parentheses), containing two members.
- { (INTORG.Legal, TIME.Jan2001), (INTORG.[R&D], TIME.Feb2001) }  
Set with two tuples, each tuple containing two members.
- { (INTORG.Legal, TIME.Jan2001, ANALYSIS.Actuals), (INTORG.[R&D], TIME.Feb2001, ANALYSIS.Budget) }  
Set with two tuples, each tuple containing three members.
- { (INTORG.Legal, TIME.Jan2001), (INTORG.[R&D], TIME.Feb2001), (INTORG.[R&D], TIME.Feb2001) }  
Set with three tuples, each tuple containing two members. One tuple is repeated.

All tuples in a set must have the same dimensions represented, and the dimensions must be in the same order. This is called the dimensionality of the tuple. Notice that all of the examples above meet this requirement. The last example has three tuples, each with two members. All three tuples contain the same dimensions and specify the INTORG dimension first and the TIME dimension second. Thus, they have the same dimensionality.

The following sets are invalid because they do not have the same dimensionality:

- { (INTORG.Legal, TIME.Jan2001), (INTORG.[R&D], ANALYSIS.Budget) }  
Invalid: TIME and ANALYSIS are different dimensions.
- { (INTORG.Legal, TIME.Jan2001), (TIME.Feb2001, INTORG.[R&D]) }  
Invalid: tuple dimensions are not in the same order.
- { (INTORG.Legal), (ANALYSIS.Actuals) }  
Invalid: INTORG and ANALYSIS are different dimensions.
- { INTORG.Legal, ANALYSIS.Actuals }  
Invalid: INTORG and ANALYSIS are different dimensions.

This example might look like a single tuple with two members. However, it is actually a tuple set with two tuples, each containing one member (using the convention of omitting parentheses for a tuple with a single member). Because the members are from different dimensions, the tuple set is invalid.

### Basic Query Syntax

The MDX query syntax enables you to define the view of the data that you want returned. Syntactically, it is similar to an SQL query. The basic syntax of a SELECT clause is as follows:

This simple query retrieves data with TIME members on the columns and INTORG members on the rows:

- `SELECT {TIME.Jan2001, TIME.Feb2001, TIME.Mar2001, TIME.Q12001} ON COLUMNS, {INTORG.Legal, INTORG.[R&D]} ON ROWS FROM [My VCube]`

*Note:* The example queries in this chapter contain line breaks only so that they fit on the page. In the %FMQUERY macro, MDX query strings cannot contain a line break. In addition, keywords are shown in upper case. However, MDX queries are not case sensitive.

The results would resemble the following:

	TIME.Jan2001	TIME.Feb2001	TIME.Mar2001	TIME.Q12001
INTORG.Legal	2	6	10	18
INTORG.[R&D]	10	40	20	70

The SELECT clause defines one or more axes, with each axis assigned a position on the table (columns or rows). The example above defines two axes: TIME on columns and INTORG on rows. Notice the braces in the row axis definition, denoting a tuple set. Each tuple in the set contains only one member. However, like any tuple set, it can contain multiple members. This feature enables you to stack multiple dimensions on an axis, mixing and matching members between dimensions.

The following example crosses the INTORG members with different ANALYSIS members on the rows:

- `SELECT {TIME.Jan2001, TIME.Feb2001, TIME.Mar2001, TIME.Q12001} ON COLUMNS, { (INTORG.Legal, ANALYSIS.Actuals), (INTORG.[R&D], ANALYSIS.Budget) } ON ROWS FROM [My VCube]`

The results would resemble the following:

	TIME.Jan2001	TIME.Feb2001	TIME.Mar2001	TIME.Q12001
INTORG.Legal ANALYSIS.Actuals	2	6	10	18
INTORG.[R&D] ANALYSIS.Budget	20	60	15	95

### WHERE Clause: Defining a Slicer

The previous examples use only two or three dimensions in the queries. For any dimensions in the cube that were not specified (such as CURRENCY, PRODUCT, or ACCOUNT), the default member for the dimension is implicitly used in the query.

What if you want to cross your table with members that are not default members? In MDX, you can use a WHERE clause to define members that apply to the entire table. This clause is known as a slicer. The example below defines a slicer for three dimensions that are not shown on the table:

- `SELECT {TIME.Jan2001, TIME.Feb2001, TIME.Mar2001, TIME.Q12001} ON COLUMNS, {INTORG.Legal, INTORG.[R&D]} ON ROWS FROM [My VCube] WHERE (CURRENCY.USD, ANALYSIS.Budget, FREQUENCY.PA)`

Results would resemble the following:

Slicer: CURRENCY.USD, ANALYSIS.Budget, FREQUENCY.PA				
	TIME.Jan2001	TIME.Feb2001	TIME.Mar2001	TIME.Q12001
INTORG.Legal	4	8	12	24
INTORG.[R&D]	20	60	15	95

Notice that the slicer in the WHERE clause is enclosed in parentheses: it is really just a tuple. Like any tuple, it can contain one or more members, and the members must be from different dimensions. In addition, the slicer in the tuple cannot contain a member from a dimension that is used in one of the axes. The following example is invalid because it uses the TIME dimension on both the rows and the slicer:

- `SELECT {TIME.Jan2001, TIME.Feb2001, TIME.Mar2001, TIME.Q12001} ON COLUMNS, {INTORG.Legal, INTORG.[R&D]} ON ROWS FROM [My VCube] WHERE (TIME.Apr2001, ANALYSIS.Budget)`

### ***SELECT Clause: Defining Axes***

So far, all the query examples have used only two axes: columns and rows. However, an MDX query can have anywhere from 0–64 axes. Beyond COLUMNS and ROWS, the axis keywords are PAGES, CHAPTERS, and SECTIONS. Here are examples of queries that use a different number of axes:

- `SELECT {TIME.Jan2001, TIME.Feb2001} ON COLUMNS FROM [My VCube] WHERE (CURRENCY.USD)`
- `SELECT {TIME.Jan2001, TIME.Feb2001} ON COLUMNS, {INTORG.Legal} ON ROWS, {ANALYSIS.Actuals, ANALYSIS.Budget} ON PAGES FROM [My VCube] WHERE (CURRENCY.USD)`
- `SELECT {TIME.Jan2001, TIME.Feb2001} ON COLUMNS, {INTORG.Legal} ON ROWS, {ANALYSIS.Actuals, ANALYSIS.Budget} ON PAGES, {FREQUENCY.PTD} ON CHAPTERS FROM [My VCube] WHERE (CURRENCY.USD)`
- `SELECT {TIME.Jan2001, TIME.Feb2001} ON COLUMNS, {INTORG.Legal} ON ROWS, {ANALYSIS.Actuals, ANALYSIS.Budget} ON PAGES, {FREQUENCY.PTD} ON CHAPTERS, {PRODUCT.Widgets, PRODUCT.Gadgets} ON SECTIONS FROM [My VCube] WHERE (CURRENCY.USD)`

Instead of using the axis keywords such as COLUMNS or PAGES, you can refer to axes by numbers, beginning with 0 (where 0=COLUMNS, 1=ROWS, 2=PAGES,

3=CHAPTERS, and 4=SECTIONS). Beyond sections, you must use numbers. The following queries are the same as the examples above, except that they use axis numbers instead of keywords:

- `SELECT {TIME.Jan2001, TIME.Feb2001} ON 0 FROM [My VCube] WHERE (CURRENCY.USD)`
- `SELECT {TIME.Jan2001, TIME.Feb2001} ON 0, {INTORG.Legal} ON 1, {ANALYSIS.Actuals, ANALYSIS.Budget} ON 2 FROM [My VCube] WHERE (CURRENCY.USD)`
- `SELECT {TIME.Jan2001, TIME.Feb2001} ON 0, {INTORG.Legal} ON 1, {ANALYSIS.Actuals, ANALYSIS.Budget} ON 2, {FREQUENCY.PTD} ON 3 FROM [My VCube] WHERE (CURRENCY.USD)`
- `SELECT {TIME.Jan2001, TIME.Feb2001} ON 0, {INTORG.Legal} ON 1, {ANALYSIS.Actuals, ANALYSIS.Budget} ON 2, {FREQUENCY.PTD} ON 3, {PRODUCT.Widgets, PRODUCT.Gadgets} ON 4 FROM [My VCube] WHERE (CURRENCY.USD)`

*Note:* You cannot skip axis definitions. For example, you cannot specify 0 and 2 and omit 1.

### **Specifying Excluded Members**

In an ODCS query, you can specify excluded members (members on an axis that should be ignored while running a query). Because there is no equivalent concept in MDX, ODCS supports an MDX extension for using this functionality in SAS Financial Management. At the end of a query, you can add an EXCLUDE clause to specify the members to be excluded from the query. For each dimension from which you want to exclude members, the EXCLUDE clause contains a tuple set separated by commas, as in these examples:

- `SELECT {TIME.Jan2001, TIME.Feb2001, TIME.Mar2001, TIME.Q12001} ON COLUMNS, { INTORG.All } ON ROWS FROM [My VCube] EXCLUDE { INTORG.Legal }`
- `SELECT {TIME.Jan2001, TIME.Feb2001, TIME.Mar2001, TIME.Q12001} ON COLUMNS, { INTORG.All } ON ROWS FROM [My VCube] WHERE ( CURRENCY.USD ) EXCLUDE { INTORG.Legal, INTORG.[R&D] }`
- `SELECT {TIME.Jan2001, TIME.Feb2001, TIME.Mar2001, TIME.Q12001} ON COLUMNS, { INTORG.All } ON ROWS, { PRODUCT.All } ON PAGES FROM [My VCube] WHERE ( CURRENCY.USD ) EXCLUDE { INTORG.Legal, INTORG.[R&D] }, { PRODUCT.Widgets }`

Notice that each set corresponds to a dimension in the query, and each tuple in the set contains only one member.

### **Supported Member Functions**

ODCS supports a limited number of functions:

#### **.VC**

Uses the virtual child of the member. Examples:

- `INTORG.Legal.VC`



- **INTORG.[R&D].VC**

#### **.DataMember**

MDX term for the ODCS term “virtual child.” This function is interchangeable with the .VC function. Example: **INTORG.Legal.DataMember**

#### **.Ignore**

Placeholder member that is never calculated. This function is used by Excel to overlay client-side calculations after the MDX table is returned. Only the dimension type code must be valid; the member code is ignored by the server. Here is an example: **PRODUCT.MyClientSideCalc.Ignore**

### **ODCS versus Standard OLAP**

The ODCS architecture differs from standard OLAP in a few ways. These differences affect MDX usage and syntax support:

- ODCS supports only a single, numeric measure. Therefore, there is never a need to use the MEASURES keyword in a query.
- Levels are not supported explicitly in ODCS, except for certain dimensions such as TIME. Currently, there is no support for referencing Levels in the query syntax.
- In ODCS, members in the same dimension must have a unique code. Because a cube has only one dimension for each dimension type, a member code is always unique in a given dimension type at query time.

This requirement provides the shortcut when defining member definitions of **DimensionTypeCode.MemberCode**, such as **TIME.Jan05**. If ODCS supported non-unique member codes in a dimension, you would need to follow the MDX standard and specify the ancestors of the member, such as **TIME.2005.Q1.Jan**.



## Chapter 4

# Customizing a Workflow

---

<b>About Customizing a Workflow</b> .....	<b>63</b>
<b>Workflow Types</b> .....	<b>63</b>
Overview .....	63
Top-Down Workflow .....	64
Bottom-Up Workflow .....	64
<b>Adding Your Custom Code to a Workflow</b> .....	<b>64</b>
The Pre and Post Classes .....	64
Steps in Customizing a Workflow .....	64
The Resource File .....	65
<b>Data Validation Example</b> .....	<b>68</b>
About the Data Validation Example .....	68
Code for the Example .....	69
Registering the Stored Process .....	71
Updating the Resource File .....	71

---

## About Customizing a Workflow

*Note:* This chapter applies to both financial form sets and operational form sets.

In SAS Financial Management, a workflow defines the review and approval process used in budgeting, forecasting, and other planning activities. Each workflow consists of a collection of states (such as READY, EDITED, and COMPLETE) and actions (such as EDIT and SUBMIT). At run time, the actions advance the workflow from one state to the next. Each action triggers a corresponding policy file (code that is associated with these actions).

You can customize a workflow by writing a stored process that executes before or after the workflow is advanced. This chapter explains how to add your custom code to a workflow. It also contains a short example of a workflow stored process.

---

## Workflow Types

### Overview

SAS Financial Management supports two types of workflows: top-down and bottom-up.

*Note:* For more information about SAS Financial Management workflows, see the *SAS Financial Management: User's Guide* or the online Help for the SAS Financial Management Add-In for Microsoft Excel.

### **Top-Down Workflow**

A top-down workflow enables users at any roll-up point to make bulk updates and adjustments down and across multiple entities and dimensions.

A data-entry project that has a top-down workflow begins when a top-down form set is published from SAS Financial Management Studio. The workflow ends when a Finance Process Administrator applies the COMPLETE action to the form set in SAS Financial Management Studio.

### **Bottom-Up Workflow**

In a bottom-up workflow, forms begin at the lower levels of the hierarchy and are aggregated and reviewed by the organization as they move up an approval hierarchy. (Optional) A bottom-up workflow can be connected to a separate reviewer workflow that supports additional reviewers in the budget approval process.

A bottom-up workflow begins when a bottom-up form set is published from SAS Financial Management Studio. The workflow ends when a Finance Process Administrator applies the COMPLETE action to the form set in SAS Financial Management Studio.

---

## **Adding Your Custom Code to a Workflow**

### **The Pre and Post Classes**

Two Java classes (Pre and Post) form the bridges between the SAS Financial Management workflow system and the SAS stored processes in which the customized code is deployed.

Whenever a policy file is triggered, the Pre.invoke method is called before the policy file is executed, and the Post.invoke method is called after the policy file is executed. These methods call a stored process if one is linked to this part of the workflow.

If the stored process fails (due to exception or error in the customized codes), the workflow does not advance to the next state.

- If the Pre operation fails, the policy file is not executed.
- If the Post operation fails, the workflow is rolled back to its previous state.

However, if the stored process itself makes any changes, such as updating the database, those changes remain.

### **Steps in Customizing a Workflow**

Do not modify the Pre and Post classes directly. To customize the workflow, follow these steps:

1. Write a SAS stored process to perform the necessary business logic.

The stored process must set the FM\_SP\_RESULT environment variable. If the operation fails, the program should set FM\_SP\_RESULT to **INVALID** and set the FM\_SP\_MESSAGE environment variable to an appropriate text message. Otherwise, the stored process should set FM\_SP\_RESULT to **VALID**.

If the value of FM\_SP\_RESULT is **INVALID**, an exception is thrown and the workflow is not advanced to the next state. You can make the value of the FM\_SP\_MESSAGE environment variable available in the stored process log.

For information about writing a stored process, see the *SAS Stored Processes: Developer's Guide*. For an example stored process for workflow customization, see “[Data Validation Example](#)” on page 68.

2. On the data tier, save the stored process in a directory such as **SAS-config-dir\Lev1\SASApp\SASEnvironment\FinancialManagement\SASCode\UserDefined**. (Create the **UserDefined** directory if it does not already exist.)
3. Log on to SAS Management Console as an administrator and register the stored process in the **/Products/SAS Financial Management/Customized workflow** folder. (You might need to create this folder.)
4. Create a resource file that links a workflow action to the stored process. If the resource file already exists, update the file with information about the new stored process. See “[The Resource File](#)” on page 65.

## The Resource File

### Update the Resource File

The resource file is an XML file that provides the location of a stored process and associates it with a specific form set and an action. A template for a resource file follows:

```
<SASWorkflowCustomizations>
  <Application name="SAS Financial Management">
    <Object type="FormSet" name="form_set_ID">
      <Action type="action_type">
        <Execute type="execute_type"
          storedProcessFullPath="path_to_stp"/>
      </Action>
    </Object>
  </Application>
</SASWorkflowCustomizations>
```

Replace the italicized strings with the appropriate values:

- *execute\_type* specifies when the stored process is called, relative to execution of the policy file. It must have a value of **pre** or **post**.
- *action\_type* is an action such as **SUBMIT** or **REJECT**.

For a list of available action types, see [Table 4.1 on page 66](#). Notice that some actions are available only in a top-down workflow or only in a bottom-up workflow.

- *path\_to\_stp* is the path to the stored process metadata definition, such as **/Products/SAS Financial Management/Customized workflow/MyCustomWorkflow**.

You can link the same stored process to more than one form set or action: just create a separate <Object> entry for each form set, action type, and execute type combination.

- *form\_set\_ID* is the ID of the form set to which the action applies. To look up a form set ID in the SASSDM database, you can use the following SQL query:

```
"select form_set_id from sassdm.sas_form_set where form_set_nm='form-set-name'"
```

Here is an example:

**Example Code 4.1** Example Resource File

```
<SASWorkflowCustomizations>
  <Application name="SAS Financial Management">
    <Object type="FormSet" name="2">
      <Action type="SUBMIT">
        <Execute type="pre" storedProcessFullPath= _
          "/Products/SAS Financial Management/Customized workflow/MyCustomWorkflow"/>
      </Action>
    </Object>
  </Application>
</SASWorkflowCustomizations>
```

*Note:* Line breaks (“\_”) added for readability.

