

SAS[®] 9.3 Providers for OLE DB Cookbook



The correct bibliographic citation for this manual is as follows: SAS Institute Inc 2011. *SAS® 9.3 Providers for OLE DB: Cookbook*. Cary, NC: SAS Institute Inc.

SAS® 9.3 Providers for OLE DB: Cookbook

Copyright © 2011, 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, May 2011

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 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

What's New in the SAS 9.3 Providers for OLE DB vii

PART 1 Introduction 1

Chapter 1 • Introduction to the SAS Providers for OLE DB	3
About the SAS Providers for OLE DB: Cookbook	3
About the SAS Providers for OLE DB	4
Data Sources and File Types Supported by the SAS Providers	5
Features Supported by the SAS Providers	7
Tips for 64-bit Programming	8
Chapter 2 • System Requirements and Installation	11
System Requirements	11
SAS Provider for OLE DB Installation	12
Accessibility Features of the SAS Providers for OLE DB	13

PART 2 Getting Started 15

Chapter 3 • Finding Recipes	17
Using This Recipe Guide	17
Local Provider Recipes	17
IOM Provider Recipes	18
SAS/SHARE Provider Recipes	20
Base SAS Provider Recipes	21
OLAP Provider Recipes	22
Chapter 4 • Learning about SAS Connections	25
What You Need to Know about SAS Connections	25
How to Identify the SAS Providers	26
ADO Connection Properties for the SAS Providers	26
Three Ways to Open an ADO Connection Object	30
Using the Data Source Property to Specify All Connection-Related Properties	31

PART 3 Connection Recipes 35

Chapter 5 • Opening an ADO Connection Object	37
Basic Connection Recipes	37
Connecting to Local Data	38
Connecting to Local Data (Single-User Server)	39
Connecting to a Remote SAS/SHARE Server	41
Connecting to a Remote SAS Workspace Server	42
Connecting to a Remote SAS Workspace Server Using SAS Objects	45

Connecting to a Remote SAS OLAP Server	46
Chapter 6 • Managing Connections	49
Supplemental Connection Recipes	49
Prompting Users for Connection Information by Displaying the Data Link Properties Dialog Box	50
Using a Microsoft Data Link (.udl) File to Provide Persistent Connection Information	56
Controlling Data Access Permissions with a Connection	57
Managing File Formats with the Local Provider	60
Reusing an Existing IOM Workspace	63
Connecting to a Specific SAS/SHARE Server Version	64
PART 4 Data Management Recipes 65	
Chapter 7 • Accessing Specific or Protected Data	67
Data Access Recipes	67
Identifying a Data Set and Returning Results	68
Specifying a Libref to Use with the IOM Provider	69
Opening a Password-Protected Data Set	71
Accessing Third-Party Data through SAS/ACCESS Engines	72
Displaying Metadata That Is Specific to SAS Data Sets	73
Reading SAS OLAP Cubes	76
Chapter 8 • Creating, Subsetting, and Deleting Data Sets	79
Creating, Subsetting, and Deleting Data Sets Recipes	79
Creating and Deleting Data Sets	80
Subsetting Data Sets for Read-Only Sequential Access	85
Subsetting Data Sets for Random and Update Access	89
Chapter 9 • Specifying How to Display Data	91
Displaying Data Recipes	91
Using SAS Formats When You Read Data	92
Using SAS Informats When You Write Data	94
Reading User-Defined SAS Formats and Informats	96
Padding Character Data with Blanks	97
Accessing Data with a Different Encoding	98
Chapter 10 • Managing Missing Values	101
Missing Value Recipes	101
Reading Missing Values from a Data Set	102
Reading Special Numeric Missing Values from a Data Set	105
Writing Missing Values to a Data Set	108
Chapter 11 • Managing Updates	111
Updating and Locking Recipes	111
Updating Recordsets	111
Implementing a Locking Strategy	113
PART 5 Tips and Best Practices 117	

Chapter 12 • Tuning the Providers for Performance	119
Properties That Affect Performance	119
How the "CacheSize" Property Affects Performance	120
How the "Maximum Open Rows" Property Affects Performance	120
How the SAS Page Size Property Affects Performance	121
How the SAS Data Set Options Property Affects Performance	122
Chapter 13 • Writing Code That Returns Provider Information	123
How to Generate a List of Supported ADO Properties	123
How to Retrieve Version Information for a Provider	125
PART 6 Troubleshooting 127	
Chapter 14 • Handling Error Objects	129
Using ADO to Handle Errors	129
Using OLE DB to Handle Errors	131
Chapter 15 • Known Issues	133
Known Issues for All Providers	133
Known Issues for the IOM Provider	134
Known Issues for the Local Provider	135
Known Issues for the SAS/SHARE Provider	136
PART 7 Appendixes 137	
Appendix 1 • ADO: Supported Cursor and Lock Type Combinations	139
Working with Cursor and Lock Type Combinations	139
Server-Side Cursor Combinations	139
Client-Side Cursor Combinations	141
Appendix 2 • ADO: Supported Methods and Properties	143
Appendix 3 • OLE DB Properties	147
OLE DB Properties: Introduction	149
OLE DB Properties: Descriptions	153
OLE DB Properties: Sorted by ADO Name	200
OLE DB Properties: Sorted by Data Provider	203
OLE DB Properties: Sorted by Group	210
Appendix 4 • OLE DB Interfaces	215
About OLE DB Interfaces	215
Standard OLE DB Interfaces	216
OLE DB for OLAP Interfaces	217
Custom Interfaces	218
Data Set Management Using the ITableDefinition Interface	225
Appendix 5 • Schema Rowsets	229
About Schema Rowsets	230
CATALOGS Schema Rowset	232
COLUMNS Schema Rowset	232
CUBES Schema Rowset	236

DIMENSIONS Schema Rowset	237
FUNCTIONS Schema Rowset	238
HIERARCHIES Schema Rowset	239
LEVELS Schema Rowset	240
MEASURES Schema Rowset	242
MEMBERS Schema Rowset	244
PROPERTIES Schema Rowset	246
PROVIDER_TYPES Schema Rowset	247
SETS Schema Rowset	249
TABLES Schema Rowset	250
Appendix 6 • OLE DB: Format Processing	253
About Format and Informat Processing with OLE DB	253
How to Specify Format Processing When Binding Columns	255
Using Formats for Input Operations	256
Overriding Formats for Input Operations	258
Processing Informats for Output Operations	261
How to Simultaneously Bind Columns to Formats and Informats	261
Appendix 7 • OLE DB: Column Mapping and Binding	263
About the Mapping and Binding Process	263
Returning Column Metadata	263
Mapping to SAS Constructs	264
Binding to Rowset Columns	265
Appendix 8 • Customized User Help for the Data Link Properties Dialog Box	269
Data Link Properties Dialog Box (Connection Tab)	269
Data Link Properties Dialog Box (Advanced Tab)	270
Glossary	273
Index	279

What's New in the SAS 9.3 Providers for OLE DB

Overview

The SAS Providers for OLE DB have the following changes and enhancements:

- retrieve SAS data set type, label, encoding, and code page from schema tables rowset
- clarify use of the `DBBINDING.dwFlags` member
- cancel MDX queries

Retrieve SAS Data Set Type, Label, Encoding, and Code Page from the Schema Rowset

For the SAS/SHARE provider and Local provider, the SAS data set type, label, encoding, and Windows code page can be retrieved from the schema rowset. The information for these fields is the same information that is provided by the `CONTENTS` procedure.

Clarify Use of the `DBBINDING.dwFlags` Member

For customers that program to the OLE DB interface, a clarification has been made to the documentation for the `DBBINDING.dwFlags` member. Previous releases of the providers used a version of the Microsoft Active Template Library for Microsoft Visual Studio 6. This version of the library did not validate the value of this member. Beginning with the 9.3 release, the providers use the Microsoft Visual Studio 2008 version of the library. This version of the library does validate the `dwFlags` member. For the SAS providers, the value must be zero. A value of zero indicates that the provider returns text,

as opposed to HTML or a COM object. SAS providers only support returning text. For more information, see [“Binding to Rowset Columns” on page 265](#).

Cancel MDX Queries

When the OLAP provider is used to perform an MDX query, the request can be cancelled with the Cancel method. This support is limited to MDX queries that are made asynchronously and with the Execute method.

Support Internet Protocol v6 Addresses

The IOM provider and the OLAP provider support using IPv6 addresses in a connection string. An example of using an IPv6 address and an example of using an IPv4 address are added in [“Internet Protocol Version 4 and Version 6 Addresses” on page 33](#).

Part 1

Introduction

Chapter 1

Introduction to the SAS Providers for OLE DB 3

Chapter 2

System Requirements and Installation 11

Chapter 1

Introduction to the SAS Providers for OLE DB

About the SAS Providers for OLE DB: Cookbook	3
How the Cookbook Can Help You Write Applications	3
What You Should Know in Order to Use This Cookbook	3
About the SAS Providers for OLE DB	4
Data Sources and File Types Supported by the SAS Providers	5
Features Supported by the SAS Providers	7
Tips for 64-bit Programming	8

About the SAS Providers for OLE DB: Cookbook

How the Cookbook Can Help You Write Applications

The *SAS Providers for OLE DB: Cookbook* provides programming recipes that are designed to jump-start your development efforts in these ways:

- by providing ready-to-use sample code
- by encouraging best practices
- by answering frequently asked questions

You can use the recipes directly or modify them to fit your needs.

Sample code for ADO is written in Visual Basic 6. Sample code for OLE DB is written in Visual C++.

Note: Unless otherwise noted, all of the recipes can be implemented by using either ADO or OLE DB. This statement is true even if the OLE DB method is not discussed in the recipe.

What You Should Know in Order to Use This Cookbook

The cookbook assumes that the following statements are true:

- You know how to program by using ADO.
- You are familiar with the OLE DB architecture and specifications.
- You know how to program in an object-oriented or object-based language such as Visual C++ and Visual Basic.

- You are familiar with the information in resources such as these:
 - Microsoft Corporation. 1998. *OLE DB Programmer's Reference and Data Access SDK*. Redmond, WA: Microsoft Press.
 - The Microsoft Developers Network (MSDN) Library.
 - Sussman, David, and Homer, Alex. 1999. *ADO 2.1 Programmer's Reference*. Chicago: Wrox Press Inc.
 - Wood, Chuck. 1999. *OLE DB and ODBC Developer's Guide*. Foster City, CA: IDG Books Worldwide.
- Either you know how to start up the SAS server that you want to access or the server has been started for you.

About the SAS Providers for OLE DB

The SAS providers for OLE DB implement data access interfaces that conform to the OLE DB specification from Microsoft, which is built on the OLE Component Object Model (COM).

- The SAS local provider supports access to Base SAS data sets that are stored on a Windows system. If you can find a Base SAS data set by using the Windows Explorer, then you can read it by using the SAS local provider. The local provider is available in 32-bit and 64-bit versions. On a Windows x64 system, you can install both versions.
- The SAS/SHARE provider supports access to Base SAS data sets and Scalable Performance Data (SPD) server data sets via a SAS/SHARE server, as well as third-party relational data sources (with licensed SAS/ACCESS engines). The SAS/SHARE provider is available in 32-bit and 64-bit versions. On a Windows x64 system, you can install both versions.
- The SAS IOM provider supports access to Base SAS and SPD Server data sets that are managed by SAS Workspace Servers, as well as third-party relational data sources (with licensed SAS/ACCESS engines). The IOM provider is available in 32-bit and 64-bit versions. On a Windows x64 system, you can install both versions.
- The SAS OLAP provider implements the OLE DB for OLAP (ODBO) interface to enable access to cubes that are managed by SAS OLAP Servers. The OLAP provider is available in 32-bit and 64-bit versions. On a Windows x64 system, you can install both versions.
- The Base SAS provider supports access to Base SAS data sets available through a local installation of Base SAS, as well as third-party relational data sources (with licensed SAS/ACCESS engines). In this context, the Base SAS installation functions as a local, single-user server. The Base SAS provider is available in 32-bit. If access is needed to data sets with a local installation of SAS from a 64-bit application, then, SAS recommends using the 64-bit version of the SAS/SHARE provider instead.

Data Sources and File Types Supported by the SAS Providers

The following table describes the different data sources and file types that are supported by the SAS providers.

Table 1.1 Supported Data

File Type/Data Source and Description	Local Provider	IOM Provider	OLAP Provider	SAS/SHARE Provider	Base SAS Provider
SAS data files Contain both the data and the descriptor information. SAS data files have a member type of DATA.	Yes	Yes	No	Yes	Yes
SAS data views A virtual data set that points to data from other sources. SAS data views have a member type of VIEW.	No	Yes	No	Yes	Yes
Temporary SAS data sets A data set that exists only for the duration of the current program or interactive SAS session. Temporary SAS data sets are not available for future SAS sessions.	No	Yes	No	Yes	Yes
SAS index An auxiliary file that is a summary of a SAS data set. Indexes are never accessed directly, but they can provide faster access to specific observations during SQL evaluation, particularly when your data set is large.	No	Yes	No	Yes	Yes
SAS audit and backup files Auxiliary files that are used to audit the changes made to a data file.	No	No (Changes to the base data set are logged.)	No	No (Changes to the base data set are logged.)	No (Changes to the base data set are logged.)
Interface files Files created by other programs, such as ORACLE, DB2, or Sybase. SAS uses special engines to read and write the data.	No	Yes	No	Yes	Yes

File Type/Data Source and Description	Local Provider	IOM Provider	OLAP Provider	SAS/SHARE Provider	Base SAS Provider
Generation data sets Historical copies of a SAS data set.	Yes, when using libname . memname #gennum in a direct open.	Yes, when using libname . memname #gennum in a direct open. (Not valid in SQL statements.)	No	Yes, when using libname . memname #gennum in a direct open. (Not valid in SQL statements.)	Yes, when using libname . memname #gennum in a direct open. (Not valid in SQL statements.)
SAS OLAP cubes A logical set of data that is organized and structured in a hierarchical, multidimensional arrangement.	No	No	Yes	No	No

Note: For more information about SAS file types, see *SAS Language Reference: Concepts* and the *SAS Procedures Guide*.

For rectangular data sources, if you can launch SAS and access a file, then that file can be accessed by the SAS/SHARE and IOM providers with the proper server configuration.

In addition, if the file is a SAS data set created on one of the supported platforms and you can access the file from Windows Explorer, then that file can be accessed by the local provider. The platforms must be compatible with your PC platform in the following ways:

- They must use the same character set as the PC: ASCII.
- They must not require floating point conversion. PC floating point numbers are IEEE.

Note: When working with data sets that were created in a different environment, the local provider translates numeric values but not character values.

The following operating environments meet the local provider's cross-platform requirements and are supported:

- Solaris
- HP-UX
- RS 6000 AIX
- MIPS ABI
- Intel ABI
- Windows
- OS/2
- AlphaVMS

The following operating environments do not meet the local provider's cross-platform requirements and are not supported:

- ALPHA_OSF

- CMS (ebcdic)
- MVS (ebcdic)
- VAX_VMS (dfloat)

Features Supported by the SAS Providers

These features are common to all of the SAS providers:

- integrated SAS formatting services, which are the included core set of SAS formats used when reading or modifying data
- use of basic OLE DB schema rowsets, which enable consumers to obtain metadata about the data source that they use

The following table lists provider-specific functionality.

Table 1.2 Feature Comparison

Provider-Specific Features	Local Provider	IOM Provider	OLAP Provider	SAS/SHARE Provider	Base SAS Provider
supports random access by using the ADO (adOpenDynamic) cursor type and recordset bookmarks	yes	yes	no	yes	yes
supports simultaneous user updates	no	yes	no	yes	yes
supports SQL processing	no	yes	no	yes	yes
provides read and update access to all data sets available in a SAS session	no	yes	no	yes	yes
provides a choice of either exclusive access (member-level lock) or multiple user access (record-level lock) to SAS data files, selectable on a per-rowset open basis	no*	yes	no	yes	yes
provides access to SAS data files via Version 7 and later SAS/SHARE servers	no	no	no	yes	no
provides access to SAS data files on Version 8 and later SAS Workspace Servers	no	yes	no	no	no
provides access to SAS cubes on SAS 9.1 and later SAS OLAP Servers	no	no	yes	no	no

Provider-Specific Features	Local Provider	IOM Provider	OLAP Provider	SAS/SHARE Provider	Base SAS Provider
provides access to SAS data files via Version 7 and later Base SAS installations	no	no	no	no	yes
provides read-only access to Version 6 and later SAS data sets created on a Windows system without a SAS session	yes	no	no	no	no
provides read-only access to transport files and data sets created by SAS Version 7 or later on any of the following platforms: Solaris, HP-UX, RS 6000 AIX, MIPS Intel ABI, Windows, OS/2, OpenVMS. Does not require a SAS session.	yes	no	no	no	no
supports MDX processing	no	no	yes	no	no

* The local provider always enforces member-level locking.

Tips for 64-bit Programming

The Local provider, IOM provider, OLAP provider, and SAS/SHARE provider are available in both 32-bit and 64-bit versions. On a 32-bit Windows system, the 32-bit version of a provider is used. On a 64-bit Windows system, a 64-bit application uses the 64-bit version of a provider. For customers that choose to leave applications in 32-bit, but run them on a 64-bit Windows system, the 32-bit application uses the 32-bit provider version, and both run in the WOW64 compatibility environment.

The Base SAS provider remains 32-bit. For customers in a 64-bit environment that want to use the Base SAS provider for local data access, SAS recommends using the 64-bit version of the SAS/SHARE provider instead. The SAS/SHARE provider offers all the features of the Base SAS provider. When using the SAS/SHARE provider to access local data, you need to use PROC ODBCSEV to start a single-user server. A license for SAS/SHARE is not required when using the SAS/SHARE provider and PROC ODBCSEV. For more information, see [“Connecting to Local Data \(Single-User Server\)” on page 39](#).

For customers that are programming to the OLE DB interface (as opposed to ADO or .NET Framework), Microsoft made a change to the OLE DB header file to accommodate 64-bit environments. This change might require revision to the application code if you plan to migrate the application to the 64-bit environment. For more information about the change to the OLE DB header file, see [INFO: OLE DB Header File Changes for 64-Bit Platform Programming](#).

For applications that are programmed to the ADO and .NET Framework interfaces, migration challenges are largely overcome. The data access interfaces are designed to avoid the details of 32-bit and 64-bit programming. Use the latest software development

tools available from Microsoft when migrating your application from 32-bit to the 64-bit environment.

The following list identifies Microsoft resources that you might find valuable for understanding the WOW64 compatibility environment and for migrating your applications:

- [64-bit Applications](#)
- [Migration Tips](#)

Chapter 2

System Requirements and Installation

System Requirements	11
SAS Provider for OLE DB Installation	12
Accessibility Features of the SAS Providers for OLE DB	13

System Requirements

This documentation assumes that you have an existing SAS environment that includes any SAS server that you use to access data. The server determines which provider that you use in your application. The following table lists the SAS servers and their associated data providers, with the exception of the local provider, which does not require a SAS installation.

Table 2.1 SAS Servers and Their Associated Data Providers

SAS Server	SAS Provider for OLE DB
SAS Workspace Server	IOM Provider
SAS OLAP Server	OLAP Provider
SAS/SHARE server	SAS/SHARE Provider
Base SAS*	Base SAS Provider

* In this context, Base SAS functions as a local server that is accessed via a client that uses the Base SAS provider. This configuration requires that Base SAS is licensed and installed on the local machine.

For information about how to start the SAS servers, see the following SAS documentation:

Table 2.2 Documentation Resources for Starting SAS Servers

Server	Documentation Title	Section
SAS Workspace Server	<i>SAS Intelligence Platform: System Administration Guide</i>	"Using SAS Management Console to Operate SAS Servers" (refer to the information about the SAS Object Spawner because it creates the SAS Workspace Server process)
SAS OLAP Server	<i>SAS Intelligence Platform: System Administration Guide</i>	"Using SAS Management Console to Operate SAS Servers"
SAS/SHARE server	<i>SAS/SHARE: User's Guide</i>	"Starting a Server: A Fast-Track Approach"

These documents are available on the Web at <http://support.sas.com>.

Here are some additional system requirements:

- Windows operating system:
 - Windows XP Professional
 - Windows Server 2003 (Standard and Enterprise)
 - Windows Server 2008 (Standard and Enterprise)
 - Windows Vista (Home editions are not supported.)
 - Windows 7 (Home editions are not supported.)
- For Windows XP and Windows Server 2003, Microsoft Data Access Components (MDAC) Version 2.7 or later must be installed. (MDAC is available from the Microsoft Web site.)
- You must have TCP/IP network connectivity. (Only the Local provider can be used without TCP/IP network connectivity.)

SAS Provider for OLE DB Installation

You can install the SAS 9.3 Providers for OLE DB from the SAS Deployment Wizard.

You can download the most current versions of the SAS Providers for OLE DB from <http://www.sas.com/apps/demosdownloads/setupintro.jsp>. Click **SAS Providers for OLE DB**.

The providers can also be installed with applications such as SAS Enterprise Guide, SAS Add-in for Microsoft Office, and SAS Stat Studio. The IOM provider is part of the Itech Windows client install.

Accessibility Features of the SAS Providers for OLE DB

The SAS Providers for OLE DB includes accessibility and compatibility features that improve usability of the product for users with disabilities. These features are related to accessibility standards for electronic information technology adopted by the U.S. Government under Section 508 of the U.S. Rehabilitation Act of 1973, as amended. If you have questions or concerns about the accessibility of SAS products, send e-mail to accessibility@sas.com.

Part 2

Getting Started

<i>Chapter 3</i>	
Finding Recipes	<i>17</i>
<i>Chapter 4</i>	
Learning about SAS Connections	<i>25</i>

Chapter 3

Finding Recipes

Using This Recipe Guide	17
Local Provider Recipes	17
IOM Provider Recipes	18
SAS/SHARE Provider Recipes	20
Base SAS Provider Recipes	21
OLAP Provider Recipes	22

Using This Recipe Guide

After you determine which SAS server your application will access, you will also know which provider to use. For example, if your data is hosted on a SAS Workspace Server, then you need to use the IOM provider.