**Table 4.1** Available Workflow Actions

Action Type	Top-down Workflow	Bottom-up Workflow	Description
SUBMIT		√	Submits a form for approval.
EDIT			Opens a form for editing.
REVIEW		√	Opens a form in Read-Only mode so that it can be reviewed.
REJECT		√	Changes the form's state to REJECTED and notifies the user who submitted the form.
APPROVE		√	Approves a form and copies that form's data to its parent form.
RECALL	√	√	Recalls a form so that it can be further edited and then pushed again or resubmitted.
PUSH	√		<p>Makes a form available to the users who are responsible for the top member's children. The amounts that have been allocated to the children of that member are copied to the forms for those child members.</p> <p>As a result, the users who are responsible for the child members to edit their forms, allocate the pushed amounts to the next level of child members, and then push their forms in turn.</p>

Action Type	Top-down Workflow	Bottom-up Workflow	Description
PUSHTOALL	√		<p>Makes a form available to the users who are responsible for all the top member's descendants. The amounts that have been allocated to the descendants of that member are copied to the forms for those descendant members.</p> <p>As a result, the users who are responsible for the descendant members to edit their forms. However, their editing is limited to redistributing amounts within their target member. No other user can push amounts to the next level of child members because PUSHTOALL cascades all the way down the target hierarchy in a single step.</p>
COMPLETE	√	√	Ends the workflow. This action can be performed only by a Finance Process Administrator.
UNCOMPLETE	√	√	Reactivates a form for further work. This action can be performed only by a Finance Process Administrator.

You can associate as many actions with a form set as necessary, but each action can have only one stored process associated with it. On the other hand, you are free to associate the same stored process with multiple actions in multiple form sets, if applicable.

Name the file **WorkflowCustomizations.xml** and save it on the middle tier, where the Web application server resides. A good location is the following directory: **SAS-config-dir\Lev1\CustomAppData\FMCustomizedWorkflow**.

### Set the JVM Options

To make the resource file available, add the following option to the JVM options for SAServer3 (the managed server to which SAS Financial Management is deployed).

```
-Dsas.workflow.customizations="file:///path-to-resource-file"
```

Here is an example:

```
-Dsas.workflow.customizations="file:///C:/SAS/Config/Lev1/CustomAppData/
FMCustomizedWorkflow/WorkflowCustomizations.xml"
```

*Note:* Line break added for readability only. For information about configuring your Web application server, go to <http://support.sas.com/resources/thirdpartysupport/v93/index.html>.

The option applies when you restart the managed server.

*Note:* You do not need to restart the managed server when you make updates to the resource file.

## Data Validation Example

### About the Data Validation Example

Here is an example of cell-based data validation that uses a stored process, an execute type of **pre**, and a SUBMIT action. At run time, when a user submits a form in the specified form set, the stored process is automatically triggered. It validates a cell value in the form. If the value is greater than 0, the SUBMIT succeeds. Otherwise, the SUBMIT fails.

The example applies only to financial models. It makes the following assumptions:

- A form set with ID **123** has been created.
- A form template with a result model (called **tst\_model**) has been saved. It includes the dimensions shown in the Dimension column of [Table 4.2 on page 68](#).
- The form cell whose value is to be validated is defined by the crossing that is exemplified by the codes in the Member Code column of the following table.

The dimensionCodes and memberCodes arrays in the example contain the values from the first and second columns, respectively, of the following table. You do not need to include all the values in the table in the two arrays, but the values of the two arrays must match. During the query, any missing dimension code-member code pairs are filled with default values from the dimensions that are defined for the results model and the default read member that is defined in the hierarchy for each dimension.

**Table 4.2** Example Dimensions and Member Codes

Dimension	Member Code
ACCOUNT_FM	6232
ANALYSIS_FM	BUDGET
Cost Center	Total
CURRENCY	EUR
fm_INTORG_CODE	WW_SA
TIME_FM	012002
fm_INTORG_CODE_TRADER	EXT
SOURCE	BaseForm
PRODUCT_FM	Jackets

The actual query is carried out in the following code:

```
model.callDoubleMethod("getCellValue", "tst_model", dimensionCodes,
    memberCodes, value);
```



Depending on the return value, the program sets the FM\_SP\_RESULT and FM\_SP\_MESSAGE environment variables. If the return value is less than or equal to 0, the program sets FM\_SP\_RESULT to **INVALID** and sets FM\_SP\_MESSAGE to a text message. Otherwise, the program sets FM\_SP\_RESULT to **VALID**.

This example uses methods from the SAS Financial Management Java API. Most of the classes in this API apply only to financial planning and reporting. For details, see [Chapter 3, “The SAS Financial Management Java API,” on page 11](#).

## Code for the Example

This SAS program retrieves the data from the cell and validates the data.

*Note:* If you are declaring a Javaobj, the picklist option is required in the DATA step so that the Javaobj can find the necessary JAR files.

### Example Code 4.2 Stored Process for Workflow Customization

```
data _null_ /picklist='finance/finance.txt';
  put 'This is a data entry validation test';

  /* Read and echo environment variables passed in from the middle tier */
  /* form ID */
  length formId $20;
  formId = symgetc("fm_sp_form_id");
  put formId=;

  /* security key */
  length secKey $200;
  secKey = symgetc("fm_sp_seckey");
  put secKey=;

  /* action on the form */
  length action $20;
  action = symgetc("fm_sp_action");
  put action=;

  /* user ID */
  length userId $20;
  userId = symgetc("fm_sp_user_id");
  put userId=;

  /* user name */
  length userName $60;
  userName = symgetc("fm_sp_user_name");
  put userName=;

  /* Instantiate the Form class */
  dcl javaobj form("com/sas/solutions/finance/api/Form",formId, trim(secKey));
  form.ExceptionDescribe(1);

  /* Call methods of the Form class and echo the results */
  /* Get the target member code */
  length targetMemberCode $50;
  form.callStringMethod("getTargetMemberCode", targetMemberCode);
  put targetMemberCode=;
```

```

/* Get the target dimension code */
length targetDimensionCode $50;
form.callStringMethod("getTargetDimensionCode", targetDimensionCode);
put targetDimensionCode=;

length cFormInfo $20000;
form.callStringMethod("getInfo",cFormInfo);
put cFormInfo=;

length authors $ 200;
form.callStringMethod("getAuthors", " ", authors);
put authors=;

length admins $30000;
form.callStringMethod("getPlanningAdministrators", " ", admins);
put admins=;

/* Instantiate the Model class */
dcl javaobj model("com/sas/solutions/finance/api/Model", trim(secKey));

/* Set up two arrays, dimensionCodes and memberCodes */
array dimensionCodes[9] $50
(
    "",
    "ANALYSIS_FM",
    "ACCOUNT_FM",
    "Cost Center",
    "CURRENCY",
    "TIME_FM",
    "fm_INTORG_CODE_TRADER",
    "SOURCE",
    "PRODUCT_FM"
);

/* Set target dimension code */
dimensionCodes[1] = targetDimensionCode;

array memberCodes[9] $30
(
    "",
    "BUDGET",
    "6232",
    "Total",
    "EUR",
    "012002",
    "EXT",
    "BaseForm",
    "Jackets"
);

/* Set target member code */
memberCodes[1] = targetMemberCode;

/* Call getCellValue method */
length value 8;
model.callDoubleMethod("getCellValue", "tst_model", dimensionCodes,
    memberCodes, value);
put value=;

```

```

/* Test for value <= 0 and set environment variables accordingly */
if value <= 0 then do;
    call symput("fm_sp_result", "INVALID");
    call symput("fm_sp_message",
        "Account 6232 of JAN2002 should be greater than 0.");
end;
else do;
    call symput("fm_sp_result", "VALID");
end;
form.delete();
model.delete();
run;

```

## Registering the Stored Process

Register the stored process in SAS Management Console. For the example code, neither **Stream** nor **Package** is selected for the results, because the only output is to the log file. In other cases, you might want the stored process to generate streaming or package output.

For more information about registering a stored process, see the *SAS Stored Processes: Developer's Guide* and the online Help for SAS Management Console.

## Updating the Resource File

The resource file (`SAS-config-dir\Levl\CustomAppData\Workflow\WorkflowCustomizations.xml`) might have an entry as follows:

```

<SASWorkflowCustomizations>
  <Application name="SAS Financial Management">
    <Object type="FormSet" name="123">
      <Action type="SUBMIT">
        <Execute type="pre" storedProcessFullPath=
          "/Products/SAS Financial Management/Customized workflow/validation"/>
      </Action>
    </Object>
  </Application>
</SASWorkflowCustomizations>

```

In this case, the execute type is set to “pre”, which means that the stored process is executed before the workflow policy file.



## Chapter 5

# Creating a Custom Cell Action

---

<b>Overview</b> .....	<b>73</b>
<b>Write the Stored Process</b> .....	<b>74</b>
About the Stored Process .....	74
Parameters That You Can Expect .....	74
<b>Register the Stored Process</b> .....	<b>77</b>
<b>Update the Resource File</b> .....	<b>79</b>
Define the Custom Action .....	79
Set the JVM Option .....	80
<b>Select the Action</b> .....	<b>80</b>

---

## Overview

This chapter explains how to create a custom cell action for use with a SAS Financial Management table in Microsoft Excel.

When a user selects a data cell and clicks the right mouse button, the **Contributing Data** action is available by default. This action enables the user to view the data records that make up the selected cell.

You can add your own custom actions that invoke a stored process that displays its output in a browser window. For example, you might create a custom action that displays the transactions that make up the selected cell. Or you might create a custom action to reconcile adjustments in consensus forecasting.

Follow these steps to create a custom action:

1. Write a stored process to run when the action is invoked.  
See [“Write the Stored Process” on page 74](#).
2. In SAS Management Console, define the stored process metadata.  
See [“Register the Stored Process” on page 77](#).
3. Define the custom action in a resource file.

For the first custom action, you must create this file and set a JVM option that points to the resource file.

See [“Update the Resource File” on page 79](#).

4. The new action is available from a read-only table in Microsoft Excel. When a user right-clicks a cell and selects **Tools**, the new action appears as a selection.

See “[Select the Action](#)” on page 80.

---

## Write the Stored Process

### About the Stored Process

Your stored process will most likely use the SAS Financial Management Java API. For information about the classes and methods that make up that API, as well as information about declaring a Javaobj object and authenticating the user, see “[Authenticating the User](#)” on page 13.

Save the stored process code on the data tier, in a location such as the `SAS-config-dir\Levl\SASApp\SASEnvironment\FinancialManagement\SASCode\UserDefined` directory. Create the `UserDefined` directory if it does not already exist.

### Parameters That You Can Expect

At run time, when a user selects a custom action, a URL is built to call the associated stored process. The URL includes the following parameters, which are available to the stored process:

Parameter Name	Value
<code>_model</code>	The model ID for this table
<code>_modelCode</code>	The model code for this table
<code>__dimension-ID</code>	The member ID for this dimension, for the selected crossing

The parameter names are available in the `_APSLIST`. For example:

```
_APSLIST=__19, __8,_archive_path,_model,_metaperson, _metauser, ...
```

Dimension IDs and member IDs are represented by parameters beginning with two underscores (`__`). The parameter name following the underscores is the dimension ID, and the parameter value is the member ID for the selected crossing. The simple example below begins by scanning the list for variables beginning with two underscores (such as `__19`) and extracting the dimension IDs and member IDs.

#### Example Code 5.1 Example Stored Process for Custom Cell Action

```
/*+-----
| Copyright (c) 2011 by SAS Institute Inc., Cary, NC, USA.
| All rights reserved.
| Name: viewtrans.sas
| Purpose: show StageFM transactions that make up a cell value
+-----+*/
```

```

*ProcessBody;

ods path(prepend) sashelp.sasweb2(read);
%rptinit(style=sasweb2);

options mprint;

/*assign library (modify path as necessary) */
libname StageFM 'C:\SAS\Config\Levl\SASApp\Data\FinancialManagement\StageFM';

%let modelCode=&_modelCode;

%global account intorg analysis time currency;

* extract crossing values from the parameter list;
%model
%getModelHierarchies('FM','Work','HierOut',"&modelCode.", environment='default')

data DimType;
set work.HierOut;
  If strip(dimension_type_cd) IN ("ACCOUNT", "INTORG", "ANALYSIS", "TIME", "CURRENCY")
  then call symputx(dimension_type_cd, dimension_cd);
run;

data _null_ /picklist='finance/finance.txt';
length parameter $32;
length value $1000;
length dimID dim member $200;
  dcl javaobj oMetadata("com/sas/solutions/finance/api/Metadata");
  oMetadata.ExceptionDescribe(1);
  oMetadata.callVoidMethod("setEnvironment", "default");
  call METADATA_PASSID("oMetadata", "");

  * get the list of filters ;
  do until(parameter = '');
    i+1;
    parameter = scan("&_APSLIST", i, ",");
    if parameter ne '' then do;
      value = symget(parameter);
      if substr(parameter,1,2)='__' then do;
        dimID=substr(parameter,3);
        *if dimID ne 'FREQ' then do;
          put dimid=;
          oMetadata.callStringMethod("getDimensionCode", trim(dimID),dim);
          oMetadata.callStringMethod("getMemberCode", trim(dimID),trim(value), member);
          /* set dimension values, such as ACCOUNT=10020 */
          call symputx(dim, member);
          put dim=;
          put member=;
        *end;
      end;
    end;
  end;
  oMetadata.delete();
run;

```

```

data StageFM.queryParameters;
length DIMENSION_TYPE_CD MEMBER_CD $32;
  DIMENSION_TYPE_CD = "ACCOUNT"; MEMBER_CD = symget("&ACCOUNT."); INCLUDE_MEMBER=1;
  INCLUDE_LEAVES=4; INCLUDE_ROLLUPS=0; output;
  DIMENSION_TYPE_CD = "TIME"; MEMBER_CD = symget("&TIME."); INCLUDE_MEMBER=1;
  INCLUDE_LEAVES=2; INCLUDE_ROLLUPS=0; output;
  DIMENSION_TYPE_CD = "CURRENCY"; MEMBER_CD = symget("&CURRENCY."); INCLUDE_MEMBER=1;
  INCLUDE_LEAVES=0; INCLUDE_ROLLUPS=0; output;
  DIMENSION_TYPE_CD = "ANALYSIS"; MEMBER_CD = symget("&ANALYSIS."); INCLUDE_MEMBER=1;
  INCLUDE_LEAVES=0; INCLUDE_ROLLUPS=0; output;
  DIMENSION_TYPE_CD = "INTORG"; MEMBER_CD = symget("&INTORG."); INCLUDE_MEMBER=1;
  INCLUDE_LEAVES=4; INCLUDE_ROLLUPS=0; output;
run;

%fmquery(modelCode="&modelCode.", localSasLibName='Work', sasLibName='StageFM',
queryDataSetName='queryParameters', resultDataSetName='NONMDXResultDataSetName',
environment='default')

/*Store the records in NONMDXResultDataSetName into macro variables*/
data _null_;
  set NONMDXResultDataSetName nobs=end;
  account_code=tranwrd(account_code, ".vc", "");
  intorg_code=tranwrd(intorg_code, ".vc", "");
  string=strip(account_code)||"+"||strip(time_code)||"+"||strip(currency_code)||
    "+"||strip(analysis_code)||"+"||strip(intorg_code);
  call symputx("dim"||strip(put(_n_,8.)), string);
  call symputx("end", end);
run;

%global sqlobs;

%macro subset_transaction_table;
  proc sql;

    select a.gl_account_id, a.initiating_internal_org_id, a.analysis_id,
      a.affected_time_period_id, a.currency_cd, a.transaction_amt
    from StageFM.gl_transaction_sum a
    where
      %do i=1 %to &end.;
        (gl_account_id = "%scan(&&dim&i.,1, "+")" and
          affected_time_period_id= "%scan(&&dim&i.,2, "+")" and
          currency_cd = "%scan(&&dim&i.,3, "+")" and
          analysis_id="%scan(&&dim&i.,4, "+")" and
          initiating_internal_org_id = "%scan(&&dim&i.,5, "+")")
        %if &i. ne &end %then %do;
          or
        %end;
        %else %do;
          ;
        %end;
      %end;

    ;
  quit;
%mend;

```



```

%subset_transaction_table

data _null_;
if (symget("SQLOBS") = 0) then do;
  file print;
  put "NOTE: No rows were found";
  value=symget("&ACCOUNT.");
  put "ACCOUNT= " value;
  value=SYMGET("&INTORG.");
  put "ORG= " value;
  value=symget("&ANALYSIS.");
  put "ANALYSIS= " value;
  value=SYMGET("&TIME.");
  put "TIME= " value;
  value=symget("&CURRENCY.");
  put "CURRENCY= " value;
end;
run;

title;
footnote;
proc printto;
quit;

ods _all_ close;
ods listing;
%stpend;

```

*Note:* Line breaks inserted for readability.

---

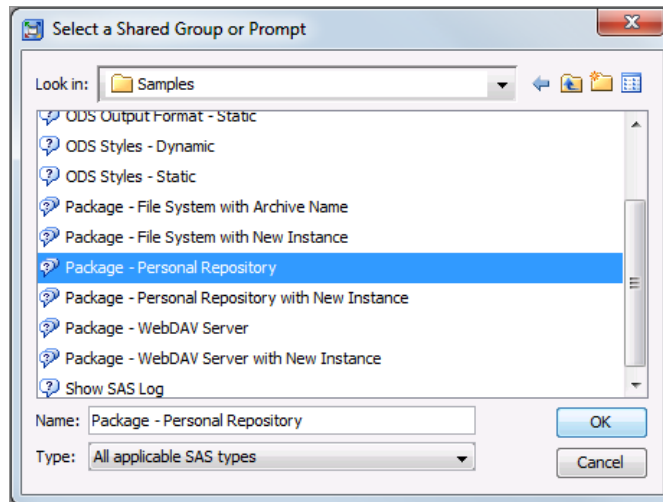
## Register the Stored Process

Define the stored process with package output to the personal repository, as follows:

1. Log on to SAS Management Console as an administrator.
2. On the **Folders** tab, right-click a shared folder and select **New Stored Process**. One possible location is the **/Products/SAS Financial Management/Custom Cell Actions** folder.

Create the **Custom Cell Actions** folder if it does not already exist.