Different recipes apply to different providers. This chapter provides a list of recipes for each provider.

- [“Local Provider Recipes” on page 17](#)
 - [“IOM Provider Recipes” on page 18](#)
 - [“SAS/SHARE Provider Recipes” on page 20](#)
 - [“Base SAS Provider Recipes” on page 21](#)
 - [“OLAP Provider Recipes” on page 22](#)
-

Local Provider Recipes

The following table lists recipes that apply to the local provider.

Table 3.1 *Recipes and Descriptions*

Recipe	Task
“Connecting to Local Data” (p. 38)	Open an ADO Connection object to access locally stored data.

Recipe	Task
“Prompting Users for Connection Information by Displaying the Data Link Properties Dialog Box” (p. 50)	Prompt users for connection information at run time by displaying the Data Link Properties dialog box.
“Using a Microsoft Data Link (.udl) File to Provide Persistent Connection Information” (p. 56)	Reference connection information that is stored in a Microsoft Data Link (.udl) file.
“Controlling Data Access Permissions with a Connection” (p. 57)	Set permissions on your data source.
“Managing File Formats with the Local Provider” (p. 60)	Specify which SAS file format to use to access a data source and access different file formats simultaneously.
“Identifying a Data Set and Returning Results” (p. 68)	Open a specific data set and return either static or dynamic results. Your options depend on which SAS provider you are using.
“Opening a Password-Protected Data Set” (p. 71)	Open a password-protected data set.
“Displaying Metadata That Is Specific to SAS Data Sets” (p. 73)	Display metadata that is specific to SAS data sets.
“Using SAS Formats When You Read Data” (p. 92)	Use SAS formats to read data.
“Padding Character Data with Blanks” (p. 97)	Preserve trailing blanks.
“Reading Missing Values from a Data Set” (p. 102)	Test for missing values.

IOM Provider Recipes

The following table lists recipes that apply to the IOM provider.

Table 3.2 Recipes and Descriptions

Recipe	Task
“Connecting to Local Data” (p. 38)	Open an ADO Connection object to access locally stored data.
“Connecting to a Remote SAS Workspace Server” (p. 42)	Connect to a remote SAS Workspace Server using a method that enables you to use ADO objects exclusively.
“Connecting to a Remote SAS Workspace Server Using SAS Objects” (p. 45)	Connect to a remote SAS Workspace Server by using the SAS Integration Technologies Object Factory.

Recipe	Task
“Prompting Users for Connection Information by Displaying the Data Link Properties Dialog Box” (p. 50)	Prompt users for connection information at run time by displaying the Data Link Properties dialog box.
“Using a Microsoft Data Link (.udl) File to Provide Persistent Connection Information” (p. 56)	Reference connection information that is stored in a Microsoft Data Link (.udl) file.
“Controlling Data Access Permissions with a Connection” (p. 57)	Set permissions on your data source.
“Reusing an Existing IOM Workspace” (p. 63)	Reuse an existing IOM Workspace.
“Identifying a Data Set and Returning Results” (p. 68)	Open a specific data set and return either static or dynamic results. Your options depend on which SAS provider you are using.
“Specifying a Libref to Use with the IOM Provider” (p. 69)	Specify (assign) a libref.
“Opening a Password-Protected Data Set” (p. 71)	Open a password-protected data set.
“Accessing Third-Party Data through SAS/ACCESS Engines” (p. 72)	Access third-party data stores that are available through your licensed SAS/ACCESS engines.
“Displaying Metadata That Is Specific to SAS Data Sets” (p. 73)	Display metadata that is specific to SAS data sets.
“Creating and Deleting Data Sets” (p. 80)	Create and delete data sets.
“Subsetting Data Sets for Read-Only Sequential Access” (p. 85)	Execute SQL queries and commands in order to subset data sets for read-only, sequential access.
“Using SAS Formats When You Read Data” (p. 92)	Use SAS formats when reading data.
“Reading User-Defined SAS Formats and Informats” (p. 96)	Read user-defined SAS formats and informats.
“Using SAS Informats When You Write Data” (p. 94)	Use SAS informats when writing data.
“Padding Character Data with Blanks” (p. 97)	Preserve trailing blanks.
“Reading Missing Values from a Data Set” (p. 102)	Test for missing values.
“Writing Missing Values to a Data Set” (p. 108)	Write missing values to a data set.
“Updating Recordsets” (p. 111)	Perform recordset updates, including batch updates.

Recipe	Task
“Implementing a Locking Strategy” (p. 113)	Lock records, especially during update access.

SAS/SHARE Provider Recipes

The following table lists SAS/SHARE provider recipes and their descriptions.

Table 3.3 Recipes and Descriptions

Recipe	Task
“Connecting to Local Data” (p. 38)	Open an ADO Connection object to access locally stored data.
“Connecting to a Remote SAS Workspace Server” (p. 42)	Connect to a remote SAS/SHARE server.
“Prompting Users for Connection Information by Displaying the Data Link Properties Dialog Box” (p. 50)	Prompt users for connection information at run time by displaying the Data Link Properties dialog box.
“Using a Microsoft Data Link (.udl) File to Provide Persistent Connection Information ” (p. 56)	Reference connection information that is stored in a Microsoft Data Link (.udl) file.
“Controlling Data Access Permissions with a Connection” (p. 57)	Set permissions on your data source.
“Connecting to a Specific SAS/SHARE Server Version” (p. 64)	Explicitly request access to a Version 7 or Version 8 SAS/SHARE server.
“Identifying a Data Set and Returning Results” (p. 68)	Open a specific data set and return either static or dynamic results. Your options depend on which SAS provider you are using.
“Opening a Password-Protected Data Set” (p. 71)	Open a password-protected data set.
“Accessing Third-Party Data through SAS/ACCESS Engines” (p. 72)	Access third-party data stores that are available through your licensed SAS/ACCESS engines.
“Displaying Metadata That Is Specific to SAS Data Sets” (p. 73)	Display metadata that is specific to SAS data sets.
“Creating and Deleting Data Sets” (p. 80)	Create and delete data sets.
“Subsetting Data Sets for Read-Only Sequential Access” (p. 85)	Execute SQL queries and commands in order to create subsets of data sets for read-only, sequential access.

Recipe	Task
“Subsetting Data Sets for Random and Update Access” (p. 89)	Use the WHERE clause with the SAS SQL procedure to subset data sets for random and update access.
“Using SAS Formats When You Read Data ” (p. 92)	Use SAS formats when reading data.
“Using SAS Informats When You Write Data” (p. 94)	Use SAS informats when writing data.
“Reading User-Defined SAS Formats and Informats” (p. 96)	Read user-defined SAS formats and informats.
“Padding Character Data with Blanks” (p. 97)	Preserve trailing blanks.
“Reading Missing Values from a Data Set” (p. 102)	Test for missing values.
“Writing Missing Values to a Data Set” (p. 108)	Write missing values to a data set.
“Updating Recordsets” (p. 111)	Perform recordset updates, including batch updates.
“Implementing a Locking Strategy” (p. 113)	Lock records, especially during update access.

Base SAS Provider Recipes

The following table lists the Base SAS provider recipes and their descriptions.

Table 3.4 Recipes and Descriptions

Recipe	Task
“Connecting to Local Data (Single-User Server)” (p. 39)	Start a local server for a single user. In this context, a local installation of Base SAS functions as the server.
“Prompting Users for Connection Information by Displaying the Data Link Properties Dialog Box” (p. 50)	Prompt users for connection information at run time by displaying the Data Link Properties dialog box.
“Using a Microsoft Data Link (.udl) File to Provide Persistent Connection Information ” (p. 56)	Reference connection information that is stored in a Microsoft Data Link (.udl) file.
“Controlling Data Access Permissions with a Connection” (p. 57)	Set permissions on your data source.

Recipe	Task
“Identifying a Data Set and Returning Results” (p. 68)	Open a specific data set and return either static or dynamic results. Your options depend on which SAS provider you are using.
“Opening a Password-Protected Data Set” (p. 71)	Open a password-protected data set.
“Accessing Third-Party Data through SAS/ACCESS Engines” (p. 72)	Access third-party data stores that are available through your licensed SAS/ACCESS engines.
“Displaying Metadata That Is Specific to SAS Data Sets” (p. 73)	Display metadata that is specific to SAS data sets.
“Creating and Deleting Data Sets” (p. 80)	Create and delete data sets.
“Subsetting Data Sets for Read-Only Sequential Access” (p. 85)	Execute SQL queries and commands in order to subset data sets for read-only, sequential access.
“Subsetting Data Sets for Random and Update Access” (p. 89)	Use the WHERE clause with the SAS SQL procedure to subset data sets for random and update access.
“Using SAS Formats When You Read Data ” (p. 92)	Use SAS formats when you read data.
“Using SAS Informats When You Write Data” (p. 94)	Use SAS informats when you write data.
“Padding Character Data with Blanks” (p. 97)	Preserve trailing blanks.
“Reading Missing Values from a Data Set” (p. 102)	Test for missing values.
“Writing Missing Values to a Data Set” (p. 108)	Write missing values to a data set.
“Updating Recordsets” (p. 111)	Perform recordset updates, including batch updates.
“Implementing a Locking Strategy” (p. 113)	Lock records, especially during update access.

OLAP Provider Recipes

The following table provides a list of SAS OLAP provider recipes and their descriptions.

Table 3.5 Recipes and Descriptions

Recipe	Task
“Connecting to a Remote SAS OLAP Server” (p. 46)	Connect to a remote SAS OLAP Server.*
“Prompting Users for Connection Information by Displaying the Data Link Properties Dialog Box” (p. 50)	Prompt users for connection information at run time by displaying the Data Link Properties dialog box.
“Using a Microsoft Data Link (.udl) File to Provide Persistent Connection Information” (p. 56)	Reference connection information that is stored in a Microsoft Data Link (.udl) file.
“Controlling Data Access Permissions with a Connection” (p. 57)	Set permissions on your data source.
“Reading SAS OLAP Cubes” (p. 76)	Read SAS OLAP cubes.

* Although it is technically possible to make a local connection to a SAS OLAP Server, typically the connection is remote.

Chapter 4

Learning about SAS Connections

What You Need to Know about SAS Connections	25
How to Identify the SAS Providers	26
ADO Connection Properties for the SAS Providers	26
Local Provider Properties	26
Base SAS Provider Properties	27
SAS/SHARE Provider Properties	28
IOM Provider Properties	28
OLAP Provider Properties	29
Three Ways to Open an ADO Connection Object	30
Using the Data Source Property to Specify All Connection-Related Properties . .	31
What Is the Data Source URI Format?	31
Data Source URI Protocols	32
Using a dataSourceString to Hold Connection Information	32
Internet Protocol Version 4 and Version 6 Addresses	33

What You Need to Know about SAS Connections

In most cases, you can use the SAS providers for OLE DB to perform the tasks that are described in the OLE DB specification. The only difference is the connection information. This chapter contains the following connection information that is specific to the SAS providers:

- how to identify each provider.
- a list of ADO connection properties for each provider.
- sample code that illustrates three ways to open an ADO Connection object. The sample code uses the ProgIDs and connection properties that are discussed in this chapter.
- an explanation of what the Data Source URI format is and how to use it when making connections. The Data Source URI format is the recommended way to specify connection information if you are running SAS 9.2 or newer and accessing data with a 9.2 or newer SAS Workspace Server or SAS OLAP Server.

TIP Refer to the information in this chapter when you are writing your application.

See Also

[“Basic Connection Recipes” on page 37](#)

How to Identify the SAS Providers

To specify the provider in your application, you enter its ProgID, which is a unique name that identifies a COM component. All providers support both version-dependent and version-independent ProgIDs. You can install different versions of the providers on the same system and write applications that use specific versions.

Table 4.1 ProgIDs and Examples

Type of Application	Form to Use	Examples*
Always uses the latest version of the provider that is installed on a machine.	"sas.provider"	"sas.LocalProvider" "sas.ShareProvider" "sas.BaseSASProvider" "sas.IOMProvider" "sas.OLAPProvider"
Only uses a specific version of a provider.	"sas.provider.major version.minor version"	"sas.LocalProvider.9.3"

* The ProgIDs are not case-sensitive.

To maintain compatibility with previous releases, the special ProgID "sas.provider.1" as a synonym for the version-independent ProgID. For example, the last IOM provider that was installed on the system can be identified in either of these two ways:

- "sas.IOMProvider"
- "sas.IOMProvider.1"

The IOM provider for SAS 9.3 can be specifically identified as "sas.IOMProvider.9.3".

See Also

[“Basic Connection Recipes” on page 37](#)

ADO Connection Properties for the SAS Providers

Local Provider Properties

This table lists the properties that the local provider supports on the ADO Connection object.

Table 4.2 Connection Properties for the Local Provider

Property	Description/Value	Required?
"Data Source"	A directory that contains the Base SAS data set that you want to access with the connection. The directory specification can be a local drive, a mounted drive, or a network drive. If you want to specify a fully qualified file path when you open a record set, set this property to "_LOCAL_".	Yes
"Mode"	The access permissions for the "Data Source." Valid values are adModeRead or adModeReadWrite. The default is adModeReadWrite in conjunction with adModeShareDenyNone.	No
"SAS File Format"	The SAS file format to associate with the "Data Source" value. If this property is not explicitly set, then the provider examines the value of the "Data Source" property to determine which file format to use. If the provider cannot determine the format, then it uses the "SAS Default File Format" property. "V9" can be used for Version 7 and Version 8 SAS data sets. For Version 6 data sets, use "V6". The 64-bit version of the local provider only works with "V9."	No

Base SAS Provider Properties

This table lists the properties that the Base SAS provider supports on the ADO Connection object.

Table 4.3 Connection Properties for the Base SAS Provider

Property	Description/Value	Required?
"Data Source"	The server ID that is established by the value of the "SAS Parameters" property.	Yes
"SAS Executable"	The fully qualified path that includes the SAS executable file (sas.exe). The default value for this property is the standard installation path of the latest major SAS release. To start different server versions, set the path to the version that you want to start.	Yes
"SAS Parameters"	The command line options that are used to start the SAS executable. This command line must include code to start the local server—for example: <code>-initstmt %sasodbc(sdplserv) -icon -nologo -notutorialdlg</code>	Yes
"Mode"	The access permissions for the "Data Source." Valid values are adModeRead or adModeReadWrite. The default is adModeReadWrite in conjunction with adModeShareDenyNone.	No

Property	Description/Value	Required?
"SAS Working Directory"	The fully qualified path to the default directory for the SAS session. This directory usually contains your SAS program files and documents. The default value for this property is the standard installation path of the latest major SAS release.	No
"SAS Server Release"	The major release number of the Base SAS installation that you are using. Valid values are 7, 8, and 9 (the default).	No

SAS/SHARE Provider Properties

This table lists the properties that the SAS/SHARE provider supports on the ADO Connection object.

Table 4.4 Connection Properties for the SAS/SHARE Provider

Property	Description/Value	Required?
"Data Source"	The server ID that is established by the server administrator when the SAS/SHARE server is started.	Yes
"Mode"	The access permissions for the "Data Source." Valid values are adModeRead or adModeReadWrite. The default is adModeReadWrite in conjunction with adModeShareDenyNone.	No
"Location"	The node that the server is running on.	Only if the node is not the one that is running the ADO application. For example, "Location" is required for remote-server access.
"User ID"	The user ID that the user provides to access a server that requires authentication.	Might be required by the server.
"Password"	The password that the user provides to access a server that requires authentication.	Yes, if the "User ID" is required.
"SAS Server Access Password"	The server-access password if one was established by the server administrator when the SAS/SHARE server was started.	Might be required by the server.
"SAS Server Release"	The major release number of the server that you are using. Valid values are 7, 8, and 9 (the default).	No

IOM Provider Properties

This table lists the properties that the IOM provider supports on the ADO Connection object.

Table 4.5 Connection Properties for the IOM Provider

Property	Description/Value	Required?
"Data Source"	Use the "_LOCAL_" keyword to indicate a new, locally instantiated SAS Workspace Server. To specify a remote server, set this property to any string that you want to associate with the connection.	Yes
"User ID"	The ID that the user provides to access a server that requires authentication.	Might be required by the server.
"Password"	The password that the user provides to access a server that requires authentication.	Yes, if the "User ID" is required.
"SAS Workspace ID"	In the IOM model, a workspace represents a single SAS session. For each workspace, the SAS Workspace Manager generates a unique ID that can be used to explicitly identify the workspace.	No
"SAS Port"	The TCP/IP port number of a remote SAS Workspace Server.	For a remote connection, you must specify either this property or the "SAS Service Name" property.
"SAS Service Name"	A logical reference to the TCP/IP port number associated with a remote SAS Workspace Server.	For a remote connection, yes, unless "SAS Port" is specified.
"SAS Machine DNS Name"	The network DNS name of a remote SAS Workspace Server or the IP address of the server.	Yes, for remote connections.
"SAS Protocol"	The protocol to use when you connect to a remote SAS Workspace Server. Valid values are ProtocolBridge, ProtocolCom, or ProtocolCorba.	Yes, for remote connections.
"SAS Server Type"	The type of server. For a SAS Workspace Server, the correct value is DBPROPVAL_DST_TDP, which identifies a tabular data server.	No

OLAP Provider Properties

This table lists the properties that the OLAP provider supports on the ADO Connection object.

Table 4.6 Connection Properties for the OLAP Provider

Property	Description/Value	Required?
"Data Source"	The name of the SAS OLAP server to which you are connecting. Typically, the value is the IP address of the remote SAS OLAP server. (Local SAS OLAP Server connections are uncommon.)	Yes

Property	Description/Value	Required?
"User ID"	The ID that the user provides to access a server that requires authentication.	Might be required by the server.
"Password"	The password that the user provides to access a server that requires authentication.	Yes, if the "User ID" is required.
"SAS Port"	The TCP/IP port number of a remote SAS OLAP Server.	Must specify either this property or the "SAS Service Name" property.
"SAS Service Name"	A logical reference to the TCP/IP port number that is associated with a remote SAS OLAP Server.	Yes, unless "SAS Port" is specified.
"SAS Protocol"	The protocol to use when you connect to a remote SAS OLAP Server. Valid values are ProtocolBridge, ProtocolCom, or ProtocolCorba.	Yes, for remote connections.

See Also

[“Basic Connection Recipes” on page 37](#)

Three Ways to Open an ADO Connection Object

In each connection recipe, one of the following methods is used to open the ADO Connection object. This topic uses connection scenarios and sample code to introduce you to each method.

Table 4.7 Samples That Illustrate Three ADO Connection Methods

Method	Connection Scenario and Sample Code
Specify the Connection object properties in a one-line, quoted connection string.	<p>You are using the most recently installed version of the SAS/SHARE provider to open a read-only connection to a remote server that requires user authentication.</p> <pre>Dim obConnection As New ADODB.Connection obConnection.Open "Provider=sas.ShareProvider; _ Data Source=shr1;Mode=adModeRead; _ Location=ShareServer.example.com; _ User ID=tjones;Password=e7tjb"</pre>
Set the Connection object ConnectionString property and then call the Open method without specifying any parameters.	<p>The SAS 9.3 version of the IOM provider is being used to open a bridge connection to a remote server with host authentication.*</p> <pre>Dim obConnection As New ADODB.Connection obConnection.ConnectionString = "Provider=SAS.IOMProvider.9.3; _ Data Source=iom-bridge://workspace.example.com:8591; _ User ID=tjones;Password=e7tjb" obConnection.Open</pre>

Method	Connection Scenario and Sample Code
Set the Connection object Provider property and then set individual property values by using the Connection objects Properties collection.	<p>You are using the most recently installed version of the local provider to open a local connection to a SAS Version 6 data set located in <code>c:\v6data</code>.</p> <pre>Dim obConnection As New ADODB.Connection obConnection.Provider = "sas.LocalProvider" obConnection.Properties("Data Source") = "c:\v6data" obConnection.Properties("SAS File Format") = "V6" obConnection.Open</pre>

* This sample code uses the Data Source URI format.

See Also

- [“How to Identify the SAS Providers” on page 26](#)
- [“ADO Connection Properties for the SAS Providers” on page 26](#)
- [“What Is the Data Source URI Format?” on page 31](#)
- [“Basic Connection Recipes” on page 37](#)

Using the Data Source Property to Specify All Connection-Related Properties

What Is the Data Source URI Format?

The Data Source URI format enables you to use the "Data Source" property to specify all connection-related properties. You can use this format with the IOM and OLAP providers in order to access SAS Workspace Servers and SAS OLAP Servers, respectively. You must be running SAS 9.2 or newer.

You can use this format in two ways:

- You can assign the connection information directly to the "Data Source" property as shown in this sample connection string.

```
Dim obConnection As New ADODB.Connection
connectionString = "Provider=sas.OLAPProvider; _
Data Source=iom-com://olap.example.com; _
User ID=myuserid; _
Password=mypassword"
```

Note: For more examples, see [“Basic Connection Recipes” on page 37](#).

- You can define a `dataSourceString` to hold the connection information. You then specify the `dataSourceString` as the value of the "Data Source" property as shown in this sample code:

```
Dim obConnection As New ADODB.Connection
connection.Properties("Data Source") = dataSourceString
```

Note: For more examples, see [“Using a dataSourceString to Hold Connection Information” on page 32](#).

Note: If you need to use the characters % or =, you must replace them with their URL-encoded values. The value for % is %25. The value for = is %3D. There are no other URL requirements.

Data Source URI Protocols

The Data Source URI format supports the following protocols, which you specify in your code.

Table 4.8 Data Source URI Protocols

URI Scheme	Protocol
iom-bridge	Bridge
iom-com	COM
iom-name	Not specified. The name is extracted from the metadata server.
iom-id	Not specified. An existing connection is used.
iom	Bridge or COM. This scheme is the only way to use Integrated Windows Authentication (IWA)*. However, you are not required to use IWA.

* For information about how IWA is used, see "Integrated Windows Authentication" in the *SAS Intelligence Platform: Security Administration Guide*.

Using a dataSourceString to Hold Connection Information

You construct the dataSourceString value based on the server information (for example, port, host name, and logical name) and the protocol that you want to use. This table contains examples.