3. On the Execution page of the wizard, select the stored process server and define the name and source for the stored process. Select the **Package** check box.
4. On the Parameters page, define any input parameters that are required by the stored process.
5. For the results options, select output to the personal repository, as follows:
  - a. Click **Add Shared**.
  - b. In the Select a Shared Group or Prompt dialog box, navigate to **SAS Folders \Products\Intelligence Platform\Samples**. Select **Package - Personal Repository**.

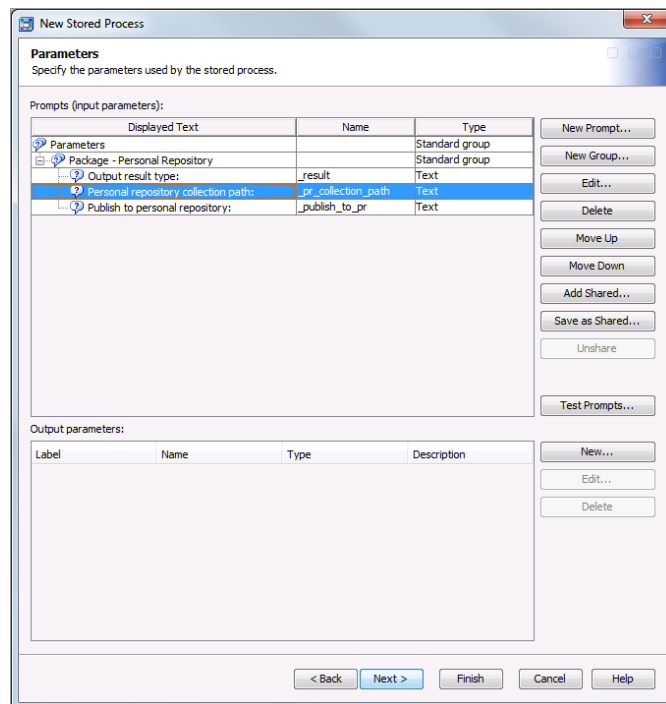
**CAUTION:**

Do not select **Package - Personal Repository with New Instance**.

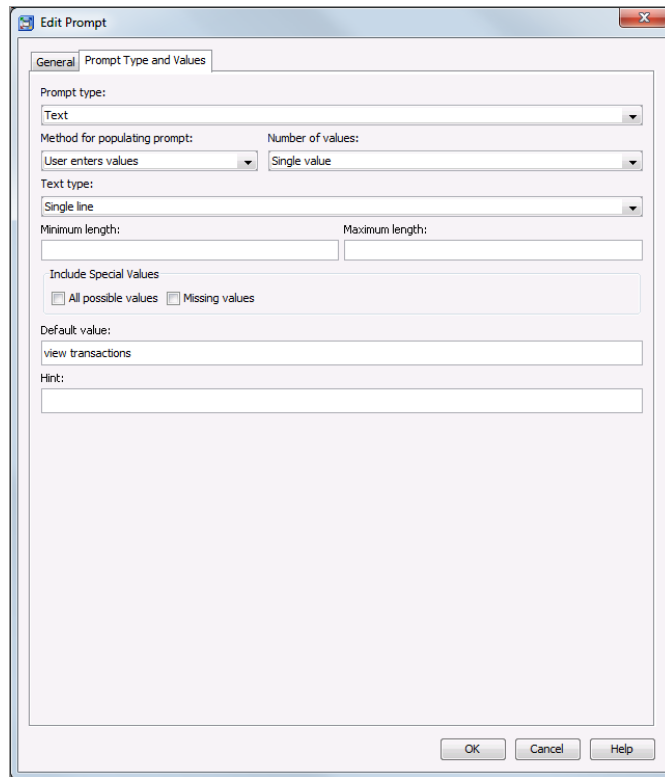
- c. Click **OK**.
- d. In the Add Package dialog box, click **OK**.
6. In the parameter list, select **Package - Personal Repository** and click **Unshare** to unshare the prompts so that you can modify them.

SAS Management Console displays a warning message and asks whether you want to continue. Click **Yes**.

7. Expand the options for the **Package - Personal Repository** prompt.



8. Select the **Personal Repository collection path** prompt and click **Edit**.
9. On the **Prompt Type and Value** tab, enter a name (such as **viewtransactions**) in the **Default value** text box.



10. Keep the defaults for the other results prompts, and save the stored process definition.
11. Make sure that users have ReadMetadata and WriteMetadata access to the stored process.

For more information about registering a stored process, see the online Help in SAS Management Console. See also the *SAS Stored Processes: Developer's Guide*.

---

## Update the Resource File

### Define the Custom Action

Custom actions are defined in a resource file that is stored on the middle tier, where the Web application server resides. A good location is a directory such as ***SAS-config-dir\Lev1\CustomAppData\FMCustomActions***. Create the ***FMCustomActions*** directory if it does not already exist.

The resource file is an XML file with the following contents:

```
<?xml version="1.0"?>
<customActions>
  <action name="action-name" onCell="true|false" onRollups="true|false"
    onLabels="true|false" onReadTable="true|false" onWriteTable="true|false">
    <description>description of this stored process</description>
    <url>URL to fallback page</url>
    <path>path to stored process metadata definition</path>
  </action>
</customActions>
```

The *action-name* is the name of the stored process, as defined in the metadata repository. In Microsoft Excel, it appears as the custom action.

The *fallback page* is the page to be displayed if the custom action fails for some reason. SAS Financial Management expects this file (with a name of `main.html`) to be available from the URL that you define in the resource file. In the example below, the fallback page would be `http://www.mycompany.com/CustomActions/Error/main.html`. Each custom action can have its own page (with its own URL), or you can specify the same URL for multiple actions.

The *path* is the path to the stored process definition in the metadata repository. Do not include a slash (/) before **Products**, and do not include the name of the stored process.

Here is an example:

```
<?xml version="1.0"?>
<customActions>
  <action name="View transactions" onCell="true" onRollups="true"
    onLabels="false" onReadTable="true" onWriteTable="false">
    <description>View transactions</description>
    <url>www.mycompany.com/CustomActions/Error</url>
    <path>Products/SAS Financial Management/Custom Cell Actions</path>
  </action>
</customActions>
```

## Set the JVM Option

If you have not already done so, tell SAS Financial Management where to find the resource file. Add the following option to the JVM options for the managed server to which SAS Financial Management is deployed (by default, `SASServer3`):

```
-Dsas.customActions.customizations=file:///path-to-resource-file
```

Here is an example:

```
-Dsas.customActions.customizations=file:///C:/SAS/Config/Lev1/CustomAppData/
FMCustomActions/CustomActions.xml
```

*Note:* Line break added for readability only. For information about configuring your Web application server, go to <http://support.sas.com/resources/thirdpartysupport/v93/index.html>.

The option applies when you restart the managed servers for SAS Financial Management and ODCS (typically, `SASServer3`, `SASServer4`, and `SASServer5`).

*Note:* If you update the resource file, you must also restart the managed servers.

---

## Select the Action

In Microsoft Excel, right-click a cell in a read-only table and select **Tools** to see the new action.

## Chapter 6

# Customizing the Form Status Dashboard

---

<b>The Form Status Dashboard</b> .....	<b>81</b>
Overview .....	81
The Dashboard Portlet .....	81
SAS BI Dashboard Viewer .....	82
Requirements for the Form Status Dashboard .....	82
<b>Indicators in the Form Status Dashboard</b> .....	<b>82</b>
Overview .....	82
Status .....	82
Approaching Deadline .....	83
<b>Customizing the Form Status Dashboard</b> .....	<b>83</b>
Overview .....	83
Modifying the Dashboard Design .....	84
Customizing the Stored Process .....	84

---

## The Form Status Dashboard



### Overview

Users can view a graphical display of form status in a SAS BI Dashboard portlet in the SAS Information Delivery Portal or in the SAS BI Dashboard Viewer.

The display is filtered to contain information only about forms that are available to the user. It contains links that take the user to the Forms workspace.

### The Dashboard Portlet

To display the Form Status dashboard in the SAS Information Delivery Portal, add a SAS BI Dashboard portlet to a portal page.

- To modify the display, click the **Edit Content** button  in the portlet's toolbar.
- To view the indicators in the SAS BI Dashboard viewer instead of in a portlet, click  in the BI Dashboard toolbar.

Click an indicator to open the Forms workspace of SAS Financial Management, with that filter applied. (This might trigger an additional logon.) For example, if you click the **Overdue** section of the **Approaching deadline** indicator, the display is filtered to show only forms that are overdue.

## SAS BI Dashboard Viewer

The SAS BI Dashboard Viewer is also available in the SAS BI Dashboard application. To open the application, enter the following Web address in your Web browser:

`http://server:port/SASBIDashboard`

- *server* is the host name of the Web application server.
- *port* is the port number of the managed server to which SAS BI Dashboard is deployed (typically, SASServer1).

## Requirements for the Form Status Dashboard

To view the Form Status dashboard, users must belong to either the BI Dashboard Users group or the BI Dashboard Administrators group and must have at least one of the following capabilities: Form Administration, Submit Financial Forms, Approve Financial Forms, or Approve and Submit Operational Forms.

Users must also have ReadMetadata permission for the `/Products/SAS Financial Management/5.3 Standard Reports` folder and ReadMetadata permission for the `/Products/SAS Financial Management/Dashboards/Form status` folder.

*Note:* Without ReadMetadata permission on a folder, users cannot navigate to items beneath that folder. For more information, see “Best Practices for Managing SAS Folders” in the *SAS Intelligence Platform: System Administration Guide*, available at <http://support.sas.com/93administration>.

---

# Indicators in the Form Status Dashboard

## Overview

By default, the Form Status dashboard contains the following indicators:

- **Status – all forms:** forms that are not yet started, in progress, or completed
- **Approaching deadline – all forms:** forms that are overdue or approaching deadline

Two additional indicators are available, displaying the same information by form set.

## Status

This indicator displays a count of forms that are not yet started, in progress, or completed. The status groups map to status in the Forms workspace as follows:

Bottom-up forms:

- **Not yet started** includes forms with a status of READY, HOLDING, or any variant of UNEDITED.

UNEDITED includes the following variants:

- `UNEDITED_READY_TO_SUBMIT`. This status can apply to a form that has no child forms, or to a parent form with child forms that have been submitted.

- **UNEDITED\_AWAIT\_CHILDREN.** This status can apply to a parent form with child forms that have not yet been submitted. Therefore, the parent form is not ready to be submitted.
- **Completed** includes forms with a status of **APPROVED**.
- **In progress** includes forms with a status of **EDITED**, **SUBMITTED**, **CHECKED OUT**, **REJECTED**, or **PARTIALLY APPROVED**.

Top-down forms:

- **Not yet started** includes forms with a status of **HOLDING** or **UNEDITED**.

*Note:* Forms with a status of **HOLDING** are available only to form administrators. For other users, these forms are not included in the dashboard or the Forms workspace.

- **Completed** includes forms with a status of **PUSHED** or **COMPLETED**.
- **In progress** includes forms with a status of **EDITED**.

## Approaching Deadline

This indicator displays a count of forms that are approaching deadline or past due. The status groups map to status in the Forms workspace as follows:

- **Approaching deadline** includes forms without **Completed** status that are within a specified number of days of the form's deadline. The range is customizable. The default is 5 days.
- **Overdue** includes forms without **Completed** status whose overdue flag has been set by the server.
- **Other** includes forms that are not overdue and not approaching deadline.

---

# Customizing the Form Status Dashboard

## Overview

The form status dashboard is based on the following components:

- dashboard content, located in the **/Products/SAS Financial Management/Dashboards/Form status** folder of the SAS content server. It includes the **Form process status** dashboard (Form process status.dcx) and the following indicators:
  - **Status – all forms** (Form status indicator - all forms.idx)
  - **Status – by form set** (Form status indicator - by form set.idx)
  - **Approaching deadline – all forms** (Forms approaching deadline indicator - all forms.idx)
  - **Approaching deadline – by form set** (Forms approaching deadline indicator - by form set.idx)

This folder also contains four indicator data files (IMX files) that provide data to the four indicators.

- a stored process (**/Products/SAS Financial Management/5.3 Standard Reports/Form Status**). Refreshing the dashboard publishes a SAS package to an archive that contains SAS data sets. Each data set provides data to the corresponding IMX files. The archive is specific to the user who is logged on to the portal.
- a Java API class (**com/sas/solutions/finance/api/FormStatusModel**) with methods that are called by the stored process.

You can customize the dashboard and the underlying code in these ways:

- You can modify the dashboard design.
- You can modify the underlying stored process (**formstat.sas**).

## Modifying the Dashboard Design

If you belong to the BI Dashboard Administrators group (or have the BI Dashboard: Administration role), it is possible to modify the dashboard design. For example, you can change the dashboard layout, add or remove indicators, or change the graph style for an indicator.

*Note:* Because the dashboard data is generated by a stored process, some dashboard customizations are not appropriate. In particular, do not add links to an indicator. The Form Status stored process creates links to the Forms workspace for the four indicators that are described above.

For more information, see the *SAS BI Dashboard 4.31: User's Guide* at <http://support.sas.com/documentation/onlinedoc/bidashboard/index.html>

## Customizing the Stored Process

The stored process that generates the indicator data is defined in the SAS content folders (**/Products/SAS Financial Management/5.3 Standard Reports/Form Status**). Refreshing the dashboard publishes a SAS package to an archive that contains SAS data sets. Each data set provides data to the corresponding IMX files.

The stored process code, **formstat.sas**, is located in the **!SASHome\SASFoundation\9.3\finance\sasstp** directory (Windows) or the **!SASHome\SASFoundation\9.3\sasstp\finance** directory (UNIX).

By default, the filter selects forms with a deadline within the next 5 days. To change this range:

1. Open the **formstat.sas** file for editing.
2. Find this line:

```
j.callStaticIntMethod("getFilterApproachingDeadlineParamValue",
    filterApproachDlineParamValue);
```

3. Comment out the line and add code that sets a new parameter value. This example sets the parameter value to 12. That is, the filter selects forms that have a deadline within the next 12 days.

```
/*
j.callStaticIntMethod("getFilterApproachingDeadlineParamValue",
    filterApproachDlineParamValue);
*/
filterApproachDlineParamValue = 12;
```



4. Save the file.

No other customizations are supported. Be aware that the underlying code, as well as this stored process, might change in a subsequent release of SAS Financial Management. If you do make additional changes, we recommend that you use a copy of formstat.sas so that your changes are not lost in the event of an upgrade.



## Chapter 7

# The SAS Financial Management Add-In API for Microsoft Excel

---

<b>Overview of Working with the SAS Financial Management Add-In API for Microsoft Excel</b>	<b>88</b>
<b>Setup for Using the API</b>	<b>88</b>
<b>General Usage Information</b>	<b>89</b>
Declaring the FMAddIn Object	89
Working with Objects	89
Handling Events	90
Activating the Log	92
<b>Summary of Classes</b>	<b>92</b>
<b>The FMAddIn Class</b>	<b>93</b>
<b>The FMCollections Class</b>	<b>97</b>
<b>The FMCrossing Class</b>	<b>99</b>
<b>The FMCrossingsCollection Class</b>	<b>100</b>
<b>The FMCube Class</b>	<b>100</b>
Overview	100
Writing Values to a Cube	102
Property Summary	103
Method Summary	104
<b>The FMCubesCollection Class</b>	<b>105</b>
<b>The FMHierarchiesCollection Class</b>	<b>105</b>
<b>The FMHierarchy Class</b>	<b>105</b>
<b>The FMMember Class</b>	<b>110</b>
<b>The FMMembersCollection Class</b>	<b>113</b>
<b>The FMTable Class</b>	<b>113</b>
<b>The FMTablesCollection Class</b>	<b>121</b>
<b>The FMUser Class</b>	<b>121</b>
<b>Including CDA Functions</b>	<b>122</b>

---

## Overview of Working with the SAS Financial Management Add-In API for Microsoft Excel

With the SAS Financial Management Add-In for Microsoft Excel and the SAS Financial Management Add-In API for Microsoft Excel, you can use Visual Basic for Applications (VBA) to write macros that interact with SAS Financial Management objects. For example, you might perform some of the following tasks:

- Launch a SAS Financial Management report in batch mode, automatically log on to the SAS Financial Management server, and print the report with updated numbers.
- Retrieve SAS Financial Management data and metadata.
- Use the FMMember selection dialog box in a cell data access (CDA) report.
- Execute code that is based on events from the SAS Financial Management objects.
- Apply custom formatting to SAS Financial Management tables.

---

## Setup for Using the API

The API requires a reference to the SASSESEExcelAddin.tlb type library. In Microsoft Excel, follow these steps to add the reference:

1. Click the **Developer** tab.
2. Click **Visual Basic**.
3. From the **Tools** menu of the Visual Basic Editor, select **References**.
4. From the list of available references, select **SASSESEExcelAddIn**.

If **SASSESEExcelAddIn** is not in the list, click **Browse** to select the file and add it to the list. The file is located in the

**SASFinancialManagementAddinforMicrosoftExcel\5.3** directory where you installed the add-in.

*Note:* If you had an earlier version of the SAS Financial Management Add-In for Microsoft Excel, deselect the check box for **SASSESEExcelAddin** on the References page, click **OK**, and exit Excel. Then re-open Excel and add the new TLB file as described above.

5. Click **OK**.

In the Microsoft Excel options, verify that your default file save format is one that supports macros:

1. In Microsoft Excel 2010, open **File** ⇒ **Options**. From the options menu on the left, click **Save**.
2. From the **Save files in this format** drop-down list, select one of the following:
  - **Excel Binary Workbook (\*.xlsb)**
  - **Excel Macro-Enabled Workbook (\*.xlsm)**
3. Click **OK**.

---

## General Usage Information

### Declaring the FMAddIn Object

In the **Declarations** section of the Workbook module, declare the FMAddIn object and other SAS Financial Management objects in code that resembles the following:

```
Public addin As FMAddIn
Public table As FMTable
Public cube As FMCube
Public user As FMUser
```

To use the events framework, the declarations for FMAddin and FMTable should resemble the following code:

```
Public WithEvents addin As FMAddIn
Public WithEvents table As FMTable
```

For more information about the events framework, see [“Handling Events” on page 90](#).

### Working with Objects

#### The FMAddin Object

To get a reference to the FMAddIn object, use code that resembles the following:

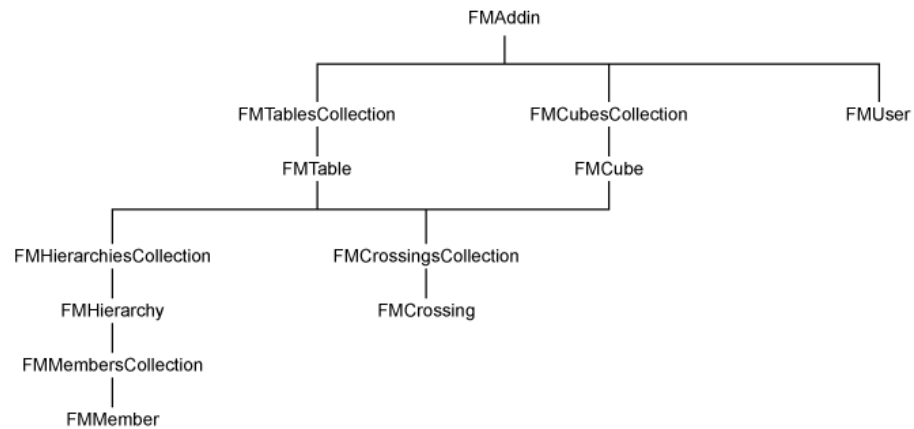
```
Dim conn As Connect
...
Set conn = Application.COMAddIns.Item("SASSESEExcelAddIn.Connect").Object
Set addin = conn.FMAddIn
```

For the remainder of this chapter, the code examples assume that you already have a reference (called **addin**) to the FMAddIn object. (Your code should contain only one instance of the FMAddIn object.)

From the FMAddIn object, you can get a reference to the FMTablesCollection object or to an FMCubesCollection object. The tables collection represents all tables in the workbook. Each FMTable object in the collection represents a data entry or read-only table in the current workbook. The cubes collection represents all virtual cubes (results models) on the server. Each FMCube object represents a virtual cube.

*Note:* We recommend using **Option Explicit** in your code. This option requires all variables to be explicitly declared.

This diagram shows classes in the API. It indicates which classes contain references to other classes. (It is not intended to imply any inheritance from one class to another.)

**Figure 7.1** Classes in the SAS Financial Management Add-In API for Microsoft Excel

### Objects in a Collection

To get a reference to an object in a collection, you can specify an index into the collection. For example, `addin.Tables(0)` references the first table in an `FMTablesCollection` object.

You can also name an object in the collection. To get a reference to an object in the `FMCubesCollection`, `FMHierarchiesCollection`, or `FMMembersCollection`, you specify the code for the cube, hierarchy, or member. For example:

```
Dim cube As FMCube
Set cube = addin.cubes("Default_Model")
```

Another approach is to iterate through the collection. This code iterates over a collection of server hierarchies in a cube:

```
For Each hierarchy In cube.ServerHierarchies
    ...
Next hierarchy
```

### Table Objects

To get a reference to a table, use the table name (tables do not have codes). For example:

```
Dim table As FMTable
Set table = addin.Tables("NewTable0")
```

In Excel, the location of a table is defined as a named range. When you add the first table, it is automatically named **NewTable0**. The next table is named **NewTable1**, and so on.

*Note:* A user might change the name of a table (in the table properties), but the new name is only for display purposes and cannot be used in the code. For more information, see the `getTableName` method of the `FMTable` class.

## Handling Events

### About Events

An event is an action that happens in Excel (for example, logging in or refreshing a table). Event handlers are called when the user performs the specified action.

For an event to be captured:

- The object that the event is associated with must be declared using the  **WithEvents**  clause; for example:
- There must be an existing reference to the object; for example:

```
Set table = addin.tables("NewTable0")
```

### **Write an Event Handler**

To write an event-handling procedure, use code similar to the following example, which is invoked when the user refreshes the worksheet:

```
Public Sub addin_AfterRefresh()  
    MsgBox "Refresh event trapped in VBA"  
End Sub
```

The name of the procedure is *object-name* + *\_* + *event-name*.

Be aware that an action can trigger multiple events. For example, if a user selects **View** ⇒ **Refresh**, the table refresh event is triggered, followed by the worksheet's refresh event. On the other hand, if a table object's Refresh method is called, or if the user performs an action that affects a single table, then only that table's refresh event is triggered.

Suppose that you want to resize the columns for a table each time the table is refreshed. To ensure that the table columns are always resized correctly, you need to add the resizing code to both the **table0\_AfterRefresh** event handler and the **addin\_AfterRefresh** event handler. The code might resemble the following:

#### **Example Code 7.1 Event Handler**

```
Public WithEvents table0 As FMTable  
Public WithEvents addin As FMAddin  
...  
' Event handler for table0  
Private Sub table0_AfterRefresh()  
    ' Temporarily disable screen updating  
    Application.ScreenUpdating = False  
  
    ' Resize columns to have a uniform width  
    startColumn = table0.Position(fmArea_Column, fmType_startColumn)  
    endColumn = table0.Position(fmArea_Column, fmType_endColumn)  
    For col = startColumn To endColumn  
        Columns(col).ColumnWidth = 20  
    Next col  
  
    ' Re-enable screen updating  
    Application.ScreenUpdating = True  
End Sub  
  
' addin object's AfterRefresh event handler  
Private Sub addin_AfterRefresh()  
    Application.ScreenUpdating = False  
  
    ' Check to be sure this table is in the active worksheet  
    If Range(table0.Name).Worksheet.Name = ActiveSheet.Name Then  
        ' Resize columns  
        startColumn = table0.Position(fmArea_Column, fmType_startColumn)  
        endColumn = table0.Position(fmArea_Column, fmType_endColumn)
```

```

        For col = startColumn To endColumn
            Columns(col).ColumnWidth = 20
        Next col
    End If

    Application.ScreenUpdating = True
End Sub

```

To handle a table refresh that occurs when the user selects **View** ⇒ **RefreshAll**, you would write similar code for the **addin\_AfterRefreshAll** event handler.

If you wanted to resize the columns of all tables to have a uniform width, then you would write an **addin\_AfterTableRefresh** event handler, which would be called for each table that was refreshed.

For more information about specific events, see the event summaries for the FMAddin class and the FMTable class.

## Activating the Log

If enabled, a log file records information about queries generated by the SAS Financial Management Add-In for Microsoft Excel. You can also write to the log using the `traceWrite` function of the FMAddin class.

By default, this log is disabled.

- For information about activating the log, see the SAS Usage Note at <http://support.sas.com/kb/46/178.html>.
- To prevent the log file from becoming too long, we recommend that you specify a `DebugLevel` no higher than 2 in the configuration file.

---

## Summary of Classes

The following table summarizes the classes that make up the API.

**Table 7.1** Summary of Classes

Class	Description
FMAddIn	The top-level class for manipulating the add-in.
FMCollections	Base class for other collections such as FMCrossingsCollection and FMTablesCollection. Its properties and methods are inherited by these subclasses.
FMCrossing	Provides access to the properties of a crossing in a table or cube.
FMCrossingsCollection	Represents a collection of crossings.
FMCube	Represents a virtual cube (results model).
FMCubesCollection	Represents a collection of cubes.



Class	Description
FMHierarchy	Represents a hierarchy.
FMHierarchiesCollection	Represents a collection of hierarchies.
FMMember	Represents a member of a hierarchy.
FMMembersCollection	Represents a collection of members.
FMTable	Represents a table.
FMTablesCollection	Represents a collection of tables.
FMUser	Represents the user who is currently logged on.

## The FMAddIn Class

The FMAddIn class is the top-level class in the API. From the FMAddIn object, you can get a reference to the tables in the workbook, the cubes that are on the server, and the current user.

**Table 7.2** FMAddIn Property Summary

Property	Description
Property <b>Cubes</b> As FMCubesCollection	A collection of cubes that are on the server (and that you have access to). Read-only.
Property <b>isLoggedIn</b> As Boolean	If <b>True</b> , the user is logged on. Read-only.
Property <b>MessageBoxEnabled</b> As Boolean	If <b>False</b> , pop-up messages are disabled from the SAS Financial Management Add-In. Typically, you would set this property to <b>False</b> when you are running in batch mode. The default is <b>True</b> . Read-write.
Property <b>MessageBoxResponseOK</b> As Boolean	The default response to any suppressed message boxes. This property applies only if MessageBox Enabled is set to <b>False</b> . A value of <b>True</b> sets the default response to <b>Yes</b> or <b>OK</b> . A value of <b>False</b> sets the default response to <b>No</b> or <b>Cancel</b> . The default is <b>True</b> . Read-write.
Property <b>Port</b> As Long	The port number of the middle-tier server on which SAS Financial Management is running. Read-only.
Property <b>ReadOnly</b> As Boolean	This property applies if the user is viewing a data-entry form. If <b>True</b> , the form cannot be edited.
Property <b>Secure</b> As Boolean	<b>True</b> if the middle-tier server is using the Secure Socket Layer (SSL) protocol. Otherwise, <b>False</b> . Read-only.

Property	Description
Property <b>Server</b> As String	The name of the middle-tier server on which SAS Financial Management is running. Read-only.
Property <b>Tables</b> As FMTablesCollection	A collection of tables. Read-only.
Property <b>Url</b> As String	The URL to the middle-tier server on which SAS Financial Management is running. Read-only.
Property <b>User</b> As FMUser	A FMUser object that represents the user who is currently logged on. Read-only.
Property <b>Version</b> As String	The name and version number of this software. Read-only.
Property <b>VersionDate</b> As String	The date of this version of the software. Read-only.
Property <b>VersionID</b> As String	The version number of this software. Read-only.

**Table 7.3** FMAddIn Class Method Summary

Function <b>enumString</b> (fmEnum As fmEnums, enumValue As Long) As String	<p>Returns the String equivalent of an enumerated constant—for example, the value returned from a write operation, the name of a role, or an area of the table.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <i>fmEnum</i>: the type of enumerated constant. This parameter can be one of the following: <b>fmBudgetMode</b>, <b>fmDisplayMode</b>, <b>fmRole</b>, <b>fmType</b>, <b>fmArea</b>, <b>fmSelection</b>, <b>fmCreditsDebitsDisplay</b>, or <b>fmWriteBackReturn</b>.</li> <li>• <i>enumValue</i>: the value to be converted into a string.</li> </ul> <p>Returns: a string that corresponds to <i>enumValue</i> for the specified type of constant.</p> <p>Many methods take enumerated constants as parameters or return them as return values. The Write method returns an enumerated constant (a numeric value). You can declare the variable that you are using for the return value as an enumerated constant and then access its string representation. This code fragment displays a message box for a write operation that failed, with the reason for the failure:</p> <pre>Dim rc As fmWriteBackReturn ... Set crossing = addin.Tables(0).crossing(4, 3) rc = crossing.Write(111) MsgBox "return from write: " &amp; _     addin.enumString(fmEnums_fmWriteBackReturn, rc)</pre>
Function <b>findTable</b> (sheetName As String, row As Long, column As Long) As FMTable	<p>Finds the table object that corresponds to the specified sheet and position.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <i>sheetname</i>: the name of a sheet in the workbook.</li> <li>• <i>row</i>, <i>column</i>: the position of a table element.</li> </ul> <p>Returns: an FMTable object.</p>

---

Function <b>getTableNames</b> (username As String) As String	<p>Returns the internal name of the table that corresponds to a name in the table properties. By default, the first table a user inserts is named <b>NewTable0</b>, the second table is <b>NewTable1</b>, and so on. The user might rename the table in the table properties. However, the new name is only a display name. The code requires the original name, which is available via the <code>getTableNames</code> function.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <i>username</i>: the table name in the table properties.</li> </ul> <p>Returns: the original table name.</p>
Function <b>Login</b> (environment As String, username As String, password As String) As Boolean	<p>Logs the user on to the middle tier.</p> <p>If the user is already logged on, this function returns <b>True</b> even if the parameter values are incorrect.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <i>environment</i>, <i>username</i>, and <i>password</i>: the environment, user name, and password for logging on to the middle tier. These parameters are the same values that you would use to log on from the <b>SAS Financial Management</b> menu in Excel. The environment value is site-specific. Environments are defined in the <code>EnvironmentFactory.xml</code> file. For more information, see <a href="#">“Specifying the SAS Financial Management Environment” on page 14</a>.</li> </ul> <p>We recommend generating an encoded or encrypted password that you can copy and paste into your code, rather than using a plain-text password. For more information, see the <i>SAS Intelligence Platform: Security Administration Guide</i>.</p> <p>Returns: <b>True</b> if the user is already logged on or if the login succeeds; otherwise, <b>False</b>.</p>
Function <b>Logout</b> () As Boolean	<p>Logs the user off the middle tier.</p> <p>Returns: <b>True</b> if the action succeeded; otherwise, <b>False</b>.</p>
Function <b>Refresh</b> () As Boolean	<p>Refreshes the selected worksheet. This action is similar to the <b>Refresh</b> action from the toolbar menu.</p> <p>Returns: <b>True</b> if the action succeeded; otherwise, <b>False</b>.</p>
Function <b>RefreshAll</b> () As Boolean	<p>Refreshes all open worksheets in the selected file. This action is similar to the <b>Refresh All</b> action from the toolbar menu.</p> <p>Returns: <b>True</b> if the action succeeded; otherwise, <b>False</b>.</p>
Function <b>traceWrite</b> (traceString As String) As Boolean	<p>Writes the contents of <i>traceString</i> to the log. This method is helpful in debugging your code.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <i>traceString</i>: the string to write.</li> </ul> <p>By default, the log is disabled. See <a href="#">“Activating the Log” on page 92</a>.</p>

---

**Table 7.4** FMAddin Event Summary

Event	Description
Event <b>AfterLogOff()</b>	Triggered after the user logs off from the middle tier. A logoff event occurs when there is a call to the Logoff method of the FMAddin object or when the user selects <b>Log Off</b> from the toolbar.
Event <b>AfterLogon()</b>	Triggered after the user has logged on. This event occurs when there is a call to the Login method of the FMAddin object, when the user selects <b>Log On</b> from the toolbar, or when the user opens an Excel report from the portal.
Event <b>AfterRefresh()</b>	Triggered after a refresh action—for example, if there is a call to <b>addin.Refresh()</b> or if the user selects <b>Refresh</b> from the toolbar.
Event <b>AfterRefreshAll()</b>	Triggered if there is a call to <b>addin.RefreshAll()</b> or if the user selects <b>Refresh All</b> from the toolbar.
Event <b>AfterTableRefresh</b> (table As FMTable)	<p>Triggered after a table has been refreshed. This event might occur if there is a call to the <b>Refresh</b> method of a table object, if the user selects <b>Refresh</b> or <b>Refresh All</b> from the toolbar, or if the user performs some other manual action, such as a pivot, that triggers a refresh.</p> <p>If the user refreshes a worksheet, the table refresh event and the addin refresh event are triggered, in that order.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><i>table</i>: an FMTable object that represents the table that was refreshed. If you refresh a worksheet that contains multiple tables, the AfterTableRefresh event is triggered multiple times, once for each table.</li> </ul> <p>This event handler displays a message that includes the name of the table that was refreshed:</p> <pre>Private Sub addin_AfterTableRefresh(ByVal table As FMTable)     txt = "Addin afterRefresh: " + table.Code     MsgBox txt End Sub</pre>
Event <b>BeforeLogOff()</b>	<p>Triggered when logoff has been requested but before the user logs off. This event handler returns a Boolean. If the return value is <b>True</b>, the logoff continues. If the return value is <b>False</b>, the logoff is canceled. Here is an example:</p> <pre>Private Function addin_BeforeLogOff() As Boolean     response = MsgBox("Do you really want to log off?", _         vbOKCancel, "SAS Financial Management")     If response = vbOK Then         addin_BeforeLogOff = True     Else         ' Cancel the logoff process         addin_BeforeLogOff = False     End If End Function</pre>

Event	Description
Event <b>BeforeTableRefresh</b> (table As FMTable)	<p>Triggered before a table is refreshed. You might use this event handler to disable screen updating while you are modifying the screen. In the AfterTableRefresh event handler, you could re-enable screen updating.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><i>table</i>: an FMTable object that represents the table that was refreshed. If you refresh a worksheet that contains multiple tables, the BeforeTableRefresh event is triggered multiple times, once for each table.</li> </ul>

## The FMCollections Class

The FMCollections class is the base class for several other collections: FMCrossingsCollection, FMCubesCollection, FMHierarchiesCollection, FMMembersCollection, and FMTablesCollection. Its properties and methods for manipulating a collection are inherited by these subclasses. Use one of the subclasses rather than invoking this class directly.

*Note:* Do not use **New** to instantiate one of these collections. If you need to create an empty FMMembersCollection object, use the GetNewMembersCollection method of the FMHierarchy class.

**Table 7.5** FMCollections Class Property Summary

Property	Description
Property <b>Count</b> As Long	The number of items in the collection. Read-only.