Table 4.9 Sample dataSourceString Values

URI Scheme	Protocol, Authentication	Server Port	Host/Logical Name	Sample dataSourceString Value
iom-bridge	Bridge, Host	1234	example.com	dataSourceString = "iom-bridge://example.com:1234"
iom-com	COM, Host	COM does not have a port that is exposed to the user.	example.com	dataSourceString = "iom-com://example.com"
iom-name	Not specified. The name is extracted from the metadata server.		"My Logical Name"	dataSourceString = "iom-name://My Logical Name"

URI Scheme	Protocol, Authentication	Server Port	Host/Logical Name	Sample dataSource String Value
iom-id	Not specified. An existing connection is reused.			dataSourceString = "iom-id://ws.UniqueIdentifier"
iom	Bridge, IWA	1234	example.com	dataSourceString = "iom://example.com:1234;" & _ "Bridge;SECURITYPACKAGE=Negotiate"
iom	COM, IWA	COM does not have a port that is exposed to the user.	example.com	dataSourceString = "iom://example.com;" & _ "COM;SECURITYPACKAGE=Negotiate"

* This logical name has been defined in a SAS Metadata Server. This sample also assumes that the SAS Metadata Server has been identified for the provider. For example, this code might be used to identify a SAS Metadata Server: *iom-bridge://metadata.example.com:8561*

Internet Protocol Version 4 and Version 6 Addresses

The previous section describes using host names in the dataSourceString value. It is possible to use both internet protocol version 4 addresses (IPv4) and internet protocol version 6 addresses (IPv6) in place of the host name.

For IPv4 addresses, replace the host name with the address in dot-decimal notation, as in the following iom-bridge example:

```
dataSourceString = "iom-bridge://10.0.0.15:8561;"
```

For IPv6 addresses, enclose the colon-delimited hexadecimal address in brackets, as in the following iom-bridge example:

```
dataSourceString = iom-bridge://[fd07:af30:a3fc:a29:20d:60ff:fe4d:2ee]:8561
```

See Also

- “How to Identify the SAS Providers” on page 26
- “ADO Connection Properties for the SAS Providers” on page 26
- “Three Ways to Open an ADO Connection Object” on page 30
- “Basic Connection Recipes” on page 37

Part 3

Connection Recipes

<i>Chapter 5</i>	
Opening an ADO Connection Object	37
<i>Chapter 6</i>	
Managing Connections	49

Chapter 5

Opening an ADO Connection Object

Basic Connection Recipes	37
Connecting to Local Data	38
Goal	38
Implementation	38
Connecting to Local Data (Single-User Server)	39
Goal	39
Implementation	39
Connecting to a Remote SAS/SHARE Server	41
Goal	41
ADO Implementation	41
OLE DB Implementation	41
Connecting to a Remote SAS Workspace Server	42
Goal	42
Implementation	42
Connecting to a Remote SAS Workspace Server Using SAS Objects	45
Goal	45
Implementation	45
Connecting to a Remote SAS OLAP Server	46
Goal	46
Implementation	47

Basic Connection Recipes

This chapter provides sample code that you can use to open an ADO Connection object for access to local and remote data. Here is a list of the recipes in this chapter:

- [“Connecting to Local Data” on page 38](#)
- [“Connecting to Local Data \(Single-User Server\)” on page 39](#)
- [“Connecting to a Remote SAS/SHARE Server” on page 41](#)
- [“Connecting to a Remote SAS Workspace Server” on page 42](#)
- [“Connecting to a Remote SAS Workspace Server Using SAS Objects” on page 45](#)
- [“Connecting to a Remote SAS OLAP Server ” on page 46](#)

See Also

- “Supplemental Connection Recipes” on page 49
- “Data Access Recipes” on page 67

Connecting to Local Data

Goal

You want to open an ADO Connection object in order to access data that is stored in one of these locations:

- your local machine
- a SAS Workspace server that is running on your local machine
- a SAS/SHARE server that is running on your local machine

This recipe applies to the local, SAS/SHARE, and IOM providers. Sample code is included. This recipe applies only to ADO.

Note: Sample code for the Base SAS provider can be found in “[Connecting to Local Data \(Single-User Server\)](#)” on page 39. There is no recipe for making a local connection by using the OLAP provider. Although it is technically possible to make a local connection to a SAS OLAP Server, a local configuration is unlikely.

Implementation

A local connection is the simplest connection that you can make. The following table explains the required scenario and contains sample code.

Table 5.1 Sample Code for Local Connections

Provider	Connection Scenario and Sample Code
local provider	<p>The data is on your local machine. You specify the physical location.</p> <pre>Dim obConnection As New ADODB.Connection obConnection.Provider = "sas.LocalProvider" obConnection.Properties("Data Source") = "c:\MySasData" obConnection.Open</pre>
IOM provider	<p>The SAS Workspace Server is running on your local machine.</p> <pre>Dim obConnection As New ADODB.Connection Dim connectionString As String connectionString = "Provider=SAS.IOMProvider;Data Source=_LOCAL_" obConnection.Open connectionString</pre> <p>After the Connection object is opened, it is assigned to the specified server for the duration of the session.</p>

Provider	Connection Scenario and Sample Code
SAS/SHARE provider	<p>The SAS/SHARE server is running on your local machine. You specify the server's ID as the value of the "Data Source" property.</p> <pre data-bbox="411 317 954 457">Dim obConnection As New ADODB.Connection obConnection.Provider = "sas.SHAREProvider" obConnection.Properties("Data Source") = "shr1" obConnection.Open</pre> <p>After the Connection object is opened, it is assigned to the specified server for the duration of the session.</p>

For an explanation of the properties used in the sample code, see [“ADO Connection Properties for the SAS Providers”](#) on page 26.

Connecting to Local Data (Single-User Server)

Goal

You want your application to use the Base SAS provider to start a local server for use by a single user. In this context, a local installation of Base SAS operates as the server.

This recipe applies to the Base SAS provider. Sample code for ADO is included.

Implementation

Adding Server Information to the TCP/IP Services File

Before you can open this connection, you must modify the TCP/IP services file to include an entry for the SAS server (that is, the Base SAS installation) that the provider will use. The TCP/IP services file contains information about the services available on the local machine, including available SAS servers. For each named service, the file specifies a port number, a protocol name, and any service alias.

Note: The TCP/IP services file is not stored in the same location on all platforms.

However, for the Windows NT, Windows 2000, and Windows XP platforms, the services file is stored at `c:\winnt\system32\drivers\etc\services`.

Entries in the services file have the following general form:

```
<service-name> <port-number/protocol-name> <aliases> # <comments>
```

To add the server information to the file:

1. Enter the server ID as the service name. The server ID is usually a case-sensitive string that is one to eight characters in length. The first character is a letter or an underscore; the remaining seven characters can include letters, digits, underscores, the dollar sign (\$), or the at (@) symbol.

Note: When you write the code to start the server, you use the same server ID that you enter as the value for the "Data Source" property. You also enter the server ID as the value to be passed into the %sasodbc macro for the "SAS Parameters" property as shown in the sample code (see [“Sample Code for Starting a Local Server”](#) on page 40).

2. Set the port number to a number above 1024 that is not already in use. Any port number that is equal to or less than 1024 is reserved. For larger networks, obtain the port number from your network administrator.
3. Set the protocol name to **TCP**.

For example, to configure the Base SAS provider to access a local server named **sdplserv**, you add the following entry to the services file (substituting the appropriate port number):

```
sdplserv          5420/tcp      # Base SAS Provider Local Server
```

Sample Code for Starting a Local Server

After you add the server entry to the TCP/IP services file, you can use the following Visual Basic code to start the local server. This sample uses the Connection object Properties collection to specify individual property values. The server ID is **sdplserv** (which is the default).

```
Dim obConnection As New ADOConnection

obConnection.Provider = "sas.BaseSASProvider"
obConnection.Properties("Data Source") = "sdplserv"
obConnection.Properties("SAS Executable") = "C:\\Program Files\\SASHome\\SASFoundation\\9.3\\sas.exe"
obConnection.Properties("SAS Parameters") = "-initstmt %sasodbc(sdplserv) -icon -nologo -notutorialdlg"
obConnection.Properties("SAS Working Directory") = "C:\\Program Files\\SASHome\\SASFoundation\\9.3\\"
obConnection.Open
```

TIP To identify a version of Base SAS that is older than Version 9 (the default), provide a value for the "SAS Server Release" property. For information about using this property, see [“Connecting to a Specific SAS/SHARE Server Version” on page 64](#).

For an explanation of the properties used in the sample code, see [“Base SAS Provider Properties” on page 27](#).

A Closer Look at the Value of the 'SAS Parameters' Property

The following line of code specifies the default value for the "SAS Parameters" property.

```
obConnection.Properties("SAS Parameters") = "-initstmt %sasodbc(sdplserv) -icon -nologo -notutorialdlg"
```

The initialization statement **-initstmt** executes a SAS macro named **%sasodbc**, which in turn invokes the server identified by **sdplserv**. The **-icon** option immediately minimizes the SAS session. The **-nologo** option suppresses the SAS logo and copyright information. The **-notutorialdlg** option suppresses starting the SAS tutorial.

The **%sasodbc** macro ships with Base SAS and is found in *!sasroot\core\sasmacro\sasodbc.sas*. This macro, which was created for use with the SAS ODBC driver, executes PROC ODBCSEVER. To specify PROC ODBCSEVER options in addition to the **sdplserv** server ID, you can modify the *sasodbc.sas* file. You can also modify *sasodbc.sas* in order to include additional SAS system options or SAS statements such as the LIBNAME statement.

CAUTION:

If you also use the SAS ODBC Driver, you should create a separate SAS macro file for use with the Base SAS provider.

Note: The PROC ODBCSEVER options are identical to the PROC SERVER statement options. For more information about the options, see the *SAS/SHARE User's Guide*.

Note: !SASROOT is the logical name for the directory in which you install SAS.

See Also

[“Connecting to Local Data” on page 38](#)

Connecting to a Remote SAS/SHARE Server

Goal

You want your application to connect to a remote SAS/SHARE server.

This recipe applies to the SAS/SHARE provider. Sample code for ADO is included.

ADO Implementation

The following Visual Basic code shows you how to specify the remote server information. This sample uses the Connection object Properties collection to specify individual property values. The server ID is **shr1**.

```
Dim obConnection As New ADO.Connection

obConnection.Provider = "sas.ShareProvider"
obConnection.Properties("Data Source") = "shr1"
obConnection.Properties("Location") = "ShareServer.example.com"
obConnection.Properties("User ID") = "jdoe"
obConnection.Properties("Password") = "djru7"
obConnection.Open
```

Connection information can also be submitted in a single connection string:

```
obConnection.Open "Provider=sas.ShareProvider;Data Source=shr1; _
                  Location=ShareServer.example.com;User ID=jdoe; _
                  Password = djru7"
```

Note: For a step-by-step example and answers to questions that are frequently asked about setting up a SAS/SHARE server, see "SAS/SHARE: Learning to Use" in the *SAS/SHARE User's Guide*.

For an explanation of the properties used in the sample code, see [“SAS/SHARE Provider Properties” on page 28](#).

OLE DB Implementation

OLE DB requires that you use a data source object to connect to a remote SAS/SHARE server. The following table shows how the ADO connection properties that were previously discussed correspond to the OLE DB data source initialization properties.

Table 5.2 OLE DB Data Source Initialization Properties

OLE DB Property Set	OLE DB Property ID	Corresponding ADO Property
DBPROPSET_DBINIT	DBPROP_INIT_DATASOURCE	"Data Source"
DBPROPSET_DBINIT	DBPROP_INIT_LOCATION	"Location"
DBPROPSET_DBINIT	DBPROP_AUTH_USERID	"User ID"
DBPROPSET_DBINIT	DBPROP_AUTH_PASSWORD	"Password"

The OLE DB data source initialization properties use the same syntax as their ADO counterparts. As shown in [“ADO Implementation” on page 41](#), the ADO property values are represented as character strings. In OLE DB, which has only C and C++ language bindings, these property values are implemented as variant type VT_BSTR.

Connecting to a Remote SAS Workspace Server

Goal

You want your application to connect to a remote SAS Workspace Server by using a method that enables you to use ADO objects exclusively. In addition, you want the SAS workspace to persist until the ADO Connection object is closed.

This recipe applies to the IOM provider. This recipe applies only to ADO. Sample code is included.

Note: The connection strings use the Data Source URI format. For more information, see [“What Is the Data Source URI Format?” on page 31](#).

For an explanation of the properties used in the sample code, see [“IOM Provider Properties” on page 28](#).

Implementation

Sample Connection Code for Common SAS 9.2 Server Configurations

The following table provides examples of connection strings for accessing common SAS 9.2 server configurations. In these examples, the SAS Workspace Server name is workspace.example.com and the port number is 8591.

Note: If you are not connecting to SAS 9.2 servers, then you cannot use the sample code in this topic.

Table 5.3 Sample Code for Common SAS 9.2 Server Configurations

Server Configuration	Sample Code
Bridge with host authentication	<pre>Dim connectionString As String connectionString = "Provider=SAS.IOMProvider.9.2; _ Data Source=iom-bridge://workspace.example.com:8591; _ User ID=jdoe;Password=734fi"</pre>
COM with host authentication	<pre>Dim connectionString As String connectionString = "Provider=SAS.IOMProvider.9.2; _ Data Source=iom-com://workspace.example.com; _ User ID=jdoe;Password=734fi"</pre>
Bridge with IWA*	<pre>Dim connectionString As String connectionString = "Provider=SAS.IOMProvider.9.2; _ Data Source=""iom://workspace.example.com:8591; _ Bridge;SECURITYPACKAGE=Negotiate"""</pre>
COM with IWA*	<pre>Dim connectionString As String connectionString = "Provider=SAS.IOMProvider.9.2; _ Data Source=""iom://workspace.example.com; _ COM;SECURITYPACKAGE=Negotiate"""</pre>

* Extra quotation marks are required because of the semicolon used in the "Data Source" property value. If you are not using a connection string, then you do not need the extra quotation marks.

Sample Connection Code for SAS 9.2 Server Configurations That Include a SAS Metadata Server

The following table provides examples of connection strings for accessing common SAS 9.2 server configurations that include a SAS 9.2 Metadata Server. In these examples, the host name is metadata.example.com and the port number is 8561.

Note: If you are not connecting to SAS 9.2 servers, then you cannot use the sample code in this topic.

Table 5.4 Sample Code for Common SAS 9.2 Server Configurations That Include a SAS Metadata Server

Server Configuration	Sample Code
Bridge with a logical server name defined in a SAS Metadata Server and host authentication	<pre>Dim connectionString As String connectionString = "Provider=SAS.IOMProvider.9.2; _ Data Source=iom-name://Workspace Server - Logical Server Name; _ SAS Metadata Location=iom-bridge://metadata.example.com:8561; _ SAS Metadata User ID=metauserid;SAS Metadata Password=metapassword"</pre>

Server Configuration	Sample Code
Bridge with a logical server name defined in a SAS Metadata Server and IWA*	<pre>Dim connectionString As String connectionString = "Provider=SAS.IOMProvider.9.2; _ Data Source=iom-name://Workspace Server - Logical Server Name; _ SAS Metadata Location=""iom://metadata.example.com:8561; _ Bridge;SECURITYPACKAGE=Negotiate"" If the user is not defined in the same authentication domain as the logical server name, then you must also include a line of code that specifies the user ID and password for the remote server. User ID=myuserid;Password=mypassword;</pre>
Logical server name defined in a SAS Metadata Server that is configured via the SAS Integration Technologies configuration utility**	<p>This type of connection does not require any additional metadata server information.</p> <pre>Dim connectionString As String connectionString = "Provider=SAS.IOMProvider.9.2; _ Data Source=iom-name://Workspace Server - Logical Server Name"</pre>

* Extra quotation marks are required because of the semicolon used in the "Data Source" property value. If you are not using a connection string, then you do not need the extra quotation marks.

** The SAS Integration Technologies configuration utility (ITConfig) enables you to create a configuration file that contains information about how to access the metadata server. The provider uses that metadata server in order to resolve logical server names.

Sample Connection Code That Uses the Properties Collection

The following Visual Basic code works with all versions of SAS. In these examples, the SAS Workspace Server name is workspace.example.com and the port number is 8591.

```
Dim obConnection As New ADODB.Connection

obConnection.Provider = "sas.IOMProvider"
obConnection.Properties("Data Source") = "workspace"
obConnection.Properties("SAS Port") = 8591
obConnection.Properties("SAS Machine DNS Name") = "workspace.example.com"
obConnection.Properties("SAS Protocol") = SASObjectManager.ProtocolBridge
obConnection.Properties("User ID") = "jdoe"
obConnection.Properties("Password") = "734fi"
obConnection.Open
```

Note: To use the sample code, you must reference the SASObjectManager Type Library in your Visual Basic project. SASObjectManager Type Library is installed with SAS Integration Technologies.

See Also

[“Connecting to a Remote SAS Workspace Server Using SAS Objects” on page 45](#)

Connecting to a Remote SAS Workspace Server Using SAS Objects

Goal

You want your application to connect to a remote SAS Workspace Server by using SAS objects. This connection method provides more flexibility in creating and managing SAS workspaces than using ADO to create the connection.

This recipe applies to the IOM provider. This recipe applies only to ADO. Sample code is included.

Note: To use the sample code, you must reference these type libraries in your Visual Basic project: Microsoft ActiveX Data Objects Library, the SAS Integrated Object Model (IOM) Type Library, and the SASObjectManager Type Library. The SAS type libraries are installed with SAS Integration Technologies.

Implementation

Sample Code That Uses SAS Objects to Connect to a Remote SAS Workspace Server

The following Visual Basic code works with all versions of SAS. In these examples, the SAS Workspace Server name is workspace.example.com and the port number is 8591.

```
Dim obConnection As New ADODB.Connection
Dim obsAS As SAS.Workspace
Dim obOF As New SASObjectManager.ObjectFactory
Dim obOK As New SASObjectManager.ObjectKeeper
Dim obServerDef As New SASObjectManager.ServerDef

' Use ServerDef attributes to identify the remote server.
obServerDef.Protocol = ProtocolBridge
obServerDef.MachineDNSName = "workspace.example.com"
obServerDef.Port = 8591

' Create a workspace on the identified remote server.
Set obsAS = obOF.CreateObjectByServer("MyServer", True, obServerDef, "johnd", "xyz")

' Add the new workspace object to the ObjectKeeper.
Call obOK.AddObject(1, "WorkspaceObject", obsAS)

' Open a connection to the workspace by using its UniqueIdentifier, which is generated automatically.
obConnection.Open "Provider=sas.iomprovider; Data Source=iom-id://" & obsAS.UniqueIdentifier
:
:
:
obConnection.Close

' Remove the workspace object from the ObjectKeeper.
Call obOK.RemoveObject(obsAS)
```

A Closer Look at the SAS Objects

Here is more information about the SAS objects that are used in the sample code:

- The `ServerDef` object specifies the set of attributes necessary to make a connection to the remote SAS Workspace Server. The sample code assigns values to these attributes: `Protocol`, `MachineDNSName`, and `Port`.
- The following line of code uses the `ObjectFactory CreateObjectByServer` method in order to create a new SAS workspace on the identified server.

```
Set obSAS = obOF.CreateObjectByServer("MyServer", True, obServerDef, "johnd", "xyz")
```

The first parameter is a name that you assign to the created server instance. The second parameter is the Boolean value `True`, which indicates that the application should not attempt to create the workspace until a connection has been established. The third parameter is the name of the `ServerDef` object that contains the connection information. The fourth and fifth parameters are the user ID and password that are required to log on to the remote server.

- The following line of code uses the `ObjectKeeper AddObject` method in order to store the new SAS workspace. Before the IOM provider can reference the workspace, the workspace must be added to the `ObjectKeeper`.

```
Call obOK.AddObject(1, "WorkspaceObject", obSAS)
```

The first two parameters are a unique numeric ID and a unique string. Either value can be used in later code in order to retrieve the workspace object. The third parameter is the name of the workspace object.

Note: For information about working with object variables and creating a workspace, see the *SAS Integration Technologies: Windows Client Developer's Guide*.

See Also

[“Connecting to a Remote SAS Workspace Server” on page 42](#)

Connecting to a Remote SAS OLAP Server

Goal

You want your application to connect to a remote SAS OLAP Server. (Although it is technically possible to make a local connection to a SAS OLAP Server, typically the connection is remote.)

This recipe applies to the OLAP provider. This recipe applies only to ADO. Sample code is included.

Note: The connection strings use the Data Source URI format. For more information, see [“What Is the Data Source URI Format?” on page 31](#).

For an explanation of the properties used in the sample code, see [“OLAP Provider Properties” on page 29](#).

Implementation

Sample Connection Code for Common SAS Server Configurations

The following table provides examples of connection strings that you use to access common SAS server configurations. In these examples, the SAS OLAP Server name is `olap.example.com` and the port number is 8591.

Note: If you are not connecting to SAS servers, then you cannot use the sample code in this topic.

Table 5.5 Sample Code for Common SAS Server Configurations

Server Configuration	Sample Code
Bridge with host authentication	<pre>Dim connectionString As String connectionString = "Provider=sas.OLAPProvider; _ Data Source=iom-bridge://olap.example.com:8591; _ User ID=jdoe;Password=djru7"</pre>
Bridge with IWA*	<pre>Dim connectionString As String connectionString = "Provider=sas.OLAPProvider; _ Data Source=""iom://olap.example.com:8591; _ Bridge;SECURITYPACKAGE=Negotiate"""</pre>

* Extra quotation marks are required because of the semicolon used in the "Data Source" property value. If you are not using a connection string, then you do not need the extra quotation marks.

Sample Connection Code for SAS Server Configurations That Include a SAS Metadata Server

The following table provides examples of connection strings for accessing common SAS server configurations that include a SAS Metadata Server. In these examples, the host name is `metadata.example.com` and the port number is 8561.

Note: If you are not connecting to SAS servers, then you cannot use the sample code in this topic.