**Table 7.6** FMCollections Class Method Summary

Class	Description
Sub <b>Add</b> (item)	<p>Adds a single item to the collection.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><i>item</i>: the item to add.</li> </ul> <p>This example creates an FMMembersCollection object and adds two members of the <b>ACCOUNT.AccountType</b> hierarchy to the collection:</p> <pre>Dim hierarchy As FMHierarchy Dim excmems As FMMembersCollection Set hierarchy = addin.Tables(0).Hierarchies("ACCOUNT.AccountType") Set excmems = hierarchy.GetNewMembersCollection() Call excmems.Add(hierarchy.Members("StatisticalBalance")) Call excmems.Add(hierarchy.Members("Equity"))</pre>

Class	Description
Sub <b>AddAll</b> (item)	<p>Adds a collection of items to the collection.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><i>item</i>: a collection of items to add (for example, an FMMembersCollection object that represents a collection of members).</li> </ul> <p>This example creates a FMMembersCollection object and adds all the members of the <b>ACCOUNT.AccountType</b> hierarchy to the collection:</p> <pre>Dim hierarchy As FMHierarchy Dim mems As FMMembersCollection Set hierarchy = addin.Tables(0).Hierarchies("ACCOUNT.AccountType") Set mems = hierarchy.GetNewMembersCollection() Call mems.AddAll(hierarchy.Members)</pre>
Sub <b>Clear</b> ()	<p>Clears the collection.</p>
Function <b>Contains</b> (item) As Boolean	<p>Returns: <b>True</b> if the collection contains the specified item.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><i>item</i>: a single item (for example, an FMMember object if you are searching an FMMembersCollection instance).</li> </ul>
Function <b>IndexOf</b> (item) As Long	<p>Returns: the zero-based index (position) of the specified item in the collection, or <b>-1</b> if the item is not found.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><i>item</i>: an object of the collection type (for example, an FMMember object).</li> </ul>
Sub <b>Insert</b> (index As Long, item)	<p>Inserts <i>item</i> at the <i>index</i> position in the collection.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><i>index</i>: a (zero-based) index into the collection.</li> <li><i>item</i>: an object of the collection type.</li> </ul>
Sub <b>InsertAll</b> (index As Long, item)	<p>Inserts a collection at the <i>index</i> position in the collection.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><i>index</i>: a (zero-based) index into the collection.</li> <li><i>item</i>: an object that represents a collection (for example, an FMMembersCollection object).</li> </ul>
Sub <b>Remove</b> (item)	<p>Removes an object from a collection (if the object is found).</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><i>item</i>: an item in a collection.</li> </ul>
Sub <b>RemoveAt</b> (index As Long)	<p>Removes the item at the <i>index</i> position in the collection.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><i>index</i>: a (zero-based) index into the collection.</li> </ul>
Function <b>ToString</b> () As String	<p>Returns: a string that represents the concatenated codes of all the elements in the collection.</p>

## The FMCrossing Class

The FMCrossing class provides access to the properties of a crossing. In a table, a crossing is determined by its position in the table (row and column). In a cube, a crossing is determined by a two-dimensional String array of dimension codes and member codes. For examples, see the FMCube class ([“The FMCube Class” on page 100](#)) or the FMTable class ([“The FMTable Class” on page 113](#)).

**Table 7.7** FMCrossing Class Property Summary

Property	Description
Property <b>Code</b> As String	An identifier for this crossing. For table crossings, the code is a string that contains information about the row and column for the crossing. For crossings in a cube, the code is a concatenated string of model code, dimension codes, and member codes, such as the following:  DefaultModel_ACCOUNT_NETINCOME_TIME_JAN2003_ANALYSIS_BUDGET...  Read-only.
Property <b>Column</b> As Long	The column position of this crossing. Applies only to tables. Read-only.
Property <b>ColumnRelative</b> As Long	The column position of this crossing, relative to the leftmost column of the table ( $crossingColumn - firstColumn + 1$ ). Applies only to tables. Read-only.
Property <b>DimensionMembers</b> As String()	A two-dimensional array of strings that contain the dimensions and members that apply to this crossing, in the form ( <i>dimension, member</i> ). Read-only.
Property <b>Length</b> As Long	The number of dimensions in this crossing. Read-only.
Property <b>NewValue</b> As Double	For a cube or a table, the NewValue is the value that is written to the server for this crossing when the BatchWrite method is called. Read/write.
Property <b>Row</b> As Long	The row position of this crossing. Applies only to tables. Read-only.
Property <b>RowRelative</b> As Long	The row position of this crossing, relative to the topmost row of the table ( $crossingRow - firstRow + 1$ ). Applies only to tables. Read-only.
Property <b>ScaledValue</b> As Double	For tables, this property contains the value of the crossing divided by the current scale of the table. This value is similar to the value that is shown in the table.  For cubes, this property contains the value of the crossing. (It is identical to the Value property.)  Read-only.
Property <b>Value</b> As Double	The value of this crossing, before any table scaling is applied. Read-only.

Property	Description
Property <b>Writeable</b> As Boolean	If <b>True</b> , the crossing is writable. Read-only.  For crossings that are derived from a cube, this property does not honor member security.

**Table 7.8** *FMCrossing Class Method Summary*

Method	Description
Function <b>GetMember</b> (item As String) As FMMember	Returns the member of this crossing for the specified dimension code.  Parameters: <ul style="list-style-type: none"> <li><i>item</i>: a dimension code, such as <b>ACCOUNT</b> or <b>ORG</b>.</li> </ul>
Function <b>GetMemberCode</b> (item As String) As String	Returns the member code in this crossing for the specified dimension code.  Parameters: <ul style="list-style-type: none"> <li><i>item</i>: a dimension code.</li> </ul>
Function <b>Write</b> (value As Double) As fmWriteBackReturn	Writes <i>value</i> to this crossing.  For crossings that are derived from a cube, this method does not honor member security.  Returns: the status of the Write operation. (See <a href="#">Table 7.3 on page 94</a> .)

## The FMCrossingsCollection Class

The FMCrossingsCollection class represents a collection of crossings. This class is a subclass of FMCollections.

The following properties are inherited from FMCollections: Count.

The following methods are inherited from FMCollections: Add, AddAll, Clear, Contains, IndexOf, Insert, InsertAll, Remove, RemoveAt, ToString.

## The FMCube Class

### Overview

An FMCube object represents a virtual cube (results model). With the FMCube class, you can access metadata from the SAS Financial Management data mart. An instance of the FMCube class can be the entry point for metadata about results models, hierarchies, and members. With FMCube methods, you can also read and write facts.

The FMCube class works independently of read-only tables, data entry tables, and CDA expressions. As a result, your code is able to interact with metadata and with facts.

To perform a query for a cube:



1. Get a reference to the cube.
2. Get a reference to the cube's crossings collection, which is empty to begin with.
3. Get the crossings for a particular set of (*dimension code*, *member code*) values, and add them to the cube's crossings collection.

At this point, you have the metadata for the crossings, but you have no corresponding values.

4. Call the cube's ExecuteQuery method to get the values for each crossing in the collection.

The (*dimension code*, *member code*) values in the cube's crossings collection are the parameters for the query.

This example creates a set of query parameters, performs a query, and displays the results.

### Example Code 7.2 Query on a Cube

```
Public addin As FMAddIn
Dim crossing As FMCrossing
Dim crs As FMCrossing
Dim crossings As FMCrossingsCollection
Dim cube As FMCube
Dim member As FMMember

Public Sub testCube()
    Set Connection = _
        Application.COMAddIns.Item("SASSESEExcelAddIn.Connect").Object
    Set addin = Connection.FMAddIn

    If addin.IsLoggedIn = False Then
        MsgBox "Please log in..."
        Exit Sub
    End If

    ' Get reference to cube
    Set cube = addin.Cubes("Default_Model")

    ' Specify dimension, member pairs
    ' to be used as default parameters for query
    Dim dm() As String
    ReDim dm(cube.ServerHierarchies.Count - 1, 1)
    dm(0, 0) = "ACCOUNT"
    dm(0, 1) = "A6520"
    dm(1, 0) = "TIME"
    dm(1, 1) = "JAN2011"
    dm(2, 0) = "FREQUENCY"
    dm(2, 1) = "PA"
    dm(3, 0) = "ORG"
    dm(3, 1) = "BMRM"
    dm(4, 0) = "ANALYSIS"
    dm(4, 1) = "BUDGET"
    dm(5, 0) = "COUNTRY_D"
    dm(5, 1) = "WW.vc"
    dm(6, 0) = "TRADER"
    dm(6, 1) = "EXT"
```

```

dm(7, 0) = "PERIODS"
dm(7, 1) = "AP.vc"
dm(8, 0) = "PRODUCT"
dm(8, 1) = "B0815"
dm(9, 0) = "CURRENCY"
dm(9, 1) = "USD"
dm(10, 0) = "SOURCE"
dm(10, 1) = "EXT"

' Get reference to crossings collection for this cube
' (Collection is currently empty.)
Set crossings = cube.crossings

' Get reference to crossings for YR2001 in TIME dimension
' and add to crossings collection
For Each member In cube.Hierarchies("TIME").GetMembers("YR2001", True, False)
    ' Replace member code for TIME dimension in array
    dm(1, 1) = member.Code
    ' Get the crossing for this member
    Set crossing = cube.crossing(dm)
    If crossing Is Nothing Then
        MsgBox "Null Crossing"
        Exit Sub
    End If

    ' Add this crossing to the collection
    Call crossings.Add(crossing)
Next member

' Execute query to fetch values for each crossing in collection
cube.ExecuteQuery

' Display values for each member of TIME dimension
For Each crs In cube.crossings
    MsgBox crs.GetMemberCode("TIME") + " = " + Str(crs.Value)
Next crs
End Sub

```

After you perform a query, the values that the query returns are available locally. Before performing any additional queries, you would call the cube's `ClearQuery` method and then define the parameters for the new query.

You must specify each of the server hierarchy members. To select the default Read member, you can use code such as the following:

```

dm(10, 0) = "SOURCE"
dm(10, 1) = cube.Hierarchies("SOURCE").ReadDefaultMember.Code

```

## Writing Values to a Cube

To write values to a cube, you can call the cube's `Write` method with the crossing and new value as arguments, or you can set the `NewValue` property for each crossing that you want to affect and then call the cube's `BatchWriteNew` method. Keep the following points in mind:

- These methods can be used only in a data-entry form or form template.

- They cannot be used to write to a crossing that contains a parent member in any of its dimensions.
- The methods ignore the **Delay writeback until refresh** option even if that setting is enabled in the form's table properties.
- They do not honor member security.
- The methods do not honor driver formulas. As an alternative, you can include the driver formula calculations in your code or execute the **Run driver formula** function on the form set. See the online Help for SAS Financial Management Studio, or the description of the generateFormulaFacts method in [“The Model Class \(Financial Models Only\)”](#) on page 41.

## Property Summary

**Table 7.9** FMCube Class Property Summary

Property	Description
Property <b>Code</b> As String	The code for this cube (for example, <b>Default_Model</b> ). Read-only.
Property <b>Crossings</b> As FMCrossingsCollection	The collection of crossings in this cube. Read-only.
Property <b>CurrencyHierarchy</b> As FMHierarchy	An FMHierarchy object that contains the currency hierarchy for this cube. Read-only.
Property <b>Description</b> As String	The description of this cube. Read-only.
Property <b>Hierarchies</b> As FMHierarchiesCollection	<p>A collection of hierarchies (both server hierarchies and client attribute hierarchies) in this cube. Read-only.</p> <p>Server hierarchies are hierarchies that are defined on the server and that are being used in the specified results model, either directly (via an FMCube object) or via a virtual cube that is attached to an FMTable object. They are based on the required dimensions (such as <b>ACCOUNT</b>, <b>ANALYSIS</b>, and <b>TIME</b>) and any custom defined dimensions (such as <b>PRODUCT</b> or <b>COSTCENTER</b>).</p> <p>Client attribute hierarchies are virtual hierarchies that exist only on the client side. They are based on dimension attributes (both system properties and custom properties).</p>
Property <b>Id</b> As Long	A unique numeric identifier from the SAS Financial Management data mart. Read-only.
Property <b>Index</b> As Long	The position of this cube within the cubes collection. Read-only.
Property <b>Name</b> As String	The name of this cube. Read-only.
Property <b>ServerHierarchies</b> As FMHierarchiesCollection	<p>The collection of server hierarchies that are associated with the cube. Read-only.</p> <p>For more information about server hierarchies, see the description of the Hierarchies property.</p>

## Method Summary

**Table 7.10** FMCube Class Method Summary

Method	Description
Function <b>BatchWrite</b> (reQuery As Boolean) As fmWriteBackReturn	Writes the accumulated transactions to the server. This method has been deprecated. Use BatchWriteNew instead.
Function <b>BatchWriteNew</b> (reQuery As Boolean) As fmWriteBackReturn()	Writes the accumulated transactions to the server. This method does not honor member security. See the note above for further restrictions. Parameters: <ul style="list-style-type: none"> <li>• <i>reQuery</i>: the requery flag. If <b>True</b>, the function performs a requery and repaint after the write operation. If <b>False</b>, the function performs a repaint only.</li> </ul> Returns: an array containing the status of the write operations. To check the status, use code such as the following: Range("B8") = addin.enumString(fmEnums_fmWriteBackReturn, rcs(0))
Sub <b>ClearQuery</b> ()	Clears the query definitions on the cube.
Function <b>Crossing</b> (item) As FMCrossing	Returns the crossing that is represented by <i>item</i> . Parameters: <ul style="list-style-type: none"> <li>• <i>item</i>: an array of <i>dimension code/member code</i> value pairs.</li> </ul> If any dimension or member is invalid, a null value is returned. Your code should check for this before proceeding.
Sub <b>ExecuteQuery</b> ()	Queries the server for all crossings that are defined for the cube.
Function <b>isWriteable</b> () As Boolean	Returns <b>True</b> if the cube is writable. This method does not honor member security.
Function <b>Write</b> (crossing As FMCrossing, newValue As Double) As fmWriteBackReturn	Writes <i>newValue</i> to the specified <i>crossing</i> . This method does not honor member security. See the note above for further restrictions. Parameters: <ul style="list-style-type: none"> <li>• <i>crossing</i>: an FMCrossing object.</li> <li>• <i>newValue</i>: the value to write.</li> </ul> Returns: the status of the Write operation. (See <a href="#">Table 7.3 on page 94.</a> )

---

## The FMCubesCollection Class

The FMCubesCollection class represents a collection of cubes that are available on the server (and that the user has permission to access). This class is a subclass of FMCollections.

The following properties are inherited from FMCollections: Count.

The following methods are inherited from FMCollections: Add, AddAll, Clear, Contains, IndexOf, Insert, InsertAll, Remove, RemoveAt, ToString.

---

## The FMHierarchiesCollection Class

The FMHierarchiesCollection class represents a collection of hierarchies. This class is a subclass of FMCollections.

The following properties are inherited from FMCollections: Count.

The following methods are inherited from FMCollections: Add, AddAll, Clear, Contains, IndexOf, Insert, InsertAll, Remove, RemoveAt, ToString.

---

## The FMHierarchy Class

The FMHierarchy class has properties and methods for accessing members of a hierarchy. For more information about types of hierarchies, see the Hierarchies property of the FMCube class ([“The FMCube Class” on page 100](#)) or the FMTable class ([“The FMTable Class” on page 113](#)).

*Note:* Some properties (such as position, role, and available members) apply only to hierarchies that are associated with a table, rather than a cube.

One method to note in the FMHierarchy class is the ShowMemberSelectionDialog method. This method displays a dialog box from which users can select a member of a specified hierarchy. The following example presents a dialog box from which the user can select a member of the TIME hierarchy.

### **Example Code 7.3** *Selecting a Hierarchy Member*

```
Dim cube As FMCube
Dim hier As FMHierarchy
Dim selmem As FMMember
Dim premem As FMMember
Dim exclude As FMMembersCollection
...
' Get an instance of the results model /cube + hierarchy
Set cube = addin.Cubes("Default_Model")
Set hier = cube.Hierarchies("TIME")

' Set a preselected member
Set premem = hier.Members("YR2005")
' ... or use the default
```

```

' Set selmem = Nothing

' Prepare a list of members to exclude from the dialog
Set exclude = hier.GetNewMembersCollection()

' Add YR1997 and all descendants of YR1997 to the list
Call exclude.Add(hier.Members("YR1997"))
For Each member In hier.GetMembers("YR1997", True, False)
    Call exclude.Add(member)
Next member

' Display the dialog with preselected member and exclusion list
Set selmem = hier.ShowMemberSelectionDialog(premem, exclude, fmDisplayMode_CodeAndDescription)
' ...or use default member and no exclusion list
' Set selmem = hier.ShowMemberSelectionDialog(Nothing, Nothing, fmDisplayMode_CodeAndDescription)

```

The `ShowMemberSelectionDialog` method displays the Select Member dialog box. In this case, **YR2005** would be pre-selected, and **YR1997** and its descendants would be excluded.

The user selects a member of the hierarchy and clicks **OK**. The return value is the selected member.

**Table 7.11** *FMHierarchy Class Property Summary*

Property	Description
Property <b>AsOf</b> As Double	The as-of date for this hierarchy. Read-only.  To view this date as an Excel date, store it in a Date field. For example:  <pre> Dim hier as FMHierarchy Dim dt as Date ... dt = hier.AsOf MsgBox Str(dt) </pre>
Property <b>AvailableMembers</b> As FMMembersCollection	A list of the hierarchy members that are available after the member selection rules have been applied. Read-only.
Property <b>Code</b> As String	The dimension code that applies to this hierarchy. Read-only.
Property <b>Description</b> As String	The description of this hierarchy. Read-only.
Property <b>DimensionCode</b> As String	The dimension code that applies to this hierarchy. Read-only.
Property <b>DimensionDescription</b> As String	The dimension description that applies to this hierarchy. Read-only.
Property <b>DimensionId</b> As Long	The dimension ID that applies to this hierarchy. Read-only.
Property <b>DimensionName</b> As String	The dimension name that applies to this hierarchy. Read-only.
Property <b>DimensionTypeCode</b> As String	The dimension type code that applies to this hierarchy. Read-only.