Table 5.6 Sample Code for Common SAS Server Configurations That Include a SAS Metadata Server

Server Configuration	Sample Code
Bridge with a logical server name defined in a SAS Metadata Server and host authentication	<pre>Dim connectionString As String connectionString = "Provider=sas.OLAPProvider; _ Data Source=iom-name://OLAP Server - Logical Server Name; _ SAS Metadata Location=iom-bridge://metadata.example.com:8561; _ SAS Metadata User ID=jdoe;SAS Metadata Password=djru7"</pre>

Server Configuration	Sample Code
Bridge with a logical server name defined in a SAS Metadata Server and IWA **	<pre>Dim connectionString As String connectionString = "Provider=SAS.OLAPProvider; _ Data Source=iom-name://OLAP Server - Logical Server Name; _ SAS Metadata Location=""iom://metadata.example.com:8561; _ Bridge;SECURITYPACKAGE=Negotiate"" If the user is not defined in the same authentication domain as the logical server name, then you must also include a line of code that specifies the user ID and password for the remote server. User ID=jdoe;Password=djru7;</pre>
Logical server name defined in a SAS Metadata Server that is configured via the SAS Integration Technologies configuration utility**	<pre>Dim connectionString As String connectionString = "Provider=SAS.OLAPProvider.9.2; _ Data Source=iom-name://OLAP Server - Logical Server Name"</pre>

* Extra quotation marks are required because of the semicolon used in the "Data Source" property value. If you are not using a connection string, then you do not need the extra quotation marks.

** The SAS Integration Technologies configuration utility (ITConfig) enables you to create a configuration file that contains information about how to access the metadata server. The provider uses that metadata server in order to resolve logical server names. This type of connection does not require any additional metadata server information.

Sample Connection Code That Uses a Connection String

The following Visual Basic code works with all versions of SAS. In these examples, the SAS OLAP Server name is `olap.example.com` and the port number is 8591.

```
Dim obConnection As New ADODB.Connection
Dim connectionString As String

connectionString = "Provider=SAS.OLAPProvider.9.2;Data Source=olap.example.com; _
SAS Port=8591;SAS Protocol= SASObjectManager.ProtocolBridge;User ID=myuserid; _
Password=mypassword"
obConnection.Open connectionString
```

Note: To use the sample code, you must reference the SASObjectManager Type Library in your Visual Basic project. SASObjectManager Type Library is installed with SAS Integration Technologies.

Chapter 6

Managing Connections

Supplemental Connection Recipes	49
Prompting Users for Connection Information by Displaying the Data Link Properties Dialog Box	50
Goal	50
Implementation	50
Using a Microsoft Data Link (.udl) File to Provide Persistent Connection Information	56
Goal	56
Implementation	56
Controlling Data Access Permissions with a Connection	57
Goal	57
ADO Implementation	57
OLE DB Implementation	58
Managing File Formats with the Local Provider	60
Goal	60
ADO Implementation	60
OLE DB Implementation	61
Reusing an Existing IOM Workspace	63
Goal	63
Implementation	63
Connecting to a Specific SAS/SHARE Server Version	64
Goal	64
ADO Implementation	64
OLE DB Implementation	64

Supplemental Connection Recipes

This chapter provides sample code that you can use to customize a basic connection. For example, when opening a connection, you can specify a file format and control permissions. You can also write code that displays the Microsoft Data Link dialog box so that users can enter connection information.

Here is a list of the recipes in this chapter:

- [“Prompting Users for Connection Information by Displaying the Data Link Properties Dialog Box” on page 50](#)

- “Using a Microsoft Data Link (.udl) File to Provide Persistent Connection Information ” on page 56
- “Controlling Data Access Permissions with a Connection” on page 57
- “Managing File Formats with the Local Provider” on page 60
- “Reusing an Existing IOM Workspace” on page 63
- “Connecting to a Specific SAS/SHARE Server Version” on page 64

See Also

- “Basic Connection Recipes” on page 37
- “Data Access Recipes” on page 67

Prompting Users for Connection Information by Displaying the Data Link Properties Dialog Box

Goal

You want your application to prompt users for connection information at run time by displaying the Data Link Properties dialog box.

This recipe applies to all the SAS providers that are being used to access SAS servers. This recipe applies only to ADO. Sample code is included.

Implementation

Sample Code That Displays the Data Link Properties Dialog Box

You use the Microsoft Data Link API in order to display the Data Link Properties dialog box, which prompts the user for connection information. The dialog box has property pages that the user completes in order to select a provider and enter connection information. After the user enters the information, an ADO Connection object is returned.

Here are two ways in which you can present the Data Link Properties dialog box to users:

- Method 1: If the data source does not require a password, then you can allow the user to select a provider in addition to entering connection information. You can also specify the provider and allow the user to enter connection information only.
- Method 2: If the data source does require a password, then you write code that specifies the provider. The user enters connection information only in the dialog box.

The following sample code can be used if the data source does not require a password and you want to allow the user to select a provider. This code displays a version of the Data Link Properties dialog box that includes the **Provider** tab.

Note: To use the sample code, you must reference these type libraries in your Visual Basic project: the Microsoft OLE DB Service Component Type Library and the Microsoft ActiveX Data Objects Library.

CAUTION:

Do not use this code if the data source requires a password. If you do, the password is converted to a string of asterisks and the call to Connection.Open fails with an invalid password error.

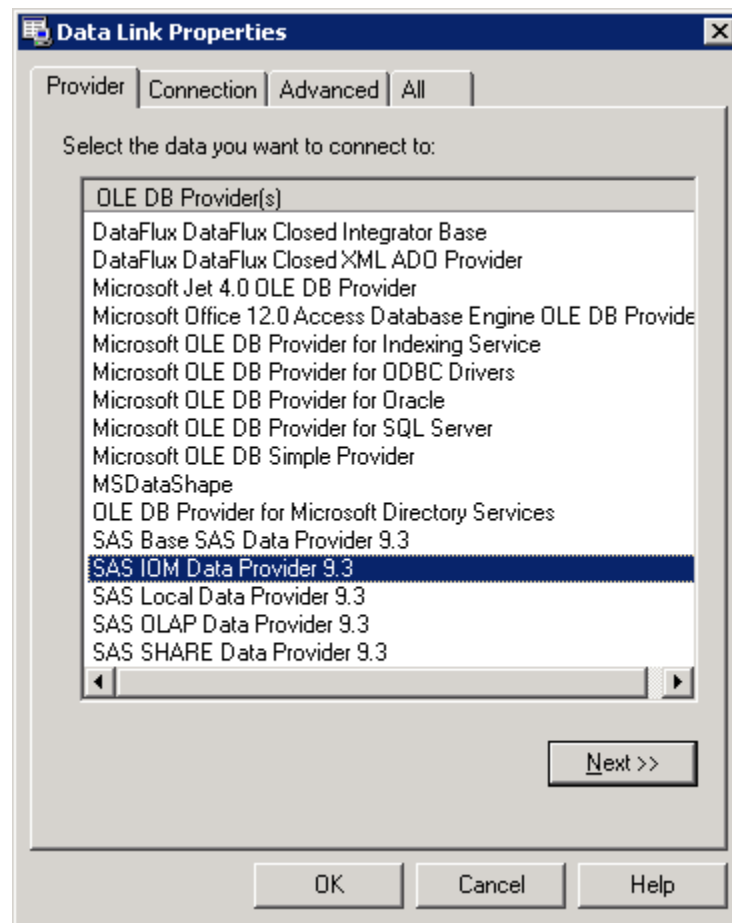
Example Code 6.1 Method 1: A Password Is Not Required and the User Is Prompted to Select a Provider

```
Dim dl As New MSDASC.DataLinks
Dim obConnection As New ADODB.Connection
Dim obRecordset as New ADODB.Recordset

Set obConnection = dl.PromptNew
obConnection.Open

obRecordset.Open "sasuser.MyData", obConnection, adOpenDynamic, adLockOptimistic, adCmdTableDirect
' Operate on obRecordset.
```

Display 6.1 Provider Tab with a List of the SAS Providers for OLE DB



If your data source is secured by a password or if you want to preselect the provider, then you can write code that hides the **Provider** tab from the user. In this case, the user enters the connection information only. Here is sample code that implements this method. The code uses the Connection object Prompt property and includes a provider name (**SAS.IOMProvider**).

Note: To use the sample code, you must reference the Microsoft ActiveX Data Objects Library in your Visual Basic project.

Example Code 6.2 *Method 2: A Password Is Required or the Provider Is Known*

```
Dim obConnection As New ADODB.Connection
Dim obRecordset As New ADODB.Recordset

obConnection.Provider = "SAS.IOMProvider"
obConnection.Properties("Prompt") = adPromptAlways
obConnection.Open

obRecordset.Open "sasuser.MyData", obConnection, adOpenDynamic, adLockOptimistic, adCmdTableDirect
' Operate on obRecordset.
```

Note: For more information about the "Prompt" property, see [“DBPROP_INIT_PROMPT” on page 163](#).

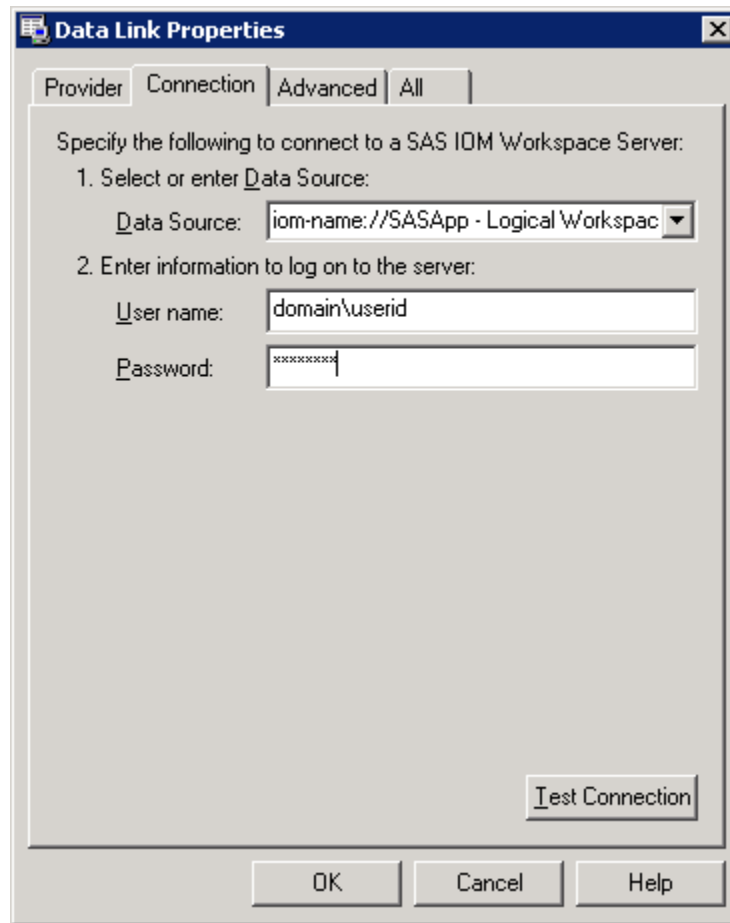
What You Need to Know about Connecting with the IOM and OLAP Providers

If the IOM or OLAP provider is used (either because you specified it or the user selected it), the user can enter connection information on either the **Connection** tab or on the **Advanced** tab. Both tabs are customized specifically for a connection to a SAS Workspace server or SAS OLAP server.

Note: The user must use the **Advanced** tab if the selected provider is defined in a SAS Metadata Server that has not been configured by using the SAS Integration Technologies configuration utility (ITConfig).

Here is the customized version of the **Connection** tab.

Display 6.2 The Customized Connection Tab As It Appears When the IOM Provider Is Selected



SAS provides customized help for completing this tab (see “[Data Link Properties Dialog Box \(Connection Tab\)](#)” on page 269). To access the information, users click the **Help** button.

The following table explains how the fields on the customized **Connection** tab correspond to the ADO and OLE DB properties.

Table 6.1 How the Fields Correspond to ADO and OLE DB Properties

Field Name	ADO Property Name	OLE DB Property Name
Data Source	"Data Source"	DBPROP_INIT_DATASOURCE
User name	"User ID"	DBPROP_AUTH_USERID
Password	"Password"	DBPROP_AUTH_PASSWORD

Here is the customized version of the **Advanced** tab.

Display 6.3 The Advanced Tab As It Appears When the IOM Provider Is Selected

SAS provides customized help for completing this tab (see “[Data Link Properties Dialog Box \(Advanced Tab\)](#)” on page 270). To access the information, users click the **Help** button.

The following table explains how the fields on the **Advanced** tab correspond to the ADO and OLE DB properties.

Table 6.2 How the Fields Correspond to ADO and OLE DB Properties

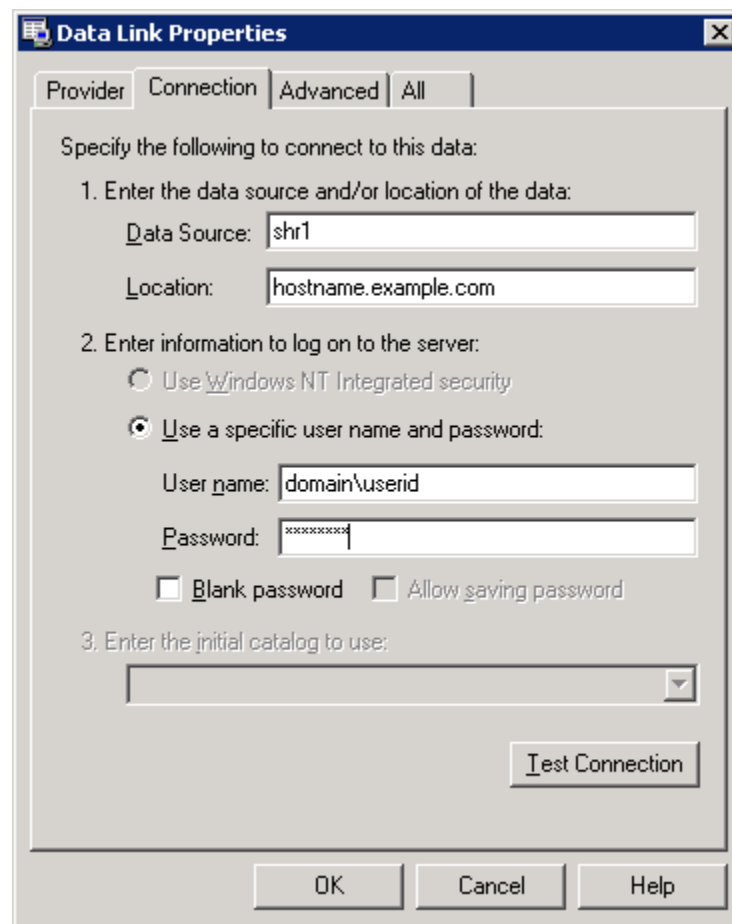
Field Name	ADO Property Name	OLE DB Property Name
SAS Metadata Server Information		
Metadata Server Location	"SAS Metadata Location"	DBPROP_SAS_INIT_METALOCATION
User name	"SAS Metadata User ID"	DBPROP_SAS_INIT_METAUSERID
Password	"SAS Metadata Location"	DBPROP_SAS_INIT_METAPASSWORD
SAS Server Information		
Data Source	"Data Source"	DBPROP_INIT_DATASOURCE

Field Name	ADO Property Name	OLE DB Property Name
Location	"Location"	DBPROP_INIT_LOCATION
User name	"User ID"	DBPROP_AUTH_USERID
Password	"Password"	DBPROP_AUTH_PASSWORD

What You Need to Know about Connecting with the Local, SAS/SHARE, or Base SAS Provider

If the local, SAS/SHARE, or Base SAS provider is being used (either because you specified it or the user selected it), the user enters information on the generic version of the **Connection** tab.

Display 6.4 The Generic Connection Tab for Use with the Local, SAS/SHARE, and Base SAS Providers



For information about completing the generic **Connection** tab, users can click the **Help** button. The standard help content is provided by Microsoft.

The following table explains how the fields on the generic **Connection** tab correspond to the ADO and OLE DB properties. These fields have specifications that are specific to SAS.

Table 6.3 How the Fields Correspond to ADO and OLE DB Properties

Field Name	ADO Property Name	OLE DB Property Name
Data Source Information		
Data Source	"Data Source"	DBPROP_INIT_DATASOURCE
Location	"Location"	DBPROP_INIT_LOCATION
SAS Server Information		
User name	"User ID"	DBPROP_AUTH_USERID
Password	"Password"	DBPROP_AUTH_PASSWORD

Using a Microsoft Data Link (.udl) File to Provide Persistent Connection Information

Goal

You want your application to reference connection information that is stored in a Microsoft Data Link (.udl) file. The .udl file is a special text file that can be used in conjunction with ADO and the Visual Basic DataEnvironment tool in order to initialize an ADO Connection object. (This connection method is similar to the method that is provided for ODBC by the ODBC Administrator.)

Note: DataEnvironment is a Visual Basic 6 GUI tool that can be used to manage static connections to known data sources. To add it to your project, select **Project** ⇒ **Add Data Environment**.

This recipe applies to all the SAS providers. This recipe applies only to ADO. Sample code is included.

Note: To use the sample code, you must reference the Microsoft ActiveX Data Objects Library in your Visual Basic project.

Implementation

Creating a Microsoft Data Link (.udl) File

To create a .udl file:

1. Create an empty text file with the extension .udl.
2. From Windows Explorer, double-click on your new file to open the Data Link Properties dialog box.

Note: The Data Link Properties dialog box is the same interface that the user sees when you create an application that uses the MSDASC.DataLinks object (see [“Prompting Users for Connection Information by Displaying the Data Link Properties Dialog Box”](#) on page 50).

3. In the Data Link Properties dialog box, enter the connection information that is required to open an ADO Connection object.
4. Click **OK** to save the .udl file with the information that you entered.

Sample Code That Shows How to Reference the File

The following sample code shows you how to reference the file in the `ConnectionString` property of a Connection object. It requires that you reference the Microsoft ActiveX Data Objects Library.

```
Dim obConnection As New ADODB.Connection

obConnection.ConnectionString = "File Name=c:\mydir\myfile.udl"
obConnection.Open
```

Note: You can also refer to the .udl file in the Visual Basic DataEnvironment tool.

TIP If you store the .udl file on a server, you can update the .udl file, instead of your application code, when a data source is moved to a new location.

Controlling Data Access Permissions with a Connection

Goal

You want your application to set permissions on your data source.

This recipe applies to all the SAS providers. Sample code for ADO is included.

ADO Implementation

Values That Are Supported for the "Mode" Property

To control the available permissions for modifying data, you use the Connection object "Mode" property. The SAS providers support the following three values for the "Mode" property:

- `adModeRead`
- `adModeReadWrite`
- `adModeShareDenyNone`

The default value for the "Mode" property is `adModeReadWrite` in conjunction with `adModeShareDenyNone`. (You cannot prevent other clients from connecting to the server; therefore `adModeShareDenyNone` is in effect regardless of your lock setting.)

How the Mode Value Affects the Lock Type

The "Mode" property value can affect which lock type is used on open recordsets. For example, the following Visual Basic code determines read-only access (`adModeRead`) before the connection is opened. Recordsets that you open with this connection will use the `adLockReadOnly` lock type, even if you request a different lock type.

```
Dim obConnection As New ADODB.Connection
```

```

obConnection.Provider = "sas.BaseSASProvider"
obConnection.Properties("Data Source") = "sdplsrv"
obConnection.Properties("SAS Executable") = "C:\\Program Files\\SASHome\\SASFoundation\\9.3\\sas.exe"
obConnection.Properties("SAS Parameters") = "-initstmt %sasodbc(sdplsrv) -icon -nologo -notutorialdlg"
obConnection.Properties("SAS Working Directory") = "C:\\Program Files\\SASHome\\SASFoundation\\9.3\\"
obConnection.Mode = adModeRead
obConnection.Open

```

For an explanation of the properties used in the sample code, see “[Base SAS Provider Properties](#)” on page 27.

Note: For more information about lock types, see “[Implementing a Locking Strategy](#)” on page 113 and “[Working with Cursor and Lock Type Combinations](#)” on page 139.

How the Mode Value Is Implemented

The following table provides additional information about how the SAS providers implement "Mode" property values.

CAUTION:

The value that you specify is not always the value that the provider uses.

Table 6.4 How ADO Mode Property Values Are Implemented

Mode Specified	Mode Used	Comment
adModeShareDenyRead, adModeShareDenyWrite, or adModeShareDenyExclusive	adModeShareDenyNone	adModeShareDenyNone is always in effect because you cannot prevent other clients from connecting to a server.
adModeWrite	adModeReadWrite	You cannot open a connection that is limited only to write operations. Reading data is always supported on an open connection. This mode is used in conjunction with adModeShareDenyNone.
adModeRead	adModeRead	If read-only access is set before the connection is opened, recordsets that are opened by using the connection will use the adLockReadOnly lock type, regardless of any other lock type that you might have requested. This mode is used in conjunction with adModeShareDenyNone.

OLE DB Implementation

To get results that are identical to the ADO Connection object "Mode" property, you set a particular DBPROP_INIT_MODE bit mask. The following table lists the one-to-one correspondence between the values of the ADO Connection object ConnectModeEnum constants (the "Mode" property names) and the bit masks defined for the OLE DB DBPROP_INIT_MODE data source property.

Table 6.5 ADO "Mode" Properties and Corresponding DBPROP_INIT_MODE Bit Masks

ConnectModeEnum Constant	DBPROP_INIT_MODE Bit Mask
adModeRead	DB_MODE_READ
adModeWrite	DB_MODE_WRITE
adModeReadWrite	DB_MODE_READWRITE
adModeShareDenyRead	DB_MODE_SHARE_DENY_READ
adModeShareDenyWrite	DB_MODE_SHARE_DENY_WRITE
adModeShareDenyExclusive	DB_MODE_SHARE_DENY_EXCLUSIVE
adModeShareDenyNone	DB_MODE_SHARE_DENY_NONE
adModeUnknown	0

When you set a particular DBPROP_INIT_MODE bit mask value, it has the same effect as when you the ADO Connection object "Mode" property to the corresponding ConnectModeEnum value, as shown in the following table:

Table 6.6 How OLE DB ModeProperty Values Are Implemented

DBPROP_INIT_MODE Bit Mask Set	DBPROP_INIT_MODE Bit Mask Used	Comment
DB_MODE_SHARE_DENY_READ, DB_MODE_SHARE_DENY_WRITE, or DB_MODE_SHARE_DENY_EXCLUSIVE	DB_MODE_SHARE_DENY_NONE	A client cannot prevent other clients from connecting to a server.
DB_MODE_WRITE	DB_MODE_READWRITE	You cannot open a connection that is limited only to write operations. Reading data is always supported.
DB_MODE_READ	DB_MODE_READ	Use of DB_MODE_READ bit without the DB_MODE_WRITE bit restricts open rowsets to read-only access. Read-only rowsets do not implement the IRowsetChange and IRowsetUpdate interfaces.