Property	Description
Property <b>DimensionTypeDescription</b> As String	The dimension type description that applies to this hierarchy. Read-only.
Property <b>DimensionTypeID</b> As Long	A unique numeric identifier from the SAS Financial Management data mart. Read-only.
Property <b>DimensionTypeName</b> As String	The dimension type name that applies to this hierarchy. Read-only.
Property <b>DisplayedMembers</b> As FMMembersCollection	<p>A collection of the hierarchy members that are currently being displayed. Read-only.</p> <p>This example retrieves the members of the ACCOUNT hierarchy that are currently displayed in the specified table and displays the results in a message box:</p> <pre>Dim txt as String Set hierarchy = addin.Tables(0).Hierarchies("ACCOUNT") txt = "All displayed members in " &amp; hierarchy.Description _     &amp; Chr\$(10) For Each member In hierarchy.DisplayedMembers     txt = txt &amp; " " &amp; member.Code Next member MsgBox txt</pre>
Property <b>DisplayMode</b> As fmDisplayMode	<p>The labeling method for this hierarchy, which specifies how displayed members are identified. The value can be one of the following:</p> <ul style="list-style-type: none"> <li>• <b>fmDisplayMode_Code</b></li> <li>• <b>fmDisplayMode_Name</b></li> <li>• <b>fmDisplayMode_Description</b></li> <li>• <b>fmDisplayMode_CodeAndName</b></li> <li>• <b>fmDisplayMode_CodeAndDescription</b></li> </ul> <p>Read/write.</p>
Property <b>HierarchyCode</b> As String	The code for this hierarchy. Read-only.
Property <b>HierarchyIndex</b> As Long	For a table, this value represents the index of this hierarchy in the set of dimensions that make up the query for the table. For cubes, this value is always -1. Read-only.
Property <b>ID</b> As Long	A unique numeric identifier from the SAS Financial Management data mart. Read-only.
Property <b>LeafMembers</b> As FMMembersCollection	A collection of the leaf members of this hierarchy. Read-only.
Property <b>Members</b> As FMMembersCollection	A collection of the members of this hierarchy. Read-only.
Property <b>Name</b> As String	The name of this hierarchy. Read-only.

Property	Description
Property <b>Position</b> As Long	The position of this hierarchy within its section. The section is determined by the Role property. (See below.) If the hierarchy is in the <b>Available</b> list, its position is <b>-1</b> . Read/write.
Property <b>ReadableMembers</b> As FMMembersCollection	A collection of hierarchy members that are readable by the current user. Read-only.
Property <b>ReadDefaultMember</b> As FMMember	The default Read member for this hierarchy. Read/write.
Property <b>Role</b> As fmRole	<p>The role of this hierarchy. Read/write.</p> <p>The role determines the section in which the hierarchy appears. It can have one of the following values:</p> <ul style="list-style-type: none"> <li>• <b>fmRole_Row</b>: row</li> <li>• <b>fmRole_Column</b>: column</li> <li>• <b>fmRole_Slicer</b>: slicer</li> <li>• <b>fmRole_Available</b>: available for use in a row, column, or slicer</li> </ul> <p>This example performs a pivot of a table by changing the Role and Position properties of a Hierarchy object. When the code has been executed, the ACCOUNT hierarchy is the first column heading in the table.</p> <pre>Set table = addin.Tables("NewTable0") Set hierarchy = table.Hierarchies("ACCOUNT") hierarchy.role = fmRole_Column hierarchy.Position = 0 ' Refresh with a requery table.Refresh (True)</pre>
Property <b>ShowAllMember</b> As Boolean	<p>This property applies to member property hierarchies (such as Product.Color). If <b>True</b>, the <b>All</b> member (representing all members of the hierarchy) is displayed in a dialog box when the SelectSlicerMember or ShowMemberSelectionDialog method is called. Selecting this member in the dialog box returns a member code of <b>_All</b>.</p> <p>If the hierarchy is derived from a table, then enabling ShowAllMember also affects the hierarchy display in a slicer in the table.</p> <p>The default is <b>True</b>. Read/write.</p> <p>Example: If hierProp represents a member property hierarchy, this code disables the display of <b>All</b> in the member selection dialog box for that hierarchy:</p> <pre>boolValue = hierProp.ShowAllMember hierProp.ShowAllMember = False Set selMember = hierProp.ShowMemberSelectionDialog _     (Null, Nothing, fmDisplayMode_Code) MsgBox "selected " &amp; selMember.Code</pre>
Property <b>TargetMember</b> As FMMember	The hierarchy member that a form is assigned to. Read-only.
Property <b>VCFilter</b> As Boolean	If <b>True</b> , the table is filtered so that virtual children are not included. If <b>False</b> , virtual children are included. Read/write.



Property	Description
Property <b>WriteDefaultMember</b> As FMMember	The default Write member for this hierarchy. Read/write.

**Table 7.12** FMHierarchy Class Method Summary

Method	Description
Function <b>ChangeSlicer</b> (item) As Boolean	<p>Changes the hierarchy member that is used as the slicer. For example, if you are using a member of the TIME hierarchy as a slicer, you might change from one year to another. (The hierarchy must already be functioning as a slicer. In other words, if the hierarchy is being used in a column or row or is simply available for use, the ChangeSlicer method does not work.)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><i>item</i>: the member of the hierarchy that will become the new slicer. It can be an index into the hierarchy or a member code.</li> </ul> <p>Returns: <b>True</b> if the change succeeded; otherwise, <b>False</b>.</p> <p>In this example, a member of the ACCOUNT hierarchy is being used as a slicer. The code changes the member to <b>A1000</b>:</p> <pre>Set hierarchy = addin.Tables(0).Hierarchies("ACCOUNT") rc = hierarchy.ChangeSlicer("A1000")</pre>
Function <b>GetMembers</b> (item, recurse As Boolean, reverse As Boolean) As FMMembersCollection	<p>Returns a collection of members of this hierarchy, beginning with the first child of the member that is selected by <i>item</i>.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><i>item</i>: the hierarchy member on which to begin processing. This parameter can be an index into the hierarchy or a member code.</li> <li><i>recurse</i>: the recursion flag. If <b>True</b>, the function performs a recursive search and returns all descendants. If <b>False</b>, it returns only the member's children.</li> <li><i>reverse</i>: the reverse order flag. If <b>True</b>, the function returns the results in reverse order. If <b>False</b>, the children are returned in the same order in which they appear in the hierarchy.</li> </ul> <p>This example returns all descendants of the first member of the ACCOUNT hierarchy for a table:</p> <pre>Dim txt As String Dim member As FMMember Set hierarchy = _     addin.Tables("NewTable0").Hierarchies("ACCOUNT") txt = "All descendants of " &amp; _     &amp; hierarchy.Members(0).Description &amp; " in " &amp; _     &amp; hierarchy.Description &amp; " hierarchy" &amp; Chr\$(10) For Each member In hierarchy.GetMembers(0, True, False)     txt = txt &amp; " " &amp; member.Code Next member MsgBox txt</pre>
Function <b>GetNewMembersCollection</b> () As FMMembersCollection	Returns an (empty) FMMembersCollection object.

Method	Description
Function <b>IsFlatDimensionType</b> () As Boolean	<p>Returns: <b>True</b> if this hierarchy belongs to a flat dimension type; otherwise, <b>False</b>. Read-only.</p> <p>There are three dimension types that must have flat hierarchies: ANALYSIS, CURRENCY, and FREQUENCY. In addition, a client attribute hierarchy is a flat dimension type. (For more information about client attribute hierarchies, see the description of the <code>FMTable.Hierarchies</code> property at <a href="#">“The FMTable Class” on page 113.</a>)</p>
Function <b>IsNonVirtualChildDimensionType</b> () As Boolean	<p>Returns: <b>True</b> if the hierarchy does not include virtual children; otherwise, <b>False</b>. Read-only.</p>
Function <b>IsServer</b> () As Boolean	<p>Returns: <b>True</b> if this hierarchy is defined on the server.</p>
Function <b>SelectSlicerMember</b> (DisplayMode As fmDisplayMode) As FMMember	<p>Displays the <b>Select Member</b> dialog box for a slicer.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><i>displayMode</i>: way displayed members are identified (the labeling method). The value can be one of the following: <code>fmDisplayMode_Code</code>, <code>fmDisplayMode_Name</code>, <code>fmDisplayMode_Description</code>, <code>fmDisplayMode_CodeAndName</code>, or <code>fmDisplayMode_CodeAndDescription</code>.</li> </ul> <p>Returns: the selected member of the hierarchy.</p>
Function <b>ShowMemberSelectionDialog</b> (item, exclude As FMMembersCollection, displayMode As fmDisplayMode) As FMMember	<p>Displays the Show Members dialog box, from which users can select a member of the specified hierarchy.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><i>item</i>: the member that you want to be highlighted in the dialog box.</li> <li><i>exclude</i>: a collection of members to be excluded from the dialog box. To display all members, create a collection but do not assign it any values.</li> <li><i>displayMode</i>: the way displayed members are identified (the labeling method). The value can be one of the following: <code>fmDisplayMode_Code</code>, <code>fmDisplayMode_Name</code>, <code>fmDisplayMode_Description</code>, <code>fmDisplayMode_CodeAndName</code>, or <code>fmDisplayMode_CodeAndDescription</code>.</li> </ul> <p>Returns: the selected member of the hierarchy, which can be used in several ways. For example, the selected member could be used as input to code that modifies a CDA expression.</p>

## The FMMember Class

The FMMember class represents a member of a hierarchy, which can be displayed or not. For members of displayed hierarchies, the selection rules can be modified.

*Note:* Some properties and methods (such as the SelectionRule property and the Expand method) apply only to members of hierarchies that are associated with tables.

**Table 7.13** FMMember Class Property Summary

Property	Description
Property <b>Asof</b> As Double	The as-of date for this member. Read-only. To view this date as an Excel date, store it in a Date field.
Property <b>Code</b> As String	The code for this hierarchy member. Read-only.
Property <b>Column</b> As Long	The column position of the top left cell of this member's position in the table. Read-only.
Property <b>Description</b> As String	The description of a member of a hierarchy. Read-only.
Property <b>ID</b> As Long	A unique numeric identifier from the SAS Financial Management data mart. Read-only.
Property <b>Level</b> As Long	The level of this member in the current hierarchy. The top member of the hierarchy has a level of 0. Read-only.
Property <b>Name</b> As String	The name of a member of a hierarchy. Read-only.
Property <b>Row</b> As Long	The row position of the top left cell of this member's position in the table. Read-only.

Property	Description
Property <b>SelectionRule</b> As fmSelection	<p>Gets or sets the selection rule for a displayed hierarchy member. Read/write.</p> <p>The value can be one of the following enumerated constants:</p> <p><b>fmSelection_Member</b>: selects the designated member.</p> <p><b>fmSelection_Descendants</b>: selects the entire subhierarchy subordinate to the designated member but not including the designated member itself.</p> <p><b>fmSelection_MemberAndChildren</b>: selects the designated member and all members that are immediately subordinate to it.</p> <p><b>fmSelection_MemberAndDescendants</b>: selects the entire subhierarchy from the designated member down.</p> <p><b>fmSelection_MemberAndLeaf</b>: selects the designated member and all members that are subordinate to it but that have no members under them. For example, in a Time hierarchy that included years, quarters, and months, this value would select year and months, but not quarters.</p> <p><b>fmSelection_Children</b>: selects all members that are immediately subordinate to the designated member.</p> <p><b>fmSelection_Leaf</b>: selects all members that are subordinate to the designated member but have no members subordinate to them.</p> <p><b>fmSelection_NoMember</b>: excludes the designated member from the subset. All members that are subordinate to the designated member are also excluded, unless you apply additional rules to one or more of these subordinate members.</p> <p><b>fmSelection_NoRule</b>: removes any rules from the designated member.</p>
<p>This code modifies selection rules for the <b>ACCOUNT</b>, <b>ANALYSIS</b>, and <b>ORG</b> hierarchies in a table:</p> <pre> Set addin = conn.FMAddIn Set table = addin.Tables("NewTable0")  Set hier = table.Hierarchies("ACCOUNT") hier.Members("A8000").SelectionRule = fmSelection_NoMember hier.Members("A7400").SelectionRule = fmSelection_Member hier.Members("A6300").SelectionRule = fmSelection_NoMember hier.Members("A7800").SelectionRule = _     fmSelection_MemberAndDescendants hier.Members("A7900").SelectionRule = fmSelection_NoMember hier.Members("A8100").SelectionRule = fmSelection_Member  Set hier = table.Hierarchies("ANALYSIS") hier.Members("ACTUAL").SelectionRule = fmSelection_Member hier.Members("BUDGET").SelectionRule = fmSelection_NoMember  Set hier = table.Hierarchies("ORG") hier.Members("PLD").SelectionRule = fmSelection_Member  ' Refresh the table to see the results table.Refresh (True) </pre>	

**Table 7.14** FMMember Class Method Summary

Method	Description
Sub <b>Collapse()</b>	Collapses the member to hide all its descendants.
Sub <b>Expand ()</b>	Expands the member to display all its children.
Sub <b>ExpandAll ()</b>	Expands the member to display all its descendants.
Function <b>getParent()</b> As FMMember	Returns: the parent of this member. If this member is a top-level member, it returns this member.
Function <b>IsClientAttributeFilter()</b> As Boolean	Returns: <b>True</b> if this member is a member attribute filter.
Function <b>IsClientCalculatedMember()</b> As Boolean	Returns: <b>True</b> if this member is a client calculated member.
Function <b>IsLeaf()</b> As Boolean	Returns: <b>True</b> if this member is a leaf member.
Function <b>IsReadable()</b> As Boolean	Returns: <b>True</b> if this member is readable by the current user.
Function <b>IsServer()</b> As Boolean	Returns: <b>True</b> if this member is defined on the server.
Function <b>IsVirtual()</b> As Boolean	Returns: <b>True</b> if this member is a virtual child.
Function <b>IsWriteable()</b> As Boolean	Returns: <b>True</b> if this member is writable by the current user.  This method does not honor member security for a member that is derived from a cube.

---

## The FMMembersCollection Class

The FMMembersCollection class represents members of a hierarchy. This class is a subclass of FMCollections.

The following properties are inherited from FMCollections: Count.

The following methods are inherited from FMCollections: Add, AddAll, Clear, Contains, IndexOf, Insert, InsertAll, Remove, RemoveAt, ToString.

---

## The FMTable Class

An FMTable object represents a read-only table or a data entry table. Some elements, such as layout and scale, can be manipulated directly on the table object. Other

operations, such as filtering virtual children and showing or hiding members, must be manipulated on the FMHierarchy or FMMember objects that belong to the table.

*Note:* The FMTable class does not apply to CDA tables. Use the FMCube class instead.

**Table 7.15** FMTable Class Property Summary

Property	Description
Property <b>Code</b> As String	The code for this table. Read-only.
Property <b>Credit</b> As fmCreditsDebitsDisplay	The manner in which credit values are displayed in this table. Read/write. Possible values are as follows:  <b>fmCreditsDebitsDisplay_Default</b> : uses the default for the result model  <b>fmCreditsDebitsDisplay_Negative</b> : displays credits as negative numbers  <b>fmCreditsDebitsDisplay_Positive</b> : displays credits as positive numbers
Property <b>Crossings</b> As FMCrossingsCollection	A collection of crossings in this table. Read-only.
Property <b>Debit</b> As fmCreditsDebitsDisplay	The manner in which debit values are displayed in this table. Read/write. Possible values are as follows:  <b>fmCreditsDebitsDisplay_Default</b> : uses the default for the result model  <b>fmCreditsDebitsDisplay_Negative</b> : displays debits as negative numbers  <b>fmCreditsDebitsDisplay_Positive</b> : displays debits as positive numbers
Property <b>DisplayDebitCreditOnLabel</b> As Boolean	This setting applies to account member labels. If <b>True</b> , each row and column heading contains the word ( <b>debit</b> ) or ( <b>credit</b> ), whichever is applicable. Read/write.
Property <b>FilterInvalid</b> As Boolean	If <b>True</b> , rows or columns that contain only invalid values are not displayed. Read/write.
Property <b>FilterInvalidOnColumns</b> As Boolean	If <b>True</b> , columns that contain only invalid values are not displayed. Read/write.
Property <b>FilterInvalidOnRows</b> As Boolean	If <b>True</b> , rows that contain only invalid values are not displayed. Read/write.
Property <b>FilterZeros</b> As Boolean	If <b>True</b> , rows or columns that contain only zero values are not displayed. Read/write.
Property <b>FilterZerosOnColumns</b> As Boolean	If <b>True</b> , columns that contain only zero values are not displayed. Read/write.