Managing File Formats with the Local Provider

Goal

You want your application to specify which SAS file format to use when it accesses a data source. You also want to know how to access different file formats simultaneously, and what happens if your application does not specify a file format.

This recipe applies to the local provider. This recipe includes sample code for ADO and OLE DB.

ADO Implementation

Specifying a File Format

To specify which file format that the local provider should use to access a data source, set the "SAS File Format" property on the Connection object. You can use the following property values:

- "V9" (the default) for SAS 9 and earlier (also valid for SAS 8 and SAS 7 data sets)
- "V8" for SAS 8 or SAS 7 data sets
- "V7" for SAS 7 or SAS 8 data sets
- "V6" for SAS 6 data sets
- "XPT" for SAS 5 transport files

For example, the following Visual Basic code could be used to access a SAS Version 6 data set named HRdata that is stored in a directory named `c:\v6data`.

```
Dim obConnection As New ADODB.Connection
Dim obRecordset As New ADODB.Recordset

obConnection.Provider = "sas.LocalProvider"
obConnection.Properties("Data Source") = "c:\v6data"
obConnection.Properties("SAS File Format") = "V6"
obConnection.Open

obRecordset.Open "HRdata", obConnection, adOpenStatic, adLockReadOnly, adCmdTableDirect
```

The code in the next example can read a SAS data set named TestDs1 from a Version 5 SAS transport file named xport1.dat:

```
Dim obConnection As New ADODB.Connection
Dim obRecordset As New ADODB.Recordset

obConnection.Provider = "sas.LocalProvider"
obConnection.Properties("Data Source") = "c:\xptdata\xport1.dat"
obConnection.Properties("SAS File Format") = "XPT"
obConnection.Open

obRecordset.CursorType = adOpenStatic
obRecordset.LockType = adLockReadOnly
obRecordset.Open "TestDs1", obConnection, adOpenStatic, adLockReadOnly, adCmdTableDirect
```

Accessing Different File Formats Simultaneously

You cannot simultaneously access different file formats by using the same Connection object. If you need simultaneous access, do one of the following tasks:

- Store the data sets in different directories.
- Create multiple Connection objects that point to the directory that contains the mixed format data sets. Use the "Data Source" property to specify the directory name. Use the "SAS File Format" property to specify the file format of each data set in the directory.

What Happens If No File Format is Set

If no "SAS File Format" property value is explicitly set and you are not accessing transport files, then the local provider applies the following two rules to set the file format for you:

- If the data sets in the specified directory are all in the same format, then the local provider uses that format.
- If there are no data sets in the directory or if the directory contains data sets with different formats, then the "SAS Default File Format" property value ("V9") is used.

The two rules for handling those cases when the file format is not explicitly set do not apply to transport files. To access transport files, enter "XPRT" as the "SAS File Format" property value.

TIP Even though you can store transport files in the same directory as SAS data sets, it is still a good practice to keep transport files in a separate directory.

Setting the Data Source Property for Transport Files

When you access SAS data sets, the Connection object "Data Source" property is the path to the directory that contains the data sets. When you are reading transport files, however, you should set the Connection object "Data Source" property to both the path and to the filename of the transport file.

The local provider can read data sets in a transport file even if there is more than one data set in the transport file. Set the first parameter of the Recordset object's Open method to the data set name, as shown in this line of code:

```
obRecordset.Open "TestDs1", obConnection, adOpenStatic, adLockReadOnly, adCmdTableDirect
```

OLE DB Implementation

Specifying a File Format

The OLE DB implementation is similar to the code used for ADO; however, for the OLE DB interface, you use these properties:

- DBPROP_INIT_DATASOURCE (corresponding to the ADO property "Data Source").
- DBPROP_SAS_INIT_FILEFORMAT (corresponding to the ADO property "SAS File Format"). The property values for DBPROP_SAS_INIT_FILEFORMAT are the same values that are used for "SAS File Format" (see ["Specifying a File Format" on page 60](#)).

The following code shows how to use the local provider to set up a data source object that accesses SAS Version 6 files in a directory named `c:\v6data`:

```

CLSID clsid;
IUnknown * pDSO;
// Turn the ProgID into a CLSID value
CLSIDFromProgID( "SAS.LocalProvider", &clsid );
// With the CLSID, create an instance of this provider's data source object
CoCreateInstance( clsid, // class identifier
NULL, // no outer unknown (that is, no aggregation)
CLSCTX_INPROC_SERVER, // all providers run in process
IID_IUnknown, // the id of the interface we want on our new object
(LPVOID *) &pDSO ); // address of interface pointer returned

DBPROPSET rgPropSet [2];
DBPROP PropA;
DBPROP PropB;
IDBProperties * pIDBProperties;
IDBInitialize *pIDBInitialize;

PropA.dwPropertyID = DBPROP_INIT_DATASOURCE;
PropA.dwOptions = DBPROPOPTIONS_REQUIRED;
PropA.colid = DB_NULLID;
PropA.vValue.vt = VT_BSTR;
PropA.vValue.pbstrVal = SysAllocString( "c:\v6data" );
PropB.dwPropertyID = DBPROP_SAS_INIT_FILEFORMAT;
PropB.dwOptions = DBPROPOPTIONS_REQUIRED;
PropB.colid = DB_NULLID;
PropB.vValue.vt = VT_BSTR;
PropB.vValue.pbstrVal = SysAllocString( "V6" );

rgPropSet [0].cProperties = 1;
rgPropSet [0].guidPropertySet = DBPROPSET_DBINIT;
rgPropSet [0].rgProperties = &PropA
rgPropSet [1].cProperties = 1;
rgPropSet [1].guidPropertySet = DBPROPSET_SAS_DBINIT;
rgPropSet [1].rgProperties = &PropB
pDSOUnk->QueryInterface( IID_IDBProperties, &pIDBProperties );
pIDBProperties->SetProperties( 2, rgPropset );
pDSOUnk->QueryInterface( IID_IDBInitialize, &pIDBInitialize );
pIDBInitialize->Initialize();

```

Accessing Different File Formats Simultaneously

You cannot simultaneously access different file formats by using the same data source object. If you need simultaneous access, do one of the following tasks:

- Store the data sets in different directories.
- Create multiple data source objects that point to the directory that contains the mixed format data sets. Use the `DBPROP_INIT_DATASOURCE` property to specify the directory name. Use the `DBPROP_SAS_INIT_FILEFORMAT` property to specify the file format of each data set in the directory.

What Happens if No File Format is Set

If the `DBPROP_SAS_INIT_FILEFORMAT` property value is not explicitly set and if you are not accessing transport files, the local provider applies the following two rules to set the file format for you:

- If the data sets in the specified directory are all in the same format, then the local provider applies that format.
- If there are no data sets in the directory or if the directory contains data sets with different formats, then the DBPROP_SAS_DEFAULTFILEFORMAT property value ("V9") is used.

The two rules for handling those cases when the file format is not explicitly set do not apply to transport files. To access transport files, enter **"XPT"** as the DBPROP_SAS_INIT_FILEFORMAT property value.

TIP Even though you can store transport files in the same directory as SAS data sets, it is still a good practice to keep transport files in a separate directory.

Reusing an Existing IOM Workspace

Goal

You want your application to reuse an existing IOM Workspace.

This recipe applies to the IOM provider. This recipe applies only to ADO. Sample code is included.

Note: To use the sample code, you must reference the SAS Integrated Object Model (IOM) Library in your Visual Basic project.

Implementation

You can use the "SAS Workspace ID" property to associate a running SAS workspace object with an ADO Connection object. "SAS Workspace ID" is a customized connection property. It indicates that you want to use a workspace that you have already created. You set this property on a Connection object after you set the Connection object Provider property and before you invoke the Open method. The following Visual Basic code shows you how to use this method:

```
Dim obConnection As New ADODB.Connection

obConnection.Provider = "sas.IOMProvider"
obConnection.Properties("SAS Workspace ID") = obSAS.UniqueIdentifier
' Set other connection properties as needed.
obConnection.Open
```

In the sample code, **obSAS** is a SAS.Workspace object that has been previously created. The workspace provides the same set of resources and facilities as an interactive or batch SAS session. The Workspace Object class is part of SAS Integration Technologies. The UniqueIdentifier workspace property instructs the IOM provider to look up an existing workspace and to establish communication with it.

Note: For information about working with object variables and creating a workspace, see the *SAS Integration Technologies: Windows Client Developer's Guide*.

Connecting to a Specific SAS/SHARE Server Version

Goal

You want your application to explicitly request access to a Version 7 or Version 8 SAS/SHARE server. By default, the SAS/SHARE provider uses the SAS 9 server access method.

This recipe applies to the SAS/SHARE provider. Sample code for ADO is included.

ADO Implementation

To specify a server version, you provide a value for the "SAS Server Release" property. You set the property on the Connection object because the connection is bound to a specific server.

Values for the "SAS Server Release" property are integer numbers that correspond to the version number of the server that is being accessed: 7, 8 (valid for both Version 7 and Version 8 servers), or 9 (the default).

The following Visual Basic code specifies the Version 7 SAS/SHARE server:

```
Dim obConnection As New ADODB.Connection

obConnection.Provider = "sas.ShareProvider"
obConnection.Properties("SAS Server Release") = 7
obConnection.Open
```

OLE DB Implementation

Use the DBPROP_SAS_SERVERRELEASE property to specify the access method. This property is a member of the DBPROPSET_SAS_DBINIT property set.

Part 4

Data Management Recipes

<i>Chapter 7</i>	
Accessing Specific or Protected Data	67
<i>Chapter 8</i>	
Creating, Subsetting, and Deleting Data Sets	79
<i>Chapter 9</i>	
Specifying How to Display Data	91
<i>Chapter 10</i>	
Managing Missing Values	101
<i>Chapter 11</i>	
Managing Updates	111

Chapter 7

Accessing Specific or Protected Data

Data Access Recipes	67
Identifying a Data Set and Returning Results	68
Goal	68
ADO Implementation	68
Specifying a Libref to Use with the IOM Provider	69
Goal	69
Implementation	69
Opening a Password-Protected Data Set	71
Goal	71
ADO Implementation	71
OLE DB Implementation	72
Accessing Third-Party Data through SAS/ACCESS Engines	72
Goal	72
Implementation	72
Displaying Metadata That Is Specific to SAS Data Sets	73
Goal	73
ADO Implementation	74
OLE DB Implementation	76
Reading SAS OLAP Cubes	76
Goal	76
Implementation	76

Data Access Recipes

This chapter provides sample code that you can use to perform tasks related to accessing and reading data after the connection is established. These tasks are described here because they have some aspects that are specific to SAS.

Note: To perform a data-related task that is not included in this chapter, see the *OLE DB Programmer's Reference and Data Access SDK*.

Here is a list of the recipes in this chapter:

- “Identifying a Data Set and Returning Results” on page 68
- “Specifying a Libref to Use with the IOM Provider” on page 69
- “Opening a Password-Protected Data Set” on page 71

- “Accessing Third-Party Data through SAS/ACCESS Engines” on page 72
- “Displaying Metadata That Is Specific to SAS Data Sets” on page 73
- “Reading SAS OLAP Cubes” on page 76

TIP To use the recipes in this chapter you need an open ADO Connection object. For more information, see “Basic Connection Recipes” on page 37.

See Also

“Supplemental Connection Recipes” on page 49

Identifying a Data Set and Returning Results

Goal

You want your application to open a specific data set and return either static or dynamic results. Your options depend on which SAS provider you are using.

This recipe applies to the local, SAS/SHARE, IOM, and Base SAS providers. Sample code for ADO is included.

ADO Implementation

Sample Code for Identifying Data Sets and Returning Results

The following table includes sample code that is used identify data sets and return results. Assume that a Connection object named obConnection is already open.

Table 7.1 Sample Code for Identifying Data Sets

Provider	Comment and Sample Code
local	Specify the data set name. In the sample code, the data set name is family . <pre>obRecordset.Open "family", obConnection, adOpenStatic, adLockReadOnly, _ adCmdTableDirect</pre>
IOM, SAS/SHARE, and Base SAS	Use SAS <i>libname.memname</i> notation. In the sample code, the data set name is mylib.cities . <pre>obRecordset.Open "mylib.cities", obConnection, adOpenStatic, _ adLockReadOnly, adCmdTableDirect</pre>

The value of the CommandType property determines whether the recordset can be modified.

Table 7.2 How the CommandType Value Affects Results

Value of the CommandType Property	Result
adCmdTableDirect	A dynamic open of a data set that can be modified. Command type adCmdTableDirect is used in the sample code and can be used with all of the providers.
adCmdTable	A static result set from a query that cannot be modified. Command type adCmdTable can be used only with the IOM, SAS/SHARE, and Base SAS providers because adCmdTable executes an SQL query in the form SELECT * FROM libname.memname . The local provider does not support SQL processing.

A Closer Look at Defining the SAS Library for the Server

SAS organizes tables in libraries. Before you can use a SAS library, you must tell SAS where it is. One way to identify the library is to use a libref, which is a short name (or alias) for the full physical name of the library. Before you can use the *libname.memname* notation, the libref that contains the data set must be defined for the server.

- For the SAS/SHARE server, the libref is assigned when the server is started.
- For a local installation of Base SAS, you assign the libref when you specify the "SAS Parameters" property in the connection code. The value of the "SAS Parameters" property contains the command line that includes code to start the server. For more information about the "SAS Parameters" property, see [“Connecting to Local Data \(Single-User Server\)” on page 39](#).
- For the SAS Workspace Server, you can use ADO to assign a libref by executing a LIBNAME statement through an ADO Command object. Or you can use the IOM DataService class to assign a libref. For more information, see [“Specifying a Libref to Use with the IOM Provider” on page 69](#).

Specifying a Libref to Use with the IOM Provider

Goal

You want to use the IOM provider to open or create a table, so you need to know how to specify (assign) a libref.

This recipe applies to the IOM provider. Sample code for ADO is included.

Implementation

Two Methods for Assigning a Libref

SAS organizes tables in libraries. Before you can use a SAS library, you must tell SAS where it is. One way to identify the library is to use a libref, which is a short name (or alias) for the full physical name of the library. Here are two ways that you can assign a libref:

- You can use ADO to assign a libref by executing a LIBNAME statement through an ADO Command object.

- You can use the IOM DataService class to assign a libref.

Both techniques assign a libref for the duration of the SAS session or until the libref is unassigned. When choosing between the two methods, consider the information in the following table:

Table 7.3 How to Choose Between the Two Methods

Your Situation or Preference	Recommended Method
You are comfortable using the SAS LIBNAME statement.	Use ADO.
You are not using SAS objects to connect to your server.	Use ADO.
You are using SAS objects to connect to your server.	Use the IOM DataService class.
You want to know whether the libref was successfully assigned.	Use the IOM DataService class (because the AssignLibref method has a return code).
You want to take advantage of other methods and attributes on the Libref object.	Use the IOM DataService class.

Using ADO to Assign a Libref

You can use ADO to assign a libref by executing a LIBNAME statement through an ADO Command object. The following Visual Basic code shows how this task is done.

```
' obConnection is an open Connection object.
Dim obRecordset As New ADODB.Recordset
Dim obCommand As New ADODB.Command

obCommand.ActiveConnection = obConnection

'Assign a libref by executing a LIBNAME statement.
obCommand.CommandType = adCmdText
obCommand.CommandText = "libname mylib 'c:\census\data'"
obCommand.Execute

'Open a data set in the assigned library.
obRecordset.Open "mylib.cities", obConnection, adOpenStatic, adLockReadOnly, adCmdTableDirect
```

Using the IOM DataService Class to Assign a Libref

You can use the IOM DataService class to assign a libref. The following Visual Basic code shows how this task is done.

Note: To use the sample code, you must reference these libraries in your Visual Basic project: SAS Integrated Object Model (IOM) Type Library and the SASObjectManager Type Library. These libraries are installed with SAS Integration Technologies.

```
Dim obObjectFactory As New SASObjectManager.ObjectFactory

Dim obConnection As New ADODB.Connection
Dim obRecordset As New ADODB.Recordset
```

```

Dim obsSAS As SAS.Workspace
Dim obLibRef As SAS.Libref

' Use the SAS Object Manager to establish a SAS workspace object.
set obsSAS = obObjectFactory.CreateObjectByServer("MyServer", True, Nothing, "", "")

Dim obObjectKeeper As New SASObjectManager.ObjectKeeper

obObjectKeeper.AddObject(1, "MyServer", obsSAS)

' Call the AssignLibref method in order to assign a libref.
Set obLibRef = obsSAS.DataService.AssignLibref("mylib", "", "c:\census\data", "")

' Open a connection.
obConnection.Open "Provider=sas.IOMProvider;SAS Workspace ID=" & obsSAS.UniqueIdentifier

' Open a data set in the assigned library.
obRecordset.Open "mylib.cities", obConnection, adOpenStatic, adLockReadOnly, adCmdTableDirect

```

In the sample code, **obsSAS**, **obLibRef**, and **obObjectFactory** are all IOM objects. For more information about the SAS object hierarchy, see the *SAS Integration Technologies: Windows Client Developer's Guide*.

Opening a Password-Protected Data Set

Goal

You want your application to open a password-protected data set.

This recipe applies to the local, SAS/SHARE, IOM, and Base SAS providers. Sample code for ADO is included.

SAS supports three types of data set passwords: READ, WRITE, and ALTER. SAS also supports a PW= data set option that assigns the same password for each level of protection. To access a data set that is protected with the PW= data set option, set each ADO or OLE DB property that provides the level of access that you need.

Note: For complete information about SAS data set passwords, see *SAS Language Reference: Concepts*.

ADO Implementation

The SAS providers implement three ADO Recordset object properties that correspond to each of the SAS password types.

Table 7.4 Recordset Properties and Corresponding SAS Password Types

Recordset Property Name	SAS Data Set Password Types
"SAS Read Password"	READ=
"SAS Write Password"	WRITE=

Recordset Property Name	SAS Data Set Password Types
"SAS Alter Password"	ALTER=

The following sample code uses the "SAS Read Password" property to open a read-protected data set.

```
' obConnection is an open Connection object.
Dim obRecordset As New ADODB.Recordset

obRecordset.ActiveConnection = obConnection
obRecordset.Properties("SAS Read Password") = "gurt96"

' The second parameter on the Open method must remain empty.
obRecordset.Open source, , adOpenStatic, adLockReadOnly, adCmdTableDirect
```

OLE DB Implementation

The SAS providers implement three OLE DB rowset properties that correspond to each of the SAS password types. The OLE DB properties are part of the DBPROP_SAS_ROWSET customized property set.

Table 7.5 Rowset Properties and Corresponding SAS Password Types

Rowset Property Name	SAS Data Set Password Types
"DBPROP_SAS_READPASSWORD"	READ=
"DBPROP_SAS_WRITEPASSWORD"	WRITE=
"DBPROP_SAS_ALTERPASSWORD"	ALTER=

Accessing Third-Party Data through SAS/ACCESS Engines

Goal

You want to access third-party data stores that are available through your licensed SAS/ACCESS engines.

This recipe applies to the SAS/SHARE, IOM, and Base SAS providers. Sample code for ADO is included.

Implementation

Assigning a Libref for a SAS/ACCESS Engine

An engine is a component of SAS software that is used to read from or write to a source of data. There are several types of SAS engines, including engines that SAS/ACCESS

software uses to connect to a variety of data sources other than Base SAS. To access third-party data that is available through your licensed SAS/ACCESS engines, you must assign a libref for the specific SAS/ACCESS engine. How you assign the libref depends on the provider.

- The IOM provider is the only provider that can be used to directly assign a libref, as illustrated by the sample code.
- If you are using the SAS/SHARE provider, the libref must have been assigned when the SAS/SHARE server was started.
- If you are using the Base SAS provider, you must specify the libref in the start-up script that is used by the provider to start a SAS session (see [“Connecting to Local Data \(Single-User Server\)”](#) on page 39).

Sample Code for Accessing Third-Party Data Using the IOM Provider

The following sample Visual Basic code uses the ADO Command and Recordset objects in order to access Oracle data. The record set is opened for read-only access as indicated by `adLockReadOnly`.

```
' obConnection is an open Connection object.
Dim obRecordset As New ADO.Recordset
Dim obCommand As New ADO.Command

obCommand.ActiveConnection = obConnection

'Assign a libref by executing a LIBNAME statement that identifies the SAS/ACCESS engine.
obCommand.CommandType = adCmdText
obCommand.CommandText = "libname mylib oracle user=todd password=king path=oraclev7;"
obCommand.Execute

'Open the data set for read-only access.
obRecordset.Open "mylib.dept", obConnection, adOpenStatic, adLockReadOnly, adCmdTableDirect
```

Note: The syntax that you use in your LIBNAME statement depends on the SAS/ACCESS engine that you are using and on your operating environment. SAS/ACCESS engines are implemented differently in different operating environments. See the documentation for your DBMS for more information.

Note: If you are writing directly to the OLE DB interface, use OLE DB rowset methods in order to read the SAS/ACCESS data set. To open the rowset for read-only access, set the DBPROP_IRowsetChange property to `False`.

See Also

[“Specifying a Libref to Use with the IOM Provider”](#) on page 69

Displaying Metadata That Is Specific to SAS Data Sets

Goal

You want your application to display metadata that is specific to SAS data sets.

Sample code for ADO is included.

ADO Implementation

How Metadata Is Exposed

Available metadata for a SAS data set includes persisted formats and informats, number of records, and whether an index exists. To extract the metadata, you use extensions to the COLUMNS and TABLES schema rowsets.