Property	Description
Property <b>FilterZerosOnRows</b> As Boolean	If <b>True</b> , rows that contain only zero values are not displayed. Read/write.
Property <b>FreezeCells</b> As Boolean	If <b>True</b> , users cannot alter the table layout by operations such as changing the role of dimensions, expanding or collapsing hierarchies, adding or removing filters, and adding or removing calculated members. Read/write.
Property <b>Hierarchies</b> As FMHierarchiesCollection	<p>A collection of hierarchies (both server hierarchies and client attribute hierarchies) in this table. Read-only.</p> <p>Server hierarchies are hierarchies that are defined on the server and that are being used in the specified results model, either directly (via an FMCube object) or via a virtual cube that is attached to an FMTable object. They are based on the required dimensions (such as <b>ACCOUNT</b>, <b>ANALYSIS</b>, and <b>TIME</b>) and any custom defined dimensions (such as <b>PRODUCT</b> or <b>COSTCENTER</b>).</p> <p>Client attribute hierarchies are virtual hierarchies that exist only on the client side. They are based on dimension attributes—both system properties and custom properties.</p> <p>The collection of hierarchies includes both hierarchies that are displayed in the table and hierarchies with a role of <b>fmRole_Available</b>, meaning that they are part of the table but are not currently displayed. Instead, their default read and write members are used in the table crossings.</p> <p><i>Note:</i> Custom properties hierarchies are treated like any other hierarchy.</p>
Property <b>Index</b> As Long	The position of this table within the tables collection. Read-only.
Property <b>Model</b> As String	The code for the model that is used in this table. Read/write.
Property <b>Name</b> As String	The name of this table. Read-only.
Property <b>ReadOnly</b> As Boolean	If <b>True</b> , this table is read-only. Read-only.
Property <b>RefreshOnOtherTableUpdate</b> As Boolean	<p>If <b>True</b>, this table is refreshed when other tables in the same worksheet change. Read/write.</p> <p>For example, you might set this property to <b>True</b> for a read-only table so that it is refreshed when a user enters a value in a data entry table in the same worksheet.</p>
Property <b>ScaleValue</b> As Double	The value by which displayed values are scaled. The actual computed values are divided by this number before they are displayed. Read/write.
Property <b>ServerHierarchies</b> As FMHierarchiesCollection	<p>The collection of server hierarchies that are associated with the table. Read-only.</p> <p>For more information about server hierarchies, see the description of the Hierarchies property.</p>

**Table 7.16** FMTable Class Method Summary Areas of a Table

Method	Description
Sub <b>BatchWrite</b> ()	<p>Writes the accumulated transactions to the server.</p> <p>Writeback is the process of writing back facts to the server. A writeback occurs when the user enters a value in a data entry table and presses ENTER. Normally, this operation requires one trip to the server for each value that the user enters. The BatchWrite method enables you to perform multiple updates with a single writeback. The process is as follows:</p> <ol style="list-style-type: none"> <li>1. Call the TransactionBegin method for a data entry table, to begin accumulating values.</li> <li>2. Write values to the table cells, either manually or programmatically. At this point, only the client-side representation is updated.</li> <li>3. Call the BatchWrite method to perform the writeback of the accumulated values.</li> </ol> <p>Here is an example:</p> <pre>Table.TransactionBegin For Each c In target     If c.Value &lt;&gt; newValue Then         table.Crossing(c.row, c.column).newValue = newValue     End If Next c Table.BatchWrite</pre> <p><i>Note:</i> If <b>Delay writeback until refresh</b> is enabled, the write still occurs, but the data is not sent to the server until the table is refreshed.</p>
Sub <b>Collapse</b> (member As FMMember)	<p>Collapses the selected member to hide all its descendants.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <i>member</i>: the hierarchy member to collapse.</li> </ul>
Function <b>Crossing</b> (row As Long, column As Long) As FMCrossing	<p>Returns the crossing at (<i>row</i>, <i>column</i>).</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <i>row</i> and <i>column</i>: the Excel row and column values, after converting the column letter to a number (<b>A=1</b>, <b>B=2</b>, and so on).</li> </ul>
Sub <b>Expand</b> (member As FMMember)	<p>Expands the selected member to display all its children.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <i>member</i>: the hierarchy member to expand.</li> </ul>
Sub <b>ExpandAll</b> (member As FMMember)	<p>Expands the selected member to display all its descendants.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <i>member</i>: the hierarchy member to expand.</li> </ul>
Sub <b>FilterMemberCombinations</b> (mems As FMMembersCollection)	<p>Creates and applies a multi-member table filter based on the selected members. For more information, see the description of the <b>Filter Member Combination</b> option in the online Help for the SAS Financial Management Add-In for Microsoft Excel.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <i>mems</i>: a collection of members.</li> </ul>



Method	Description
Sub <b>FilterMembers</b> (row As Long, column As Long)	<p>Creates and applies a single-member table filter based on the selected row or column heading. For more information, see the description of the <b>Filter Member Combination</b> option in the online Help for the SAS Financial Management Add-In for Microsoft Excel.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><i>row, column</i>: the coordinates of the row or column heading that is used as the filter.</li> </ul>
Function <b>isDataArea</b> (sheetName As String, row As Long, column As Long) As Boolean	<p>Returns <b>True</b> if the specified position is within the data area. For more information about the data area, see the description of the Position function.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><i>sheetName</i>: the name of the worksheet.</li> <li><i>row, column</i>: the coordinates of the position.</li> </ul>
Function <b>Pivot</b> (hierarchy As FMHierarchy, role As fmRole, position As Long) As Boolean	<p>Changes the layout of the selected table by changing the role of the hierarchy that was passed in. You must refresh the table in order to see the effects of the pivot operation.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><i>hierarchy</i>: the hierarchy whose role is to be changed.</li> <li><i>role</i>: the new role for this hierarchy. A role of <b>fmRole_Slicer</b>, <b>fmRole_Row</b>, or <b>fmRole_Column</b> places the hierarchy in the slicer, row, or column section of the table. A role of <b>fmRole_Available</b> removes the hierarchy from its previous role as a slicer, row, or column and places it in the list of available hierarchies.</li> <li><i>position</i>: the hierarchy's position within its section (slicer, row, or column). If the section contains more than one hierarchy, existing hierarchies are pushed up or down as necessary to accommodate the position that you specify for this hierarchy. A position of <b>0</b> represents the highest position for the specified role.</li> </ul> <p>Returns: <b>True</b> if the operation succeeded; <b>False</b> if the role is the same as the current role or if an error is encountered.</p> <p>This example uses the Pivot method of the Table object to change the layout of a table. Assume that the column headings for a table are <b>TIME</b> and <b>ANALYSIS</b>, and the only row heading is <b>ACCOUNT</b>. The following code removes <b>ANALYSIS</b> from the column headings and adds it to the row headings. The new row headings would be <b>ANALYSIS</b> and <b>ACCOUNT</b>, in that order.</p> <pre>Dim table as FMTable Dim hierarchy as FMHierarchy set table = addin.tables("NewTable0") Set hierarchy = table.Hierarchies("ANALYSIS") rc = table.Pivot(hierarchy, fmRole_Row, 0) table.refresh(true)</pre>

Method	Description
Function <b>Position</b> (area As fmArea, type As fmType) As Long	<p>Returns the position of an element within the table. One common use for this method is to determine a range for applying custom formats to a table.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li> <b>area</b>: the area of the table for which you want to know the position. This parameter can be one of the following: <ul style="list-style-type: none"> <li><b>fmArea_Table</b>: the entire table</li> <li><b>fmArea_Slicer</b>: the table slicer area</li> <li><b>fmArea_Row</b>: the row heading area</li> <li><b>fmArea_Column</b>: the column heading area</li> <li><b>fmArea_Data</b>: the data area</li> <li><b>fmArea_Drillpath</b>: the drill-path area of the table</li> </ul> </li> <li> <b>type</b>: the type of position to return, which can be one of the following: <ul style="list-style-type: none"> <li><b>fmType_startRow</b>: the position of the starting row of the specified area</li> <li><b>fmType_endRow</b>: the position of the ending row of the specified area</li> <li><b>fmType_startColumn</b>: the position of the starting column of the specified area</li> <li><b>fmType_endColumn</b>: the position of the ending column of the specified area</li> <li><b>fmType_width</b>: the width of the specified area, in terms of number of columns</li> <li><b>fmType_height</b>: the height of the specified area, in terms of number of rows</li> <li><b>fmType_rowOffset</b>: the number of rows before the start of this table (regardless of the area)</li> <li><b>fmType_columnOffset</b>: the number of columns before the start of this table (regardless of the <i>area</i> parameter)</li> </ul> </li> </ul> <p>Returns: a value for the specified area and type.</p> <p>This example finds the positions of the start and end rows and columns in the data area of a table:</p> <pre> startRow = table.Position(fmArea_Data, fmType_startRow) endRow = table.Position(fmArea_Data, fmType_endRow) startColumn = table.Position(fmArea_Data, fmType_startColumn) endColumn = table.Position(fmArea_Data, fmType_endColumn) </pre>

Method	Description
Sub <b>Refresh</b> (reQuery As Boolean)	<p>Refreshes the table.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><i>reQuery</i>: the requery flag. If <b>True</b>, the function performs a requery and repaint. If <b>False</b>, the function performs a repaint only.</li> </ul> <p>Consider carefully whether a requery is needed or whether a repaint is sufficient. The refresh operation requires more resources when a requery is included.</p> <p>This example refreshes the specified table and performs a requery:</p> <pre>table.Refresh(True)</pre> <p>This example repaints the table without performing a requery:</p> <pre>table.Refresh(False)</pre>
Sub <b>RemoveAllMemberCombinationFilters</b> ()	Deletes all table filters.
Function <b>TargetHierarchy</b> () As FMHierarchy	Returns the target hierarchy for this table and model.
Sub <b>TransactionBegin</b> ()	Begins accumulating transactions for later writeback using the BatchWrite method.
Sub <b>UnfilterMemberCombinations</b> (mems As FMMembersCollection)	<p>Deletes the table filter that is specified by the combination of members.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><i>mems</i>: a collection of members.</li> </ul>
Function <b>Write</b> (row As Long, column As Long, newValue As Double) As fmWriteBackReturn	<p>Writes <i>newValue</i> to the crossing that is specified by <i>row</i> and <i>column</i>.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><i>row</i> and <i>column</i>: the row and column values that determine the crossing. For column values, convert letters to numbers (for example, cell <b>A3</b> is the crossing that is determined by a row value of <b>3</b> and a column value of <b>1</b>).</li> <li><i>newValue</i>: the value to write.</li> </ul> <p>Returns: the status of the Write operation. (See <a href="#">Table 7.3 on page 94</a>.)</p> <p><i>Note:</i> If <b>Delay writeback until refresh</b> is enabled, the write still occurs, but the data is not sent to the server until the table is refreshed.</p>
Function <b>Writeable</b> (row As Long, column As Long) As Boolean	<p>Determines whether a specified crossing is writable. Read-only.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><i>row</i> and <i>column</i>: the row and column values that determine the crossing. For column values, convert letters to numbers.</li> </ul> <p>Returns: <b>True</b> if the crossing is writable; otherwise, <b>False</b>.</p>

This diagram illustrates the areas of a table. (See the Position method of the FMTable class.)

Figure 7.2 Areas of a Table

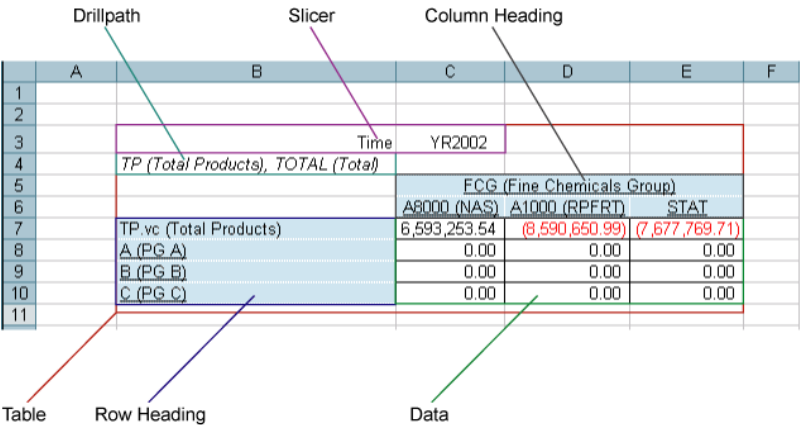


Table 7.17 FMTTable Class: Event Summary

Event	Description
Event <b>AfterRefresh()</b>	<p>Triggered after the table has been refreshed. This event might occur if the code calls the <b>Refresh</b> method of a table object, if the user selects <b>Refresh</b> or <b>RefreshAll</b> from the toolbar menu, or if the user performs some other manual action that triggers a refresh. If the user refreshes a worksheet, the table refresh event and the addin refresh event are triggered, in that order.</p> <p>Here is an example of an event handler for a table called <b>table1</b>:</p> <pre>Private Sub table1_AfterRefresh()     ' Assumes that screen updating has been disabled     ' in the BeforeRefresh event handler     ...     ' Perform some actions     ...     ' Re-enable screen updating     Application.ScreenUpdating = True End Sub</pre> <p>Notice that the name of the event handler includes the name of the object (in this case, <b>table1</b>). It applies only to this table, not to any other tables in the worksheet. If you wanted to affect a different table (for example, <b>table2</b>), you would write a second event handler, <b>table2_afterRefresh()</b>. To handle all table refresh events identically, you could use the <b>addin_AfterTableRefresh</b> event handler.</p>
Event <b>BeforeRefresh()</b>	<p>Triggered before the table is refreshed. One use for this event handler is to disable screen updating. You could re-enable screen updating in the <b>AfterRefresh</b> event handler. For more information, see the description of <b>AfterRefresh</b>.</p> <p>Here is an example of an event handler for a table called <b>table1</b>:</p> <pre>Private Sub table1_BeforeRefresh()     ' Disable screen updating     Application.ScreenUpdating = False     ' Perform some actions     ... End Sub</pre>

## The FMTablesCollection Class

The FMTablesCollection class represents a collection of tables. This class is a subclass of FMCollections.

The following properties are inherited from FMCollections: Count.

The following methods are inherited from FMCollections: Add, AddAll, Clear, Contains, IndexOf, Insert, InsertAll, Remove, RemoveAt, ToString.

## The FMUser Class

The FMUser class contains information about the user who is currently logged on.

**Table 7.18** FMUser Class Property Summary

Property	Description
Property <b>BudgetMode</b> As FMBudgetMode	Returns one of the following values: <ul style="list-style-type: none"> <li>• <b>fmBudgetMode_Create</b>: if the user is editing a template</li> <li>• <b>fmBudgetMode_Entry</b>: if the user is editing a form</li> </ul> Otherwise, the value is <b>fmBudgetMode_None</b> . Read-only.
Property <b>FormId</b> As Long	The form ID. Read-only.
Property <b>FormSetId</b> As Long	The form set ID. Read-only.
Property <b>FormSetName</b> As String	The name of the form set. Read-only.
Property <b>FormTemplateId</b> As Long	The form template ID. Read-only.
Property <b>LockName</b> As String	If the form is opened in data-entry mode, this property contains the name that is associated with the lock. The lock name can be viewed in the log. (See “ <a href="#">Activating the Log</a> ” on page 92.)
Property <b>ReadOnly</b> As Boolean	This property applies only if a user has a form open. It has a value of <b>True</b> if the form is read-only. The user might have launched the form for viewing only, or the user might not have permission to edit the form. Read-only.
Property <b>UserContext</b> As String	Returns the user context (session ID). Read-only.
Property <b>UserId</b> As String	The user ID (for example, <b>sasdemo</b> ). Read-only.
Property <b>UserName</b> As String	The user display name (for example, <b>SAS Demo User</b> ). Read-only.

Property	Description
Property <b>WorkflowMethod</b> As String	The workflow method, which can be one of the following: <b>TopDown</b> or <b>BottomUp</b> . Read-only.

---

## Including CDA Functions

To use CDA functions in your VBA code, create a string that is the CDA expression (using a function such as CDAGet or CDADesc).

This simple example creates an expression to get the value of the ColorChoice custom property that is associated with the My\_Product.TShirts dimension member. It inserts the CDA expression in the current worksheet.

```
Dim prop As String
Dim Target As Range
prop = "=CDAProperty("My_Model", "My_Product", "TShirts", "ColorChoice")"
Set Target = Cells(20, 3)
Target.Value = prop
```

## Chapter 8

# Configuring Secure Socket Layer (SSL)

---

<b>Introduction</b>	<b>123</b>
<b>Overview of Configuring SSL for SAS Financial Management</b>	<b>124</b>
<b>Configure the Web Application Server</b>	<b>124</b>
<b>Modify the Remote Services</b>	<b>125</b>
Overview	125
Modify the wrapper.conf File	125
Modify the Start-up Script	125
<b>Import the Certificate into Your Browser</b>	<b>126</b>
<b>Edit the SAS Metadata</b>	<b>126</b>
Configure the Web Applications	126
Configure the SAS Content Server	126
Modify Content Mapping	127
Modify the Foundation Services	127
<b>Modify the SAS Environment Files</b>	<b>128</b>
Overview	128
Update the EnvironmentFactory.xml File	128
Update the sasv9_usermods.cfg File	128
Update the sas-environment.xml File	129
<b>Restart and Test</b>	<b>129</b>
<b>Configuring Desktop Clients for Use with an SSL-Enabled Server</b>	<b>129</b>
Updating the INI Files	129
Import the Certificate to the Client Machines	130

---

## Introduction

This chapter explains how to configure one-way Secure Socket Layer (SSL) on systems with SAS Financial Management installed. It assumes that you have a basic understanding of SSL and that you know how to obtain and use trusted certificates.

The third-party support center has information about configuring Oracle WebLogic Server, IBM WebSphere Application Server and JBoss Application Server. See [support.sas.com/resources/thirdpartysupport/v93/appservers/index.html](http://support.sas.com/resources/thirdpartysupport/v93/appservers/index.html).

In addition, see your Web application server's documentation for SSL implementation details at the following Web sites:

- <http://www.jboss.org/docs>
- <http://www.oracle.com/technology/documentation/index.html>
- <http://www.ibm.com/support/documentation/us/en>

For information about OpenSSL, see <http://www.openssl.org>.

---

## Overview of Configuring SSL for SAS Financial Management

To configure one-way SSL for SAS Financial Management, follow these steps:

1. Make sure that SAS Financial Management is running correctly without SSL.
2. Obtain the necessary digital certificates and configure the managed servers for SSL.  
See “Configure the Web Application Server” on page 124.
3. Configure the remote services to support SSL.  
See “Modify the Remote Services” on page 125.
4. Import the server certificate into your browser.  
See “Import the Certificate into Your Browser” on page 126.
5. Configure the metadata for SSL.  
See “Edit the SAS Metadata” on page 126.
6. Modify the sas-environment.xml file and the EnvironmentFactory.xml file.  
See “Modify the SAS Environment Files” on page 128.
7. Restart the remote services and the managed servers, and verify the SSL connection.  
See “Restart and Test” on page 129.

---

## Configure the Web Application Server

Following instructions in the documentation for your Web application server, configure the Web application server for one-way SSL:

1. Obtain or create the necessary certificates and other files (keystore and truststore).  
For test purposes, use a self-signed certificate.
2. For each server, edit the applicable configuration file or settings to enable one-way SSL. Keep the following notes in mind:
  - Be sure to change the port number to the secure port for the server, and change the protocol to https.
  - For HTTPS port numbers, see “Configuring your WebLogic Application Server (Domain Configuration)” in the Instructions.html file from your installation. This



file is located in the **SAS-config-dir\Lev1\Documents** folder on the middle tier.

- (SASServer1 and SASServer2 only) In the **-Dsas.auto.publish.port** JVM argument, change the port number to the secure port for the managed server, and add an argument to set the protocol. For example:

```
-Dsas.auto.publish.port=7002
-Dsas.auto.publish.protocol=https
```

*Note:* The port number will vary.

- For WebLogic, configure the Admin server and Node Manager as well if the site requires it.
3. (WebLogic only) In the Foreign JNDI Providers service configuration, change the Provider URL of the SharedServicesJNDIProvider to reference the t3s protocol and the secure port number that applies to SASServer1.

---

## Modify the Remote Services

### Overview

Modify the JVM parameters for the SAS Remote Services to include the certificate authority (CA) keystore:

- If you run the remote services as a service, see [“Modify the wrapper.conf File” on page 125](#).
- If you run the remote services from a start-up script, see [“Modify the Start-up Script” on page 125](#).

### Modify the wrapper.conf File

If you run the SAS Remote Services as a Windows service, add the following parameters to the wrapper.conf file. This file is in the **SAS-config-dir\Lev1\Web\Application\RemoteServices** directory.

```
wrapper.java.additional.13=-Djavax.net.ssl.trustStore=path-to-keystore\truststore
wrapper.java.additional.14=-Djavax.net.ssl.trustStorePassword=password
```

These parameter values must match the values that you defined for the managed servers. The parameter numbers might be different if your site's file has more or fewer parameters.

*Note:* For two-way SSL, the file would need to contain these additional parameters:

```
wrapper.java.additional.15=-Djavax.net.ssl.keyStore="path-to-keystore\keystore"
wrapper.java.additional.16=-Djavax.net.ssl.keyStorePassword=password
```

### Modify the Start-up Script

If you run the SAS Remote Services from a batch script, add those parameters to the **start2** and **start3** sections in the RemoteServices.bat or RemoteServices.sh file.

---

## Import the Certificate into Your Browser

Import the server's certificate into your browser's trusted signer's area, so that the browser knows it can trust the certificate.

For instructions, consult the Help for your browser.

---

## Edit the SAS Metadata

### Configure the Web Applications

Configure the Web applications to support SSL, as follows:

1. On the **Plug-ins** tab of SAS Management Console, navigate to **Application Management** ⇒ **Configuration Manager**.
2. Open the properties for each application. On the **Connection** tab, modify the following properties:

Field	Instructions
Application protocol	Select <b>https</b> .
Port number	Enter the secure port number for the managed server to which you deployed this application.

*Note:* If an application (such as the SAS Web Application Themes) uses static content only, you might want that application to continue to use the HTTP protocol. Using the HTTP protocol for some applications can improve performance. However, it requires that you leave both the secure and nonsecure ports open for that managed server.

### Configure the SAS Content Server

Configure the communication protocol and port number for the SAS Content Server, as follows:

1. On the **Plug-ins** tab of SAS Management Console, navigate to **Server Manager**.
2. Open the properties for the **SAS Content Server**.
3. On the **Options** tab, modify the following properties:

Field	Instructions
Application protocol	Select <b>https</b> .
Port number	Enter the same secure port number that you used for the Web Infrastructure Platform (SASServer1).

## Modify Content Mapping

Modify the content mapping for the **SAS Folders**, as follows:

1. On the **Folders** tab of SAS Management Console, open the properties for **SAS Folders**.
2. Select the **Content Mapping** tab.
3. Select **WebDAV location** if it is not already selected.
4. From the **Server** drop-down list, select **SAS Content Server**.

As a result, the **URL** field displays the updated URL for the SAS Content Server, including the HTTPS protocol and the new port number.

## Modify the Foundation Services

Modify the WebDAV connection information in the foundation services, as follows:

1. On the **Plug-ins** tab of SAS Management Console, navigate to **Environment Management** ⇒ **Foundation Services Manager** ⇒ **Remote Services** ⇒ **Core**.
2. Open the properties for **Information Service**.
3. On the **Service Configuration** tab, click the **Configuration** button.
4. Click the **Repositories** tab.
5. In the **Information Repositories** list, select **WebDAV**. At the bottom of the page, click **Edit**.
6. In the DAV Repository Definition dialog box, change the port number to the same port number that you used for SAS Content Manager.
7. Select the **Secure** check box.
8. Save your changes.

Repeat these steps for the following foundation services:

- SASBIPortlets4.3 Local Services
- SASJSR168RemotePortlet4.3 Local Services
- SASPackageViewer4.3 Local Services
- SASPortal4.3 Local Services
- SASStoredProcess9.3 Local Services
- SASWebReportStudio4.3 Local Services
- SASFinancialManagement5.3 Local Services

## Modify the SAS Environment Files

### Overview

You must modify the SAS environment files to reflect the new protocol and port numbers.

Make a backup copy of each file before you modify it.

If you have not already done so, publish the environment files to an HTTP server, as described in “Installing the Client Applications” in the *SAS Financial Management: System Administrator's Guide*.

### Update the EnvironmentFactory.xml File

Edit the EnvironmentFactory.xml and EnvironmentFactory.odcs.xml files as follows:

1. On the middle-tier server, change directory to **SAS-config-dir\Lev1\Web\Applications\SASSolutionsServices5.3**.
2. In the EnvironmentFactory.xml file, modify the URLs as follows:
  - a. Change **http** to **https**.
  - b. Change the port numbers to reference the secured ports.
  - c. (WebLogic only) Change **t3** to **t3s**.

For example:

```
<java.naming.provider.url>t3s://server-name:7202</java.naming.provider.url>
```

3. Copy the modified EnvironmentFactory.xml file to the sas.solutions.common.war directory within the sas.solutionsservices5.3.ear application (replacing the current version of that file). If you are deploying SAS Solutions Services as an exploded EAR, this location is **SAS-config-dir\Lev1\Web\Staging\exploded\sas.solutionsservices5.3.ear\sas.solutionscommon.war**.
4. Make similar changes to the EnvironmentFactory.odcs.xml file. It is located in the **SAS-config-dir\Lev1\Web\Applications\SASODCSForSolutions5.3** directory.
5. In the deployed files for your HTTP server, modify the EnvironmentFactory.xml file as follows:
  - a. Find the section that matches your environment.
  - b. Within that section, modify the URLs as described in Step 2.

### Update the sasv9\_usermods.cfg File

Modify the sasv9\_usermods.cfg file, as follows:

1. Open the sasv9\_usermods.cfg file for editing.

This file is located on the data tier, in the **SAS-config-dir\Lev1\SASApp** directory.

2. In the JREOPTIONS, change the protocol and port number for the EnvironmentFactory.xml file as follows:

```
-JREOPTIONS=(-Denv.factory.location=http://hostname:secure-port/SASConfig/EnvironmentFactory.xml)
```

- *hostname* is the name of the middle-tier server.
- *secure-port* is the secure port number for SAS Server3, where SAS Financial Management is deployed.

### Update the sas-environment.xml File

1. In the **SAS-config-dir\Lev1\Web\Common** directory, modify the sas-environment.xml file as follows:
  - a. Find this entry:
 

```
<service-registry>
  http://server:port/SASWIPServices/remote/serviceRegistry
</service-registry>
```
  - b. Change the protocol from **http** to **https**.
  - c. Change *port* to reflect the secure port number for the sas.wip.services93.ear application.
2. If you deployed the environment files to an HTTP server, modify the sas-environment.xml file as follows:
  - a. Find the section that matches your environment.
  - b. Within that section, modify the URLs as described above.

---

## Restart and Test

1. Restart the remote services, the HTTP server (if there is one), and the managed servers.
2. Verify the SSL connection by logging on to one of the Web applications using the HTTPS protocol and new port number.
3. In a production environment, you might also want to disable the nonsecure ports.

---

## Configuring Desktop Clients for Use with an SSL-Enabled Server

### Updating the INI Files

Some customer sites require that all client communication be conducted over secure communication channels. You can configure SAS Financial Management Studio and the

SAS Financial Management Add-In for Microsoft Excel to communicate via a secure connection to the middle tier.

For SAS Financial Management Studio:

1. Edit the INI file in **SAS-install-dir**  
**\SASFinancialManagementStudio\5.3**.
2. Change the URL to the environment file to use the HTTPS protocol and the secure port. For example:  
`-Denv.definition.location=https://myhttpserver:secure-port/sas-environment.xml`

*Note:* If SAS Financial Management is using the default file, **!SASHOME**  
**\sassw.config**, it is possible to modify that file instead of the INI file. Be aware that this setting might also apply to other SAS desktop applications. For more information about this file, see “Configuring the Environment Files” in the *SAS Financial Management: System Administrator’s Guide*.

For the SAS Financial Management Add-In for Microsoft Excel:

1. Edit the **SASSolutionsOfficeClient.ini** file in the **SAS-install-dir**  
**\SASFinancialManagementAddinforMicrosoftExcel\5.3** directory.
2. Change the URL to the environment file to use the HTTPS protocol and the secure port. For example:  
`[Environment Factory]  
https://myhttpserver:secure-port/EnvironmentFactory.xml`

## Import the Certificate to the Client Machines

A customer site will deploy a signed certificate on the server. Because it is signed, there are no issues with any client, and no client component must be modified to trust this connection.

However, it is possible to use a demo or test certificate that is not signed by a trusted third-party certificate authority. In those cases (and only in those cases), it is necessary to update the client's JRE to import the demo certificate, so that the client can communicate over SSL to the test configuration on the middle-tier server.

To import the CA certificate to a client machine, follow these steps:

1. On the client machine, find the **lib\security** directory of the JRE that is used by SAS Financial Management Studio.

Typically, the JRE is specified in the `-vm` parameter of the `.INI` file for SAS Financial Management Studio. Otherwise, the default JRE on the client machine is used. For details, see “Installing Client Applications” in the *Installation Instructions for SAS Financial Management Release 5.3*.

2. Copy the certificate file from the middle-tier server to the **lib\security** directory on the client machine.
3. On the client machine, open a command prompt window and change directory to the **lib\security** directory.
4. Execute the following command:

```
..\..\bin\keytool.exe -import -alias alias -file certificate-name  
-keystore cacerts
```

*certificate-name.cer* is the name of the certificate file that you copied from the middle-tier server.

Here is a WebLogic example:

```
..\..\bin\keytool.exe -import -alias WLrootcert -file myCA.cer -keystore cacerts
```

5. The keytool program prompts you for a password. The default password is **changeit**.
6. Respond **Y** to the prompt **Trust this certificate?**

The keytool program displays a message confirming that the certificate was imported.





# Index

---

## A

AdminQuery class 17  
 alerts  
   portlets 7  
   types 7  
 AuditHistory class 22

## B

BaseApi class 24  
 BaseQuery class 26  
 bottom-up workflow 64

## C

CDA functions  
   in VBA code 122  
 cell actions  
   *See* [custom cell actions](#)  
 cell comments, extracting 28  
 CellComments class 28  
   *See also* [FMCELLC macro](#)  
 configuration directory 3  
 content management 5, 6  
 conventions 3  
 custom cell actions 73  
   invoking 80  
   JVM options 80  
   parameters 74  
   resource file 79  
   stored process 74, 77  
 CycleQuery class 30

## D

dashboard  
   *See* [Form Status dashboard](#)  
 desktop clients  
   configuring for SSL-enabled server 129  
 documentation conventions 3  
 documentation links 3

## E

EnvironmentFactory.odcs.xml file 128  
 EnvironmentFactory.xml file 14, 128, 129  
 error messages  
   for SAS Financial Management Add-In  
     for Microsoft Excel 92  
   for SAS Financial Management Java  
     API 15

## F

FMAddin class 89, 93  
 FMCELLC macro 28  
 FMCrossing class 99  
 FMCrossingsCollection class 100  
 FMCube class 100  
 FMCubesCollection class 105  
 FMHierarchiesCollection class 105  
 FMHierarchy class 105  
 FMMember class 110  
 FMMembersCollection class 113  
 FMQUERY macro 51  
   *See also* [MDX reference](#)  
   copying MDX strings 55  
   example (MDX) 54  
   example (non-MDX) 54  
   MDX queries 54  
   non-MDX queries 53  
   query data set 53  
 FMTable class 113  
 FMTablesCollection class 121  
 FMUser class 97, 121  
 Form class 33  
 Form Status dashboard 81  
   customizing 83  
   design 84  
   in dashboard viewer 82  
   in portlet 81  
   indicators 81  
   requirements for viewing 82

stored process 84  
 Form Status stored process  
   customizing 84

**G**

GETALLMODELS macro 44  
 getclass1 option 13  
 GETFORMS macro 44  
 GETFORMSETS macro 44  
 GETMODELHIERARCHIES macro 44  
 GETMODELMEMBERS macro 44  
 GETMODELPROPERTIES macro 44

**H**

HTTPS  
   *See* SSL

**J**

Javaobj  
   deleting 14  
   methods 14  
   overview 13

**L**

link to SAS Financial Management 7

**M**

MDX reference 55  
   defining a slicer 58  
   EXCLUDE clause 60  
   members 56  
   ODCS versus standard OLAP 61  
   overview 55  
   query syntax 58  
   SELECT clause 59  
   supported member functions 60  
   tuple sets 57  
   tuples 56  
   WHERE clause 58  
 METADATA\_PASSID function 13  
 Metadata class 40  
 Microsoft Excel  
   API for 88  
   custom cell actions 73  
 Model class 41  
   *See also* FMQUERY macro  
   *See also* Model macros  
 Model macros 44  
 MySQL  
   installation directory 3

**P**

picklist option 13  
 portal  
   *See* SAS Information Delivery Portal  
 portal customization  
   Alerts portlets 7  
 portlets  
   Alerts 7

**R**

Reports workspace 6  
 RPTINIT macro 15

**S**

SAS BI Dashboard  
   Form Status dashboard 81  
 SAS BI Dashboard application 82  
 SAS BI Dashboard portlet 81  
 SAS BI Dashboard Viewer 82  
 SAS Financial Management Add-In API  
   for Microsoft Excel 88  
   collections 90  
   declaring the FMAddin object 89  
   diagram of classes 89  
   events 90  
   FMAddin class 93  
   FMCrossing class 99  
   FMCrossingsCollection class 100  
   FMCube class 100  
   FMCubesCollection class 105  
   FMHierarchiesCollection class 105  
   FMHierarchy class 105  
   FMMember class 110  
   FMMembersCollection class 113  
   FMTable class 113  
   FMTablesCollection class 121  
   FMUser class 97, 121  
   log file 92  
   summary of classes 92  
   tables 90  
   working with objects 89  
 SAS Financial Management Add-In for  
   Microsoft Excel  
     configuring for SSL-enabled server 129  
 SAS Financial Management Java API 12  
   accessing object methods 14  
   AdminQuery class 17  
   AuditHistory class 22  
   authenticating the user 13  
   BaseApi class 24  
   BaseQuery class 26  
   CellComments class 28  
   CycleQuery class 30  
   error messages 15

- FMCELLC macro 28
  - FMQUERY macro 51
  - Form class 33
  - getClass1 option 13
  - handling exceptions 15
  - instantiating an object 13
  - Javaobj 13, 14
  - JRE options 14
  - log file 15
  - METADATA\_PASSID function 13
  - Metadata class 40
  - Model class 41
  - Model macros 44
  - picklist option 13
  - SAS Financial Management
    - environment 14
    - saveclass1 option 13
    - summary of classes 16
    - user authentication 13
  - SAS Financial Management Studio
    - configuring for SSL-enabled server 129
  - SAS Information Delivery Portal 5
    - content administrator 7
    - link to SAS Financial Management 7
    - page templates 7
    - SAS Financial Management content 6
  - sas-environment.xml file 129
  - saveclass1 option 13
  - Secure Socket Layer
    - See* SSL
  - SSL 123
    - and JRE on client machines 130
    - applications 126
    - content mapping 127
    - demo certificates 130
    - desktop clients 129
    - EnvironmentFactory.odcs.xml file 128
    - EnvironmentFactory.xml file 128, 129
    - foundation services 127
    - importing a certificate to client
      - machines 130
    - managed servers 124
    - overview 124
    - references 123
    - remote services 125
    - SAS Content Server 126
    - sas-environment.xml file 128, 129
    - test certificates 130
    - testing 129
  - standard reports 8
  - stored process
    - example 15
    - RPTINIT macro 15
  - stored processes 8
    - for administrators 8
    - for custom cell actions 74
    - for workflow customizations 68
    - standard reports 8
- T**
- top-down workflow 64
- V**
- validation example
    - for workflow customizations 68
  - Visual Basic for Applications (VBA) 88
- W**
- what's new 1
  - workflow
    - See also* workflow customizations
    - actions 65
    - alerts 7
    - bottom-up 64
    - definition 63
    - top-down 64
  - workflow customizations 63
    - available actions 65
    - data validation example 68
    - JVM options 67
    - Pre and Post classes 64
    - programming hooks 64
    - resource file 65
    - steps 64
    - stored process 68, 71