Note: For more information, see “COLUMNS Schema Rowset” on page 232 and “TABLES Schema Rowset” on page 250.

Sample Code for Displaying Information about Formats and Informats

The following Visual Basic code shows how to display information about SAS formats and informats that are persisted on a data set named Shoes in the SASUSER library.

This sample code applies to the local, SAS/SHARE, IOM, and Base SAS providers.

```
' obConnection is an open Connection object.
Dim obRecordset As New ADO.Recordset

' Get schema information for Sasuser.Shoes.
Set obRecordset = obConnection.OpenSchema(adSchemaColumns,
    Array(Empty, Empty, "Sasuser.Shoes"))

' Display fields pertaining to SAS formats and informats.
Do Until obRecordset.EOF
    Debug.Print "Table name: " & obRecordset!TABLE_NAME & vbCr & _
    "Column Name: " & obRecordset!COLUMN_NAME & vbCr & _
    "Format name: " & obRecordset!FORMAT_NAME & vbCr & _
    "Format length: " & obRecordset!FORMAT_LENGTH & vbCr & _
    "Informat name: " & obRecordset!INFORMAT_NAME & vbCr & _
    "Informat length: " & obRecordset!INFORMAT_LENGTH
    obRecordset.MoveNext
Loop
```

A Closer Look at the Parameters for the OpenSchema Method

The following line of code specifies the parameters for the OpenSchema Method:

```
Set obRecordset= obConnection.OpenSchema(adSchemaColumns,
    Array(Empty, Empty, "SASUSER.SHOES"))
```

- **For the first parameter, which is Query Type**, the sample code specifies adSchemaColumns. adSchemaColumns returns information for all columns on all data sets in the open connection.
- **For the second parameter, which is Criteria**, the sample code passes in an array that limits the returned information to just SASUSER.SHOES columns. To achieve this result, the array specifies a nonempty value for the TABLE_NAME, which is the third constraint available for adSchemaColumns. (adSchemaColumns has four available constraints: TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME.)

TIP To return metadata for all data sets in the open connection, write similar code and specify adSchemaTables (instead of adSchemaColumns) as the value of the QueryType parameter.

Sample Code for Displaying SAS Data Set Information

The following C# code shows how to display information about the SAS data set type, label, and encoding for the data set named Shoes in the SASUSER library.

This code applies to the local provider.

```
// obConnection is an open Connection object.
Recordset rs = new Recordset();
try {
    rs.ActiveConnection = obConnection;
    rs.Open("shoes", Type.Missing, CursorTypeEnum.adOpenForwardOnly,
        LockTypeEnum.adLockReadOnly,
        (int)CommandTypeEnum.adCmdTableDirect);

    // Recordset Properties for the input table
    Console.WriteLine("SAS Data Set Type property: " +
        rs.Properties["SAS Data Set Type"].Value);

    Console.WriteLine("SAS Data Set Label property: " +
        rs.Properties["SAS Data Set Label"].Value);

    Console.WriteLine("SAS Data Set Encoding property: " +
        rs.Properties["SAS Data Set Encoding"].Value);

    Console.WriteLine("SAS Data Set Windows Code Page property: " +
        rs.Properties["SAS Data Set Windows Code Page"].Value);

    rs.Close();
} catch (Exception e) {
    // exception handling
}
```

The following C# code produces similar results to the previous example, but accesses the Schema Rowset Tables. The code applies to the local provider and SAS/SHARE provider.

```
// obConnection is an open Connection object.
Recordset rs = new Recordset();
try {
    rs.ActiveConnection = obConnection;
    object[] restrictions = new object[] { null, null, "shoes" };
    rs = obConnection.OpenSchema( SchemaEnum.adSchemaTables,
        restrictions, Type.Missing);

    // Fields of the Tables schema rowset for the input table
    Console.WriteLine("SAS_DATASET_TYPE field:" +
        rs.Fields["SAS_DATASET_TYPE"].Value);

    Console.WriteLine("SAS_DATASET_LABEL field:" +
        rs.Fields["SAS_DATASET_LABEL"].Value);

    Console.WriteLine("SAS_DATASET_ENCODING field:" +
        rs.Fields["SAS_DATASET_ENCODING"].Value);

    Console.WriteLine("SAS_DATASET_WINDOWS_CODEPAGE field: " +
        rs.Fields["SAS_DATASET_WINDOWS_CODEPAGE"].Value);

    rs.Close();
}
```

```

    } catch (Exception e) {
        // exception handling
    }

```

OLE DB Implementation

When you are programming directly to the OLE DB interface, you can obtain SAS metadata through schema rowset extensions or through the custom OLE DB rowset interfaces listed below. Schema rowset extensions and these custom interfaces return the same information.

- “ISASColumnsInfo Custom Interface” on page 218
- “ISASDataSetInfo Custom Interface” on page 220
- “ISASDataSetInfo90 Custom Interface” on page 223

Reading SAS OLAP Cubes

Goal

You want your application to read SAS OLAP cubes.

This recipe applies to the OLAP provider. This recipe applies only to ADO MD, which is an extension to ADO that enables you to read multidimensional schema, query cubes, and retrieve the results. Sample code is included.

Note: This recipe requires that you reference these type libraries in your Visual Basic project: the Microsoft ActiveX Data Objects Library and the Microsoft ActiveX Data Objects (Multidimensional) Library.

Implementation

Opening an ADO MD Cellset Object

The following Visual Basic code shows you how to open an ADO MD Cellset object. You first specify an MDX query and then execute it on a SAS OLAP Server.

```

' obConnection is an open Connection object.
Dim obCellset As ADOMD.Cellset

Set obCellset.ActiveConnection = obConnection

obCellset.Source = "SELECT" & _
    " {[dealers].[dealer].members} ON COLUMNS," & _
    " {[cars].members} ON ROWS" & _
    " FROM MDDBCARS" & _
    " WHERE ([measures].[SALES_SUM])"

obCellset.Open

```

Displaying the Results of the MDX Query

After the Cellset object is open, you can display the results of the MDX query. The following Visual Basic code shows how this task is done. The sample code prints the data to the Visual Basic Immediate window by calling Debug.Print.

```

' Print the cellset axis structure and count the number of cells.
Dim cCells As Long
Dim obAxis As ADOMD.Axis
Dim obPosition As ADOMD.Position
Dim obMember As ADOMD.Member

cCells = 1
For Each obAxis In obCellset.Axes
    Debug.Print obAxis.Name & ", cells on axis: " & obAxis.Positions.Count
    cCells = cCells * obAxis.Positions.Count
    For Each obPosition In obAxis.Positions
        For Each obMember In obPosition.Members
            Debug.Print vbTab & obMember.Name & ": " & obMember.Caption
        Next
    Next
Next

' Print the slicer axis structure.
Debug.Print vbNewLine & "Filter/Slicer Axis:"
For Each obPosition In obCellset.FilterAxis.Positions
    For Each obMember In obPosition.Members
        Debug.Print vbTab & obMember.Name & ": " & obMember.Caption
    Next
Next
Debug.Print ""

If cCells = 1 Then
    Debug.Print "1 cell returned." & vbNewLine
Else
    Debug.Print cCells & " cells returned." & vbNewLine
End If

' Print the cell values by ordinal.
Dim i As Long, j As Long
Dim obCell As ADOMD.Cell
Dim strPositions As String

For i = 0 To cCells - 1
    Set obCell = obCellset(i)
    strPositions = "Cell " & i & " "

    ' Print the position array for this cell.
    ' (Example: Cell 0 would be Cell(0,0,0) in a 3-d array).
    strPositions = strPositions & "("
    For j = 0 To obCell.Positions.Count - 1
        If j > 0 Then strPositions = strPositions & ", "
        strPositions = strPositions & obCell.Positions(j).Ordinal
    Next
    strPositions = strPositions & "): "

    ' Retrieve the cell value and check for NULL.

```

```
    If IsNull(obCell.value) Then
        strPositions = strPositions & "-null-"
    Else
        strPositions = strPositions & obCell.value
    End If
    Debug.Print strPositions
Next

' Clean up and exit.
obCellset.Close
Set obCellset = Nothing
Set obCell = Nothing
Set obAxis = Nothing
Set obPosition = Nothing
Set obMember = Nothing
```

Chapter 8

Creating, Subsetting, and Deleting Data Sets

Creating, Subsetting, and Deleting Data Sets Recipes	79
Creating and Deleting Data Sets	80
Goal	80
Implementation	80
Subsetting Data Sets for Read-Only Sequential Access	85
Goal	85
ADO Implementation	85
OLE DB Implementation	86
Subsetting Data Sets for Random and Update Access	89
Goal	89
Implementation	90

Creating, Subsetting, and Deleting Data Sets Recipes

This chapter provides sample code that you can use to perform tasks related to creating and deleting data sets and subsetting returned data. These tasks are performed after the connection is established. These tasks are included because they have some aspects that are specific to SAS.

Note: To perform a data-related task that is not included in this chapter, see the *OLE DB Programmer's Reference and Data Access SDK*.

Here is a list of the recipes in this chapter:

- “Creating and Deleting Data Sets” on page 80
- “Subsetting Data Sets for Read-Only Sequential Access” on page 85
- “Subsetting Data Sets for Random and Update Access” on page 89

TIP To use the recipes in this chapter, you need an open ADO Connection object. For more information, see “Basic Connection Recipes” on page 37.

See Also

“Supplemental Connection Recipes” on page 49

Creating and Deleting Data Sets

Goal

You want your application to create and delete data sets.

This recipe applies to the SAS/SHARE, IOM, and Base SAS providers. This recipe includes sample code for ADO and OLE DB.

Implementation

Three Methods for Creating and Deleting Data Sets

Here are three ways that you can create and delete data sets.

- You can use SQL CREATE TABLE and DROP TABLE statements. This method can be used with ADO and OLE DB consumers.
- You can use Microsoft ActiveX Data Object Extensions for DDL and Security (ADOX). This method is used with ADO consumers.
- You can use the ITableDefinition interface. This method is used with OLE DB consumers.

Before deciding which method to use, review the information in the following table.

Table 8.1 How to Select a Method

Your Situation or Preference	Recommended Method
You want to use ADO.	Use SQL or ADOX.
You want to use OLE DB.	Use SQL or the ITableDefinition interface.

Using SQL Commands with an ADO or OLE DB Consumer

You can pass SQL commands to the providers by using either ADO or OLE DB. The CREATE TABLE statement is used to create a data set. The DROP TABLE statement is used to delete a data set. For example, the statement shown below creates a data set named **newtable** in the **sasuser** library with three columns: a numeric column named **i**, a character column of length 40 named **name**, and a numeric column named **age**.

```
create table sasuser.newtable ( i num, name char(40), age num );
```

The following SQL statement creates a data set named **newtable** in the **sasuser** library that is a copy of the data set **oldtable** in the **sasuser** library.

```
create table sasuser.newtable as select * from sasuser.oldtable;
```

Note: For more examples, see “[Subsetting Data Sets for Read-Only Sequential Access](#)” on page 85. For more information about SQL syntax, see the documentation for PROC SQL in the *Base SAS Procedures Guide*.

Using ADOX with an ADO Consumer

If you are creating an ADO consumer, you can use ADOX to create new data sets and delete existing ones.

Note: To use ADOX, you must add the following two library references to your Visual Basic project: Microsoft ActiveX Data Objects Library and the Microsoft ADO Extension for DDL and Security.

The following Visual Basic code shows you how to use ADOX to create a data set, append to a data set, and delete a data set.

```
' obConnection is an open Connection object.

Dim cat As New ADOX.Catalog
Dim table As New ADOX.table
Dim tablename As String

tablename = "sasuser.newtable"

' Create the data set.
table.Name = tablename
table.Columns.Append "i", adDouble, 0
table.Columns.Append "name", adChar, 40
table.Columns.Append "age", adDouble, 0

' Append the new data set to the collection of data sets.
' Call the provider to create the data set on disk.
cat.ActiveConnection = obConnection
cat.Tables.Append table

' Open a Recordset object and add rows to the new data set.

' Delete the new table.
cat.Tables.Delete tablename
```

Using the ITableDefinition Interface with an OLE DB Consumer

OLE DB consumers can use the ITableDefinition interface that is exposed by the Session object to create and delete data sets. The following C++ code shows how this task is done.

The code uses the IOM provider to create and delete a table named **newtable** in the **sasuser** library. The sample first creates the table with three columns: a numeric column named **i**, a character column of length 40 named **name**, and a numeric column named **age**. The code then deletes that same table.

Note: For more information, see [“Data Set Management Using the ITableDefinition Interface”](#) on page 225.

```
#include atlbase.h
#include comdef.h
#include comutil.h
#include oledb.h
#include atldbcli.h

#include <iostream>
#include <iomanip>
using std::cout;
using std::hex;
```

```

using std::setw;
using std::setfill;
using std::right;
using std::endl;

#define FAIL_IF( hr ) issue_if_failed( __FILE__, __LINE__, hr )
inline void issue_if_failed( char* file, ULONG line, HRESULT hr )
{
    /* This method throws an error if hr is a failure. */
    if( FAILED( hr ) )
    {
        ATLTRACE( %s(%d): Failure 0x%X\n, (char*)file, line, hr );
        _com_issue_error( hr );
    }
}

HRESULT InstantiateProvider( CComPtr<IUnknown>& spUnkDataSrc )
{
    /* spUnkDataSrc contains a NULL IUnknown pointer. */
    /* This method instantiates the IOM Provider Data Source object and */
    /* stores a pointer to its IUnknown interface in spUnkDataSrc. */
    CLSID clsid;

    FAIL_IF( CLSIDFromProgID( L"SAS.IOMProvider", &clsid ) );
    FAIL_IF( CoCreateInstance( clsid, NULL, CLSCTX_INPROC_SERVER, IID_IUnknown, (void**)&spUnkDataSrc ) );

    return S_OK;
}

HRESULT InitializeProvider( CComPtr<IUnknown>& spUnkDataSrc )
{
    /* spUnkDataSrc contains an IUnknown pointer to an uninitialized IOM Data Source Object. */
    /* This method initializes the IOM Data Provider. */

    CComQIPtr<IDBInitialize> spInit = spUnkDataSrc;
    CComQIPtr<IDBProperties> spProp = spUnkDataSrc;
    if ( spInit == NULL || spProp == NULL )
        return E_FAIL;

    CDBPropSet Props( DBPROPSET_DBINIT );
    Props.AddProperty( DBPROP_INIT_DATASOURCE, (WCHAR*)L_LOCAL_ );
    FAIL_IF( spProp->SetProperties( 1, &Props ) );

    FAIL_IF( spInit->Initialize() );

    return S_OK;
}

HRESULT CreateSession( CComPtr<IUnknown>& spUnkDataSrc, CComPtr<IUnknown>& spUnkSession )
{
    /* spUnkDataSrc contains a pointer to the IUnknown interface of an initialized Data Source Object */
    /* spUnkSession is where a pointer to the IUnknown interface of the newly created
       Session Object will be stored */
    HRESULT hr = E_FAIL;
    CComQIPtr<IDBCreateSession> spCreateSession = spUnkDataSrc;
    if( spCreateSession == NULL )

```

```

cout << "ERROR: IDBCreateSession not found" << endl;
else
{
hr = spCreateSession->CreateSession( NULL, IID_IUnknown, &spUnkSession.p );
FAIL_IF( hr );
}
return hr;
}

HRESULT CreateNewTable( CComPtr<IUnknown>& spUnkSession )
{
/* spUnkSession contains a pointer to the IUnknown interface of an open Session Object */
/* This method creates a table named newtable from the sasuser library */
HRESULT hr = E_FAIL;
CComQIPtr<ITableDefinition> spTableDef = spUnkSession;
if( spTableDef == NULL )
cout << "ERROR: ITableDefinition not found" << endl;
else
{
CComPtr<IUnknown> spUnkNewTable = NULL;
DBORDINAL cColumnDescs = 3;
DBCOLUMNDESC rgColumnDescs[3];
DBID in_table;

in_table.eKind = DBKIND_NAME; /* only DBKIND_NAME supported */
in_table.uName.pwszName = L"sasuser.newtable";

memset( rgColumnDescs, 0, 3*sizeof(DBCOLUMNDESC) );

rgColumnDescs[0].wType = DBTYPE_R8;
rgColumnDescs[0].ulColumnSize = 8;
rgColumnDescs[0].dbcid.eKind = DBKIND_NAME; /* only DBKIND_NAME supported */
rgColumnDescs[0].dbcid.uName.pwszName = L"i";

rgColumnDescs[1].wType = DBTYPE_WSTR;
rgColumnDescs[1].ulColumnSize = 40;
rgColumnDescs[1].dbcid.eKind = DBKIND_NAME; /* only DBKIND_NAME supported */
rgColumnDescs[1].dbcid.uName.pwszName = L"name";

rgColumnDescs[2].wType = DBTYPE_R8;
rgColumnDescs[2].ulColumnSize = 8;
rgColumnDescs[2].dbcid.eKind = DBKIND_NAME; /* only DBKIND_NAME supported */
rgColumnDescs[2].dbcid.uName.pwszName = L"age";

hr = spTableDef->CreateTable(
NULL /* outer unknown */,
&in_table /*id of table to create */,
cColumnDescs /* number of columns */,
rgColumnDescs /* description of columns */,
IID_IUnknown /* interface to get on new data set */,
0 /* number of property sets */,
NULL /* property sets */,
NULL /* id of table created, always equal to the requested id */,
&spUnkNewTable.p /* pointer to new data set */ );
}

```

```

return hr;
}

HRESULT DeleteNewTable( CComPtr<IUnknown>& spUnkSession )
{
    /* spUnkSession contains a pointer to the IUnknown interface of an open Session Object */
    /* This method deletes a table named newtable from the sasuser library */
    HRESULT hr = E_FAIL;
    CComQIPtr<ITableDefinition> spTableDef = spUnkSession;
    if( spTableDef == NULL )
        cout << "ERROR: ITableDefinition not found" << endl;
    else
    {
        DBID tableid;
        tableid.eKind = DBKIND_NAME; /* only DBKIND_NAME supported */
        tableid.uName.pwszName = L"sasuser.newtable";
        hr = spTableDef->DropTable( &tableid );
    }
    return hr;
}

int main(int argc, char* argv[])
{
    CoInitialize(NULL);

    try
    {
        HRESULT hr = E_FAIL;
        CComPtr<IUnknown> spUnkDataSrc = NULL; /* pointer to IUnknown interface of a Data Source Object */
        CComPtr<IUnknown> spUnkSession = NULL; /* pointer to IUnknown interface of a Session Object */
        CComPtr<IUnknown> spUnkCommand = NULL; /* pointer to IUnknown interface of a Command Object */
        CComPtr<IUnknown> spUnkRowset = NULL; /* pointer to IUnknown interface of a Rowset Object */
        FAIL_IF( InstantiateProvider( spUnkDataSrc ) ); /* creates an instance of the IOM Provider Data Source */
        FAIL_IF( InitializeProvider( spUnkDataSrc ) ); /* initializes the Data Source object to
                                                    connect to the local workspace server */
        FAIL_IF( CreateSession( spUnkDataSrc, spUnkSession ) ); /* creates an instance of a Session Object */
        FAIL_IF( CreateNewTable( spUnkSession ) ); /* create a data set named newtable in the sasuser library */
        FAIL_IF( DeleteNewTable( spUnkSession ) ); /* delete a data set named newtable in the sasuser library */
    }
    catch( _com_error& e )
    {
        cout << "ERROR: 0x" << hex << setw(8) << setfill('0') << right << e.Error() << endl;
    }

    CoUninitialize();
    return 0;
}

```

Subsetting Data Sets for Read-Only Sequential Access

Goal

You want your application to execute SQL queries and commands in order to subset data sets for read-only sequential access.

This recipe applies to the SAS/SHARE, IOM, and Base SAS providers. This recipe includes sample code for ADO and OLE DB.

ADO Implementation

Three Methods for Executing SQL Queries and Commands

The IOM, SAS/SHARE, and Base SAS providers can be used to pass SQL statements in three ways:

- calling Execute on an open Connection object
- calling Execute on an open Command object
- calling Open on a Recordset object and passing adCmdText as the option

Note: All three methods require that you reference the Microsoft ActiveX Data Objects Library in your Visual Basic project.

Calling Execute on an Open Connection Object

The following code uses an open ADO Connection object in order to pass SQL statements to a provider. The data set has three columns: a numeric column named **i**, a character column of length 40 named **name**, and a numeric column named **age**.

```
' obConnection is an open Connection object.
Dim obRecordset As New ADODB.Recordset

' Create a new table in the Sasuser library.
obConnection.Execute "create table sasuser.newtable ( i num, name char(40), age num );"

' Insert values into the new table.
obConnection.Execute "insert into sasuser.newtable values( 0, "Bill", 32 ) values( 1, "John", 99 );"

' Use a SELECT statement to open a Recordset object on the new table.
Set obRecordset = obConnection.Execute("select * from sasuser.newtable")

' Do something with the Recordset object.
MsgBox obRecordset.GetString

' Close the Recordset object.
obRecordset.Close
```

Calling Execute on an Open Command Object

The same SQL statements shown in the previous example can be executed through an ADO Command object, as shown in the following Visual Basic code:

```
' obConnection is an open Connection object.
Dim obCommand As New ADODB.Command
Dim obRecordset As New ADODB.Recordset

obCommand.ActiveConnection = obConnection

' Create a new table.
obCommand.CommandText = "create table sasuser.newtable ( i num, name char(40), age num );"
obCommand.Execute

' Insert values into the new table.
obCommand.CommandText = "insert into sasuser.newtable values( 0, "Bill", 32 ) values( 1, "John", 99 );"
obCommand.Execute

' Open a Recordset object on the new table.
obCommand.CommandText = "select * from sasuser.newtable"
Set obRecordset = obCommand.Execute()

' Do something with the Recordset object.
MsgBox obRecordset.GetString

' Close the Recordset object.
obRecordset.Close
```

Calling Open on a Recordset Object and Passing adCmdText as the Option

SQL statements that result in recordsets can be executed through an ADO Recordset object by using the adCmdText option. The following Visual Basic code shows how this task is done.

```
' obConnection is an open Connection object.
Dim obRecordset As New ADODB.Recordset

' Execute a command that results in a recordset.
' SQL result sets are forward-only and read-only.
obRecordset.Open "select * from sasuser.newtable", obConnection, adOpenForwardOnly, _
                adLockReadOnly, adodb.adCmdText

' Do something with the Recordset object.
MsgBox obRecordset.GetString

' Close the Recordset object.
obRecordset.Close
```

OLE DB Implementation

OLE DB consumers can use the ICommand family of interfaces that are exposed by the Command object to execute SQL statements. The most common interfaces are ICommand and ICommandText. The following C++ code shows how this task is done. In the sample code, the first two statements return **NULL** for the pointer to the rowset

because no rowset is created. The third statement creates a rowset and returns a pointer to its IUnknown interface.

```
#include atlbase.h
#include comdef.h
#include comutil.h
#include oledb.h
#include atldbcli.h

#include <iostream>
#include <iomanip>
using std::cout;
using std::hex;
using std::setw;
using std::setfill;
using std::right;
using std::endl;

#define FAIL_IF( hr ) issue_if_failed( __FILE__, __LINE__, hr )
inline void issue_if_failed( char* file, ULONG line, HRESULT hr )
{
    /* This method throws an error if hr is a failure. */
    if( FAILED( hr ) )
    {
        ATLTRACE( %s(%d): Failure 0x%X\n, (char*)file, line, hr );
        _com_issue_error( hr );
    }
}

HRESULT InstantiateProvider( CComPtr<IUnknown>& spUnkDataSrc )
{
    /* spUnkDataSrc contains a NULL IUnknown pointer. */
    /* This method instantiates the IOM Provider Data Source object and */
    /* stores a pointer to its IUnknown interface in spUnkDataSrc. */
    CLSID clsid;

    FAIL_IF( CLSIDFromProgID( L"SAS.IOMProvider", &clsid ) );
    FAIL_IF( CoCreateInstance( clsid, NULL, CLSCTX_INPROC_SERVER, IID_IUnknown, (void**)&spUnkDataSrc ) );

    return S_OK;
}

HRESULT InitializeProvider( CComPtr<IUnknown>& spUnkDataSrc )
{
    /* spUnkDataSrc contains an IUnknown pointer to an uninitialized IOM Data Source Object. */
    /* This method initializes the IOM Provider specifically. Initializing the SAS/SHARE */
    /* Provider requires different properties as described in Establishing an ADO Connection. */
    CComQIPtr<IDBInitialize> spInit = spUnkDataSrc;
    CComQIPtr<IDBProperties> spProp = spUnkDataSrc;
    if( spInit == NULL || spProp == NULL )
        return E_FAIL;

    CDBPropSet Props( DBPROPSET_DBINIT );
    Props.AddProperty( DBPROP_INIT_DATASOURCE, (WCHAR*)L_LOCAL_ );
    FAIL_IF( spProp->SetProperties( 1, &Props ) );
}
```

```

    FAIL_IF( spInit->Initialize() );

    return S_OK;
}

HRESULT CreateSession( CComPtr<IUnknown>& spUnkDataSrc, CComPtr<IUnknown>& spUnkSession )
{
    /* spUnkDataSrc contains a pointer to the IUnknown interface of an initialized Data Source Object */
    /* spUnkSession is the location for a pointer to the IUnknown interface of the newly created Session Object */
    HRESULT hr = E_FAIL;
    CComQIPtr<IDBCreateSession> spCreateSession = spUnkDataSrc;
    if( spCreateSession == NULL )
        cout << "ERROR: IDBCreateSession not found" << endl;
    else
    {
        hr = spCreateSession->CreateSession( NULL, IID_IUnknown, &spUnkSession.p );
        FAIL_IF( hr );
    }
    return hr;
}

HRESULT CreateCommand( CComPtr<IUnknown>& spUnkSession, CComPtr<IUnknown>& spUnkCommand )
{
    /* spUnkSession contains a pointer to the IUnknown interface of an open Session Object */
    /* spUnkCommand is the location for a pointer to the IUnknown interface of the newly created Command Object */
    HRESULT hr = E_FAIL;
    CComQIPtr<IDBCreateCommand> spCreateCommand = spUnkSession;
    if( spCreateCommand == NULL )
        cout << "ERROR: IDBCreateCommand not found" << endl;
    {
        hr = spCreateCommand->CreateCommand( NULL, IID_IUnknown, &spUnkCommand.p );
        FAIL_IF( hr );
    }
    return hr;
}

HRESULT ExecutesQL( CComPtr<IUnknown>& spUnkCommand, bstr_t bstrSQL, CComPtr<IUnknown>& spUnkRowset )
{
    /* spUnkCommand contains a pointer to the IUnknown interface of an open command */
    /* spUnkRowset is where the IUnknown pointer to the created rowset (if any) will be stored */
    HRESULT hr = E_FAIL;
    CComQIPtr<ICommandText> spCommandText = spUnkCommand;
    if( spCommandText == NULL )
        cout << "ERROR: ICommandText not found" << endl;
    else
    {
        FAIL_IF( spCommandText->SetCommandText( DBGUID_DBSQL, bstrSQL ) );
        hr = spCommandText->Execute( NULL, IID_IUnknown, NULL, NULL, &spUnkRowset.p );
        FAIL_IF( hr );
    }
    return hr;
}

int main(int argc, char* argv[])
{

```

```

CoInitialize(NULL);

try
{
HRESULT hr = E_FAIL;
CComPtr<IUnknown> spUnkDataSrc = NULL; /* pointer to IUnknown interface of a Data Source Object */
CComPtr<IUnknown> spUnkSession = NULL; /* pointer to IUnknown interface of a Session Object */
CComPtr<IUnknown> spUnkCommand = NULL; /* pointer to IUnknown interface of a Command Object */
CComPtr<IUnknown> spUnkRowset = NULL; /* pointer to IUnknown interface of a Rowset Object */
FAIL_IF( InstantiateProvider( spUnkDataSrc ) ); /* creates an instance of the IOM Provider Data Source */
FAIL_IF( InitializeProvider( spUnkDataSrc ) ); /* initializes the Data Source object to
                                             connect to the local workspace server */
FAIL_IF( CreateSession( spUnkDataSrc, spUnkSession ) ); /* creates an instance of a Session Object */
FAIL_IF( CreateCommand( spUnkSession, spUnkCommand ) ); /* creates an instance of a Command Object */

/* create a new data set */
FAIL_IF( ExecuteSQL( spUnkCommand, "create table sasuser.newtable ( i num, name char(40), age num );",
                    spUnkRowset ) );

/* add some rows to the new data set */
FAIL_IF( ExecuteSQL( spUnkCommand, "insert into sasuser.newtable values( 0, \"Bill\", 32 )
                                values( 1, \"John\", 99 );", spUnkRowset ) );

/* open a Recordset on the new data set */
FAIL_IF( ExecuteSQL( spUnkCommand, "select * from sasuser.newtable", spUnkRowset ) );

/* do something with spUnkRowset */
/* ... */
}
catch( _com_error& e )
{
cout << "ERROR: 0x" << hex << setw(8) << setfill('0') << right << e.Error() << endl;
}

CoUninitialize();
return 0;
}

```

Subsetting Data Sets for Random and Update Access

Goal

You want your application to use the WHERE clause with the SAS SQL procedure to subset data sets for random and update access.

This recipe applies to the SAS/SHARE and Base SAS providers. Sample code for ADO is included.

Note: For more information about WHERE clause processing, see *SAS Language Reference: Concepts* and the *Base SAS Procedures Guide*.

Implementation

Sample Code for Subsetting Data Sets

The SAS SQL procedure enables you to subset data sets by using a WHERE clause. A data set that is opened by a SAS WHERE clause has the same features as other SAS data sets, so you can open it for both random and update access.

To control WHERE clause processing, you use the "SAS Where" provider property. The value of the "SAS Where" property is a string that is a complete and valid SAS WHERE expression. The following Visual Basic code shows how this task is done.

```
' obConnection is an open Connection object.
Dim obRecordset As New ADODB.Recordset

obRecordSet.ActiveConnection = obConnection
obRecordSet.Properties("SAS Where") = "flight > 306"

' The second parameter on the Open method must remain empty.
obRecordSet.Open "airline.Flights", , adOpenStatic, adLockReadOnly, adCmdTableDirect
```

A Closer Look at the 'SAS Where' Property

Although the "SAS Where" provider property might seem to serve the same function as the ADO Recordset object's "Filter" property, there are differences:

- The "SAS Where" property sends the clause to the server, and the server determines which records pass the WHERE clause. The server returns only records that pass the WHERE clause.
- The "Filter" property sends all of the records in the data set to the client. The client then processes each record to determine whether it should be in the recordset.

See Also

[“Subsetting Data Sets for Read-Only Sequential Access” on page 85](#)

Chapter 9

Specifying How to Display Data

Displaying Data Recipes	91
Using SAS Formats When You Read Data	92
Goal	92
ADO Implementation	92
OLE DB Implementation	94
Using SAS Informats When You Write Data	94
Goal	94
ADO Implementation	94
OLE DB Implementation	96
Reading User-Defined SAS Formats and Informats	96
Goal	96
ADO Implementation	96
Padding Character Data with Blanks	97
Goal	97
ADO Implementation	98
OLE DB Implementation	98
Accessing Data with a Different Encoding	98
Goal	98
OLE DB Implementation	98

Displaying Data Recipes

This chapter provides sample code that you can use to perform tasks related to displaying data after the connection is established. These tasks are included because they have some aspects that are specific to SAS.

Note: To perform a data-related task that is not included in this chapter, see the *OLE DB Programmer's Reference and Data Access SDK*.

TIP To use the recipes in this chapter, you need an open ADO Connection object. For more information, see [“Basic Connection Recipes” on page 37](#).

See Also

[“Supplemental Connection Recipes” on page 49](#)

Using SAS Formats When You Read Data

Goal

You want your application to use SAS formats when reading data.

This recipe applies to the local, SAS/SHARE, IOM, and Base SAS providers. Sample code for ADO is included.

ADO Implementation

How to Structure the Value of the "SAS Formats" Property

SAS implements a set of formats that can be used to transform basic character and numeric values into formatted character values. In ADO, formats are controlled with the "SAS Formats" customized Recordset property. This property takes a string value that specifies the format that you want to use, and specifies any overriding information. The string value also lists one or more columns to which the format should be applied. Use the following form to enter the string value:

```
[+] COLUMN [=FORMAT [w] . [d]]
```

Here is an explanation of the form:

- `+`, an optional modifier that exposes the column in two conditions: formatted and unformatted.
- `COLUMN`, the name of a column in the recordset (which is the same as the variable name in the SAS data set). If you need to specify more than one column, you must separate the column names with commas.
- `FORMAT`, the name of the format that you want to apply.
- `w`, the width that you want to use with the format.
- `d`, the number of decimal places (if any) to include with numeric formats.

The order in which the columns are listed in the string does not affect the order in which they are returned by an ADO Recordset object.

For example, assume that a data set has three variables that appear in the following order:

1. **SALEDATE**, a SAS date with a default format of MMMYY
2. **QUANTITY**, a SAS numeric value with no default format
3. **PRICE**, a SAS numeric value with no default format

The following string value is based on the sample data set:

```
SALEDATE=MMDDYY8. , +PRICE=DOLLAR8.2
```

The results appear in the following four columns on the ADO Recordset object:

1. **SALEDATE**, a string column with the MMDDYY8. format applied
2. **QUANTITY**, a numeric column with no format applied
3. **PRICE**, a numeric column with no format applied

4. `PRICE_DOLLAR8.2` as a string column with the `DOLLAR8.2` format applied.

Note: The fourth column is constructed through the use of the + modifier; it is not persisted beyond the life of its exposure on the ADO Recordset object.

How Default Formats Are Applied

If there is no explicit format information, or if you specify "`_ALL_`" as the "SAS Formats" property, the SAS providers apply format properties to all columns according to the following rules:

- If no format name, width, or decimal value is specified with a column name, then the providers default to the format information that is persisted with the data set.
- If no format specification is stored in the data set, then the providers use the system defaults. The system defaults are `W.D` for numeric columns and `$CHAR.` for character columns.

How the Plus (+) Modifier Works

To see the same physical column of data exposed as two columns, one formatted and one unformatted, prefix the column name with the plus (+) sign modifier. When you make this request, the formatted column appears as a constructed, temporary column named `COLUMN_FORMAT`. The formatted column is not persisted beyond the life of its exposure on the ADO Recordset object.

To see all columns returned in both their formatted and unformatted forms, use the form "`+_ALL_`".

CAUTION:

Data integrity problems can occur when the data provider cannot determine which version of a column (formatted or unformatted) should be written to the data set. To prevent these problems, use the + modifier to restrict the recordset to read-only access.

How to Determine Persisted SAS Format Information

To determine column metadata at run time, use the Connection object's `OpenSchema` method and set its `QueryType` parameter to `adSchemaColumns`. The returned recordset will contain the persisted SAS format information for each column, along with the defined OLE DB schema information.

Note: For more information about displaying SAS metadata, see [“Displaying Metadata That Is Specific to SAS Data Sets”](#) on page 73.

Sample Code for Setting the "SAS Formats" Property

The following Visual Basic code shows how to set the "SAS Formats" property so that the columns in the recordset will include the formatting as indicated by the "SAS Formats" value.

```
' obConnection is an open Connection object.
Dim obRecordset As New ADODB.Recordset

obRecordset.ActiveConnection = obConnection
obRecordset.Properties("SAS Formats") = "saledate=mmddyy8.,+price=dollar8.2."

' The second parameter on the Open method must remain empty.
obRecordset.Open "sales", , adOpenStatic, adLockReadOnly, adCmdTableDirect
```

OLE DB Implementation

An application that calls a SAS provider directly through the OLE DB interfaces should use the SASFORMAT structure with the DBBINDING extensions. For more information, see [“About Format and Informat Processing with OLE DB”](#) on page 253.

See Also

[“Using SAS Informats When You Write Data”](#) on page 94

Using SAS Informats When You Write Data

Goal

You want your application to use SAS informats when writing data.

This recipe applies to the SAS/SHARE, IOM, and Base SAS providers. Sample code for ADO is included.

ADO Implementation

How to Structure the Value of the 'SAS Informats' Property

SAS implements a set of informats that can be used to transform formatted character values into basic character and numeric values. Informats are controlled with the SAS Informats customized Recordset property.

This property takes a string value that specifies the informat that you want to use. The string value also lists one or more columns to which it should be applied. Use the following form to enter the string value:

```
COLUMN [=INFORMAT [w] . [d] ]
```

Here is an explanation of the form:

- *COLUMN*, the name of a column in the recordset (or variable in the data set). (If you need to specify more than one column, you must separate the column names with commas.)
- *INFORMAT*, the name of the informat that you want to apply
- *w*, the width that you want to use with the informat
- *d*, the number of decimal places (if any) to include with numeric informats

For example, the following informat string value removes the dollar sign and commas from the values in a column named `amount`. For an input value of \$1,000,000, the result is a numeric value of `1000000`.

```
amount=DOLLAR11.
```

How Default Informats Are Applied

If there is no explicit informat information (or if you specify `"_ALL_"` as the "SAS Informats" property value), the IOM and SAS/SHARE providers apply informat properties to all columns according to the following rules:

- If no informat name, width, or decimal value is specified with a column name, then the providers use the default informat information that is persisted with the data set.
- If no informat specification is persisted in the data set, then the providers use the system default values. The system default values are W.D for numeric columns and \$CHAR. for character columns.

Note: The "SAS Informats" property does not support use of the plus sign (+) modifier with the "_ALL_" keyword. This modifier can be used with the "SAS Formats" property to expose data as two columns: one formatted and one unformatted.

How the Use of Informats Affects Formatting

Whenever an informat is applied to a column, the SAS providers also apply format properties to that column. If a format is not explicitly specified through the use of the "SAS Formats" property, then the provider that is being used in the application applies the column's default format. (For more information about using formats, see ["Using SAS Formats When You Read Data" on page 92.](#))

The following Visual Basic code uses a simple SAS data set named **people** that contains two columns: **name** and **birthday**. The code specifies an informat for the **birthday** column, which is sufficient for the task of adding a few new records to the data set.

```
' obConnection is an open Connection object.
Dim obRecordset As New ADODB.Recordset

obRecordset.ActiveConnection = obConnection
obRecordset.Properties("SAS Informats") = "birthday=MMDDYY8."

' The second parameter on the Open method must remain empty.
obRecordset.Open "people", , adOpenStatic, adLockPessimistic, adCmdTableDirect
obRecordset.AddNew Array("name", "birthday"), Array("Beth", "02/13/73")
obRecordset.AddNew Array("name", "birthday"), Array("David", "05/01/65")
obRecordset.Close
```

Because the "SAS Formats" property is not used to explicitly specify a format for **birthday**, the column's default format is used. Because this code only writes data, the applied format does not affect the results. For applications that read and write records, you must ensure that each column uses the appropriate informat and format.

In the following Visual Basic code, the first statement applies a format to the **birthday** column. The second statement applies a complementary informat to the **birthday** column.

```
obRecordset.Properties("SAS Formats") = "birthday=MMDDYY8."
obRecordset.Properties("SAS Informats") = "birthday=MMDDYY8."
```

By contrast, in the following code, the property settings for **birthday** are not complementary. The format (DOLLAR9.2) implies that the underlying numeric value is monetary, but the informat (MMDDYY8.) implies that the underlying value is a SAS date.

```
obRecordset.Properties("SAS Formats") = "birthday=DOLLAR9.2"
obRecordset.Properties("SAS Informats") = "birthday=MMDDYY8."
```

Even though this pairing of format and informat is not logical, the providers will not prohibit you from entering this type of configuration. You must make reasonable choices.

Note: As with informats, you can use the "_ALL_" keyword to specify that all columns use either system default formats or persisted formats.

How to Determine Persisted SAS Informat Information

To determine column metadata at run time, use the Connection object `OpenSchema` method and set its `QueryType` parameter to `adSchemaColumns`. The returned recordset will contain the persisted SAS informat information for each column, along with the defined OLE DB schema information.

Note: For more information about displaying SAS metadata, see [“Displaying Metadata That Is Specific to SAS Data Sets”](#) on page 73.

OLE DB Implementation

An application that calls a SAS provider directly through the OLE DB interfaces should use the `SASFORMAT` structure with the `DBBINDING` extensions. For more information, see [“About Format and Informat Processing with OLE DB”](#) on page 253.

See Also

[“Using SAS Formats When You Read Data ”](#) on page 92

Reading User-Defined SAS Formats and Informats**Goal**

You want your application to read user-defined SAS formats and informats.

This recipe applies to the IOM provider. Sample code for ADO is included.

Note: The SAS Workspace Server must have access to the SAS catalog that contains the user-defined formats and informats.

ADO Implementation**Sample Code for Reading User-Defined Formats**

The following Visual Basic code shows you how to read a SAS data set named `books` that uses formats that are contained in a catalog named `formats`. The data set is located in a directory named `c:\storage`. The catalog is located in a directory named `c:\public`.

Note: If you define the `libref` to point to the location where the catalog is stored, your SAS Workspace Server can access your user-written formats when you process data sets by using either ADO recordsets or OLE DB rowsets.

```
' obConnection is an open Connection object.
Dim obRecordset As New ADODB.Recordset

' Use the formats persisted on the data set
obRecordset.Properties("SAS Formats") = "_ALL_"

' Assign a libref for the formats catalog.
obConnection.Execute ("libname library 'c:\public' ")
```

```
' Assign a libref for the data set.
obConnection.Execute ("libname mylib 'c:\storage' ")

' The second parameter on the Open method must remain empty.
bRecordset.Open "mylib.books", , adOpenDynamic, adLockReadOnly, adCmdTableDirect
```

Note: Currently, user-written formats cannot be processed on the client side, which is why the IOM provider can process user-written formats but the local, SAS/SHARE, and Base SAS providers cannot. The IOM provider delegates all format processing to the server side. However, the local, SAS/SHARE, and Base SAS providers do all format processing on the client side.

CAUTION:

Do not attempt to open an ADO Recordset object if the underlying data set contains user-written formats that are not immediately available to the IOM workspace. This action results in a fatal error that is generated even if you are not attempting to use the formats when reading the data.

How to Manage Errors When an Unsupported Provider Is Used

By default, you will get an error if you attempt to use either the local, SAS/SHARE, or Base SAS provider to perform a row I/O operation with a user-written format or informat. For example, if you try to read a row and apply a customized format, the attempt will fail. Likewise, an update or add row operation will fail if a user-written informat is associated with any column in the row.

You can enable processing to continue without error by changing the value of the Recordset property "SAS Format Error" (the OLE DB property name is DBPROP_SAS_FMTERR). This property controls how the local, SAS/SHARE, and Base SAS providers process format and informat names that cannot be resolved.

By default, the value of this property is **True**, which causes unknown formats and informats to be flagged as errors. If you set this property to **False** when opening a recordset, the default format or informat (W.D for numeric columns and \$CHAR. for character columns) will be applied when an unknown format or informat is encountered.

See Also

- [“Specifying a Libref to Use with the IOM Provider” on page 69](#)
- [“Using SAS Formats When You Read Data ” on page 92](#)

Padding Character Data with Blanks

Goal

You want your application to preserve trailing blanks.

This recipe applies to the local, SAS/SHARE, IOM, and Base SAS providers. Sample code for ADO is included.

ADO Implementation

By default, the SAS providers trim trailing blanks from character columns. This behavior differs from the default SAS DATA step behavior in which trailing blanks are preserved. To force the SAS providers to preserve trailing blanks, set the "SAS Preserve Trailing Blanks" property to **True**. The following sample Visual Basic code shows how this task is done.

```
' obConnection is an open Connection object.
Dim obRecordset As New ADODB.Recordset

obRecordset.ActiveConnection = obConnection
obRecordset.Properties("SAS Preserve Trailing Blanks") = True

' The second parameter on the Open method must remain empty.
obRecordset.Open "sashelp.shoes", , adOpenStatic, adLockReadOnly, adCmdTableDirect
```

OLE DB Implementation

When writing to the OLE DB interface, use `DBPROP_SAS_BLANKPADDING` (a member of the `DBPROPSET_SAS_ROWSET` property set) to control padding behavior.

Accessing Data with a Different Encoding

Goal

You want to access data sets that use an encoding that is different from the machine locale, and want to display characters instead of question marks (?).

OLE DB Implementation

When the local provider is used to access data that contains international characters, and the character variables are not bound correctly, the character values appear as question marks (?). When reading character data from a data set that has an encoding that is different from the machine locale, bind the type as `DBTYPE_WSTR`.

Example Code 9.1 Binding Character Data as `DBTYPE_WSTR`

```
DBBINDING          rgBind[MAXBINDINGS];
DBCOLUMNINFO       *rgInfo = NULL;
DBORDINAL          cNumColumns, i = 0;

for (i = 0; i < cNumColumns; i++)
{
    rgBind[i].cbMaxLen = rgInfo[i].ulColumnSize + sizeof(wchar_t); 1
    // more code
    rgBind[i].wType = DBTYPE_WSTR; 2
    rgBind[i].bPrecision = rgInfo[i].bPrecision;
    rgBind[i].bScale = rgInfo[i].bScale;
    rgBind[i].obLength = dwOffset + offsetof(COLUMNDATA, dwLength);
    rgBind[i].obStatus = dbOffset + offsetof(COLUMNDATA, dwStatus);
```

```
rgBind[i].obValue = dbOffset + offsetof(COLUMNNDATA, bData);  
}
```

- 1 This line of code shows how to calculate the length of the character data. In UTF-16 encoding, each character is stored in two bytes.
- 2 When binding to DBTYPE_WSTR, then the local provider returns the character data in UTF-16 encoding.

There are three ways to bind character data:

DBTYPE_STR	The local provider interprets the data according to the Windows code page that is associated with the locale of the machine. This is also known as the thread encoding. This value is different from the Windows system code page.
DBTYPE_WSTR	The local provider interprets the data as UTF-16.
DBTYPE_BSTR	The local provider interprets the data as a BSTR, as defined by Microsoft. This is similar to UTF-16 and is the type that is used by all ADO client applications.

Chapter 10

Managing Missing Values

Missing Value Recipes	101
Reading Missing Values from a Data Set	102
Goal	102
ADO Implementation	102
OLE DB Implementation	103
Reading Special Numeric Missing Values from a Data Set	105
Goal	105
Reading Special Numeric Missing Values	105
Writing Missing Values to a Data Set	108
Goal	108
ADO Implementation	108
OLE DB Implementation	109

Missing Value Recipes

This chapter provides sample code that you can use to perform tasks related to reading and writing missing values. These tasks are included because they have some aspects that are specific to SAS.

Note: To perform a data-related task that is not included in this chapter, see the *OLE DB Programmer's Reference and Data Access SDK*.

Here is a list of the recipes in this chapter:

- “Reading Missing Values from a Data Set” on page 102
- “Reading Special Numeric Missing Values from a Data Set” on page 105
- “Writing Missing Values to a Data Set” on page 108

TIP To use the recipes in this chapter, you need an open ADO Connection object. For more information, see “Basic Connection Recipes” on page 37.

See Also

“Supplemental Connection Recipes” on page 49

Reading Missing Values from a Data Set

Goal

You want your application to test for missing values.

Note: The OLE DB specification supports only one type of missing value for each data type. Consequently, the SAS providers support only missing numeric values and missing character values.

This recipe applies to the local, SAS/SHARE, IOM, and Base SAS providers. This recipe includes sample code for ADO and OLE DB.

ADO Implementation

How Missing Values Are Represented

ADO represents missing numeric values as a Variant data type with a value of Null, which means that the IsNull function can be used to test for missing values. If a numeric field value in a record is missing, then IsNull returns a true value for that field.

To test for missing character values, you compare trimmed character values in the data set against the empty string "".

Note: Schema rowsets return missing character values as Null values in order to provide greater interoperability with third-party clients such as Microsoft Excel. The IsNull function also returns true for missing character values in the rowsets. Always check the type of the column to determine whether there is a missing character value.

Representation of missing character values depends on whether blank padding is preserved (see [“Padding Character Data with Blanks” on page 97](#)).

- If blank padding is preserved, then missing character values are represented as a string of spaces with a length equal to the width of the particular column.
- If blank padding is not preserved, then missing character values are represented as an empty (zero length) string.

Note: If you are not sure about the state of the "SAS Preserve Trailing Blanks" property, you can use the trim function and compare the value against the empty string "".

Sample ADO Code That Tests for Missing Values

The following Visual Basic code tests every character and numeric value in a data set to determine whether a value is missing or not.

```
' obConnection is an open Connection object.
Dim obRecordset As New ADODB.Recordset
Dim field As Field

obRecordset.Open "sasuser.shoes", obConnection, adOpenDynamic, adLockPessimistic, adCmdTableDirect
obRecordset.MoveFirst
While Not obRecordset.EOF
  For Each field In obRecordset.Fields
    If (field.Type = adVarChar Or field.Type = adVarWChar) And _
      (IsNull(field.Value) Or Trim(field.Value) = "") Then
```

```

Debug.Print "missing character value found in column " & field.Name
ElseIf IsNull(field.Value) Then
Debug.Print "missing numeric value found in column " & field.Name
Else
Debug.Print "no missing value found"
End If
Next field
obRecordset.MoveNext
Wend
obRecordset.Close

```

Sample Code That Traps Errors Caused by Missing Numeric Values (ADO)

Because missing numeric values are represented as a Variant data type with a value of Null instead of as a numeric data type, you cannot use missing numeric values in operations that require numeric data types. If you do, your application will produce a run-time error on the first missing numeric value that it encounters.

To trap the error and display a message to the user, use the Visual Basic On Error statement. The following Visual Basic code shows how this task is done:

```

Function TotalStores() as Integer
    Dim Total as Integer
    Total = 0
    On Error GoTo Missing
    While Not obRecordset.EOF
        Total = Total + obRecordset!Stores
        obRecordset.MoveNext
    Wend
    On Error GoTo 0
    TotalStores = Total
    Exit Function

Missing:
    If Err.Number = 94 Then
        MsgBox "Missing value encountered"
    Else
        MsgBox "Run-time error: " & Err.Number & vbNewLine & Err.description
    End If
    Resume Next
End Function

```

OLE DB Implementation

How Missing Values Are Represented

When you write to the OLE DB interface, use the IRowset::GetData method to test for missing numeric values in a data set. If a numeric field value in a column in the client's row data buffer is missing, then the dwStatus value is set to DBSTATUS_S_ISNULL. If the client wishes to receive the appropriate SAS missing value indicator, then the client should set the DBPROP_SAS_MISSING_VALUES property to **True**. The possible missing value indicator values are as follows:

- NumericMissingIgnore(0)
- NumericMissingUnderscore(16)

- NumericMissingDot(32)
- NumericMissingA(33)
- NumericMissingZ(58)
- NotMissing(127)

If a character field value in a column in the client's row data buffer is missing, then the `dwLength` value is set to zero (0) or the `bData` value contains only spaces.

Note: Schema rowsets will return missing character values as NULL values in order to provide greater interoperability with third-party clients such as Microsoft Excel. The `dwStatus` flag will also be set to `DBSTATUS_S_ISNULL` for missing character values in the rowsets. Always check the column type to determine whether the value is a missing character or a missing numeric.

The representation of missing character values depends on whether blank padding is preserved. In the case of OLE DB, the `DBPROP_SAS_BLANKPADDING` property controls this behavior.

- If blank padding is not preserved (`DBPROP_SAS_BLANKPADDING` is set to `VARIANT_FALSE`), then the `dwLength` member of the column in the client's row data buffer will be set to zero (which represents the width of the empty column).
- If blank padding is preserved (`DBPROP_SAS_BLANKPADDING` is set to `VARIANT_TRUE`), then the `bData` member of the column in the client's row data buffer will be a string of spaces.

Sample Code That Tests for Missing Values (OLE DB)

The following C++ code shows how to test for missing values in a single observation that is returned from `IRowset::GetData`. The code assumes that you have completed the following tasks:

1. called `IColumnInfo::GetColumnInfo` to get a `DBCOLUMNINFO` array and to determine the number of columns in the data set
2. used the `DBCOLUMNINFO` array to create a `DBBINDING` array
3. called `IAccessor::CreateAccessor` to request an accessor
4. called `IAccessor::GetBindings` to recover the bindings for the accessor that you created
5. used the bindings to determine how much space to allocate for the row data
6. called `IRowsetLocate::GetRowsAt` or `IRowset::GetNextRows` to get an `HROW`
7. used the `HROW` and the accessor in the call to `GetData`

Note: The exact steps that show how to get a row of data from an OLE DB provider can be found in the *OLE DB Programmer's Reference and Data Access SDK*.

```
hr = pIRowset->GetData(hRow, hAccessor, pRowData)
if (FAILED(hr)){ /* an error occurred */ }
for (DBORDINAL i = 0; i < cNumColumns; i++)
{
    pCol = (COLUMNDATA*)(pRowData + pBindings[i].obLength);
    if (pBindings[i].wType == DBTYPE_R8)
    {
        if (pCol->dwStatus == DBSTATUS_S_ISNULL)
        {
            //This is a missing numeric value
        }
    }
}
```

```

    }
    else
    {
        //This is a non-missing numeric value
    }
}
else if (pBindings[i].wType == DBTYPE_STR || pBindings[i].wType == DBTYPE_WSTR)
{
    DWORD j;
    // Loop through each character in the data member to
    // see if it contains a string of blanks or if it has a length of zero
    for (j = 0;
        j < pCol->dwLength &&
        (char*)pCol->bData[j] &&
        (char*)pCol->bData[j] == ' ');
        j++);
    if (j == pCol->dwLength)
    {
        //This is a missing character value
    }
    else
    {
        //This is a non-missing character value
    }
}
else
{
    // An unexpected type was returned
}
}

```

See Also

[“Writing Missing Values to a Data Set” on page 108](#)

Reading Special Numeric Missing Values from a Data Set

Goal

You want to read the special numeric missing value indicator in SAS data sets.

This recipe applies to the local provider. Sample code for ADO is provided. For OLE DB, SAS recommends using the DBPROP_SAS_MISSING_VALUES property instead.

Reading Special Numeric Missing Values

The local provider provides ADO rowset property “SAS Get Missing Values Grid” so that the local provider can return results that indicate the special numeric missing values. SAS supports special missing numeric values to represent different categories of missing data. For more information about special numeric missing values, see “Missing Values” in *SAS Language Reference: Concepts*.

In order to return the special missing numeric value, when this property is set to True, the provider returns a grid with values that are the opposite of the standard data. Numeric fields with special numeric missing values normally return Null. When this property is set to True, those fields return a value that represents the special numeric missing value.

```
// This sample uses C# syntax
//
// obConnection is an ADODB.Connection object and
// obRecordset is an ADODB.Recordset object
//
try
{
    string connectionString = "Provider=sas.LocalProvider; Data Source=\"c:\\data\"";
    obConnection.Open(connectionString, "", "",
        (int)ADODB.ConnectOptionEnum.adConnectUnspecified);

    obRecordset.ActiveConnection = obConnection;
    obRecordset.Properties["SAS Get Missing Values Grid"].Value = true;
    obRecordset.Open(dataSetName, System.Type.Missing,
        CursorTypeEnum.adOpenForwardOnly, LockTypeEnum.adLockReadOnly,
        (int)CommandTypeEnum.adCmdTableDirect);
}
catch (Exception ex)
{
    if (obConnection.Errors.Count > 0)
    {
        foreach (ADODB.Error e in obConnection.Errors)
            Console.WriteLine(e.Description);
    }
    else
    {
        Console.WriteLine(ex.ToString());
    }
}
}
```

Consider the following table that represents a SAS data set. It has a character variable and a numeric variable. The second observation has the standard SAS missing value. The third and fourth observations use the special numeric missing variables.

Table 10.1 Input Table with Special Numeric Missing Values

CharacterVariable	NumericVariable
FirstObservation	10
SecondObservation	.
ThirdObservation	.A
FourthObservation	._

When the “SAS Get Missing Values Grid” property is set to True and the data set is read, the results resemble the following table. All character variables are set to missing. Non-missing numeric variables are set to missing. The fields with special numeric missing values in the original data set return values to represent the missing value.

Table 10.2 Result Recordset with “SAS Get Missing Values Grid” Set to True

CharacterVariable	NumericVariable
	32
	33
	16

The following table shows the mapping from the special numeric missing value to the value that is returned in the record set.

Table 10.3 Special Numeric Missing Value Representation

Missing Value	Returned Value
.	32
A	33
B	34
C	35
D	36
E	37
F	38
G	39
H	40
I	41
J	42
K	43
L	44
M	45
N	46
O	47
P	48
Q	49

Missing Value	Returned Value
R	50
S	51
T	52
U	53
V	54
W	55
X	56
Y	57
Z	58
–	16

Writing Missing Values to a Data Set

Goal

You want your application to write missing values to a data set.

This recipe applies to the SAS/SHARE, IOM, and Base SAS providers. This recipe includes sample code for ADO and OLE DB.

ADO Implementation

To write missing values to a data set, you specify null as the value of the empty field. For missing character values, you can also set the value of the field to a string that contains all spaces (including the empty string " "). The following Visual Basic code sets every field in the data set to missing:

```
' obConnection is an open Connection object.
Dim obRecordset As New ADODB.Recordset
Dim field As Field

' Specify the LockType.
obRecordset.LockType = adLockPessimistic

' The fourth parameter is empty because the LockType has already been specified.
obRecordset.Open "sashelp.shoes", obConnection, adOpenDynamic, , adCmdTableDirect

obRecordset.MoveFirst
While Not obRecordset.EOF
  For Each field in obRecordset.Fields
```

```

        If field.Type = adWChar Then 'character value
        field.Value = ""
        ' or field.Value = Null
        ' or field.Value = " "
        Else 'numeric value
        field.Value = Null
        End If
    Next field
    obRecordset.MoveNext
Wend

```

OLE DB Implementation

To write missing values to a data set, you specify `DBSTATUS_S_ISNULL` as the value of the `dwStatus` of the column in the client's row data buffer. You perform this task before passing that buffer to the `IRowsetChange::SetData` method.

For missing character values, you can also fill the `bData` member of the column in the client's row data buffer with a null-terminated string of spaces (including the empty string `" "`). The length of the string of spaces can be up to the length indicated by the `dwLength` value.

The following C++ code shows how to set all the fields of one observation in a data set to `Null`. The iteration through the fields starts at 1 rather than 0 because the first column contains the bookmark for that row.

This example assumes that the rowset has been opened in immediate update mode. If a delayed update is used instead, an additional call to `IRowsetUpdate::Update` must be made before the changes will be accepted. The code assumes that you have completed the following tasks:

1. called `IColumnsInfo::GetColumnInfo` to get a `DBCOLUMNINFO` array and to determine the number of columns in the data set
2. used the `DBCOLUMNINFO` array to create a `DBBINDING` array
3. called `IAccessor::CreateAccessor` to request an accessor
4. called `IAccessor::GetBindings` to recover the bindings for the created accessor
5. used the bindings to determine how much space to allocate for the row data
6. called `IRowsetLocate::GetRowsAt` or `IRowset::GetNextRows` to get an `HROW`
7. used the `HROW` and the accessor in the call to `GetData`

Note: The exact steps for getting a row of data from an OLE DB provider can be found in the *OLE DB Programmer's Reference and Data Access SDK*.

```

hr = pIRowset->GetData(hRow, hAccessor, pRowData);
if (FAILED(hr)) { /* an error occurred */ }
for (DBORDINAL I = 1; I < cNumColumns; I++)
{
    pCol = (COLUMNDATA*) (pRowData +
        pBindings[I].obLength); pCol->dwStatus = DBSTATUS_S_ISNULL;
}
hr = pIRowset->QueryInterface(IID_IRowsetChange, (void**) &pIRowsetChange);
if (FAILED(hr)) { /* an error occurred */ }
hr = pIRowsetChange->SetData(hRow, hAccessor, pRowData);
if (FAILED(hr)) { /* an error occurred */ }

```

The following example shows an alternative method for writing character values to a data set:

```

hr = pIRowset->GetData(hRow, hAccessor, pRowData);
if (FAILED(hr)){ /* an error occurred */ }
for (DBORDINAL i = 1; i < cNumColumns; i++)
{
    pCol = (COLUMNNDATA*)(pRowData + pBindings[i].obLength);
    if (pBindings[i].wType == DBTYPE_STR)
    {
        memset(pCol->bData, ' ', pCol->dwLength);
    }
    else // numeric column
    {
        pCol->dwStatus = DBSTATUS_S_ISNULL;
    }
}
hr = pIRowset->QueryInterface(IID_IRowsetChange, (void**)&pIRowsetChange);
if (FAILED(hr)){ /* an error occurred */ }
hr = pIRowsetChange->SetData(hRow, hAccessor, pRowData);
if (FAILED(hr)){ /* an error occurred */ }

```

See Also

[“Reading Missing Values from a Data Set” on page 102](#)

Chapter 11

Managing Updates

Updating and Locking Recipes	111
Updating Recordsets	111
Goal	111
Implementation	112
Implementing a Locking Strategy	113
Goal	113
Implementation	113

Updating and Locking Recipes

This chapter provides sample code that you can use to perform tasks related to updating and locking ADO recordsets. These tasks are included because they have some aspects that are specific to SAS.

Note: To perform a data-related task that is not included in this chapter, see the *OLE DB Programmer's Reference and Data Access SDK*.

Here is a list of the recipes in this chapter:

- [“Updating Recordsets” on page 111](#)
- [“Implementing a Locking Strategy” on page 113](#)

TIP To use the recipes in this chapter, you need an open ADO Connection object. For more information, see [“Basic Connection Recipes” on page 37](#).

See Also

[“Supplemental Connection Recipes” on page 49](#)

Updating Recordsets

Goal

You want your application to perform recordset updates, including batch updates.

This recipe applies to the SAS/SHARE, IOM, and Base SAS providers. This recipe applies only to ADO. Sample code for batch updating is included.

Implementation

The Recordset Property Values

To perform recordset updates, you set the Recordset properties to the values shown in the following table.

Table 11.1 ADO Recordset Properties and Values

ADO Recordset Property	Value
CommandType	adCmdTableDirect*
LockType	adLockBatchOptimistic, adLockOptimistic, or adLockPessimistic For information about using adLockBatchOptimistic, see “Sample Code for Using adLockBatchOptimistic” on page 112.
CursorType	If possible, set the property to a value other than its default, which is adForwardOnly.

* If you set the CommandType property to a value other than adCmdTableDirect, the LockType property might revert to adLockReadOnly. Because the provider is not guaranteed to honor the requested lock type, you should always check the value of LockType after the recordset is opened.

Sample Code for Using adLockBatchOptimistic

The SAS providers maintain a row cache. When you open a Recordset object by using the adLockBatchOptimistic lock type, each altered record is added to the cache. The maximum number of records allowed in the cache is set by the "Maximum Open Rows" property.

A call to the UpdateBatch method transmits each modified row to the data source and flushes out the row cache. For the call to be successful, you must limit the number of rows that are updated between each call to UpdateBatch. The number of rows that are updated between each call should not exceed the value of the "Maximum Open Rows" property.

The following Visual Basic code shows you how to update records by using the adLockBatchOptimistic lock type:

```
' obConnection is an open Connection object.
Dim obRecordset As New ADODB.Recordset

' Set the ActiveConnection, CursorType, and LockType here instead of on the Open method.
obRecordset.ActiveConnection = obConnection
obRecordset.CursorType = adOpenStatic
obRecordset.LockType = adLockBatchOptimistic

' Set the maximum number of rows for batch update.
obRecordset.Properties("Maximum Open Rows") = 1000

' Set exclusive member-level lock. All other users get read-only access.
```

```

obRecordset.Properties("SAS Optimistic Locking") = False

obRecordset.Open "mydata.finance", , , , adCmdTableDirect

' Update every row, setting the BALANCE column of each row to 0.
While Not obRecordset.EOF
  obRecordset!BALANCE = 0
  obRecordset.MoveNext
Wend
obRecordset.UpdateBatch

```

A Closer Look at adLockBatchOptimistic

Recordset objects that are opened by using the adLockBatchOptimistic lock type have the following limitations:

- SAS permits batch updates only when the data set has been opened with an exclusive member lock. If a data set has not been opened with an exclusive member lock, then the UpdateBatch method ignores the batch request and updates the records one at a time. To enable an exclusive member lock, set the customized Recordset property "SAS Optimistic Locking" to **False**.

Note: An exclusive member lock means that only one user can update the data set. So, only one user at a time can perform batch updates.

- The number of records that can be updated with one batch update is limited by the value of the "Maximum Open Rows" property. The default value of this property is 100. If you need to update more than 100 records at a time, then change the property value to a higher number.

Implementing a Locking Strategy

Goal

You want your application to lock records, especially during update access. (Locking records during update access can prevent data loss.)

Note: When a record is being updated, ADO copies the record to a local record buffer where the changes are made. The changes are not transmitted to the data source until you call a Recordset Update or UpdateBatch method.

Although all of the providers support locking with read-only access, locking with update access is available only for the SAS/SHARE, IOM, and Base SAS providers. This recipe applies only to ADO. Sample code is included.

Implementation

Locking Records with Read-Only Access

By default, ADO locks open Recordset objects and sets the lock type to adLockReadOnly. When your application does not need update access, you can use the following code, which does not explicitly set the lock type:

Note: The local and OLAP providers support only read access.

```

' obConnection is an open Connection object.
Dim obRecordset As New ADODB.Recordset

' Do not set the lock type parameter. Let it default to adLockReadOnly.
obRecordset.Open "mylib.standard", obConnection, , , adCmdTableDirect

```

Locking Records During Concurrent Updating

If your application needs to support multiple users who can modify the same record concurrently, then use the `adLockPessimistic` lock type. This lock type is the most stringent lock type available. It locks the record from the time editing begins until an `Update` or `MoveNext` method is called. The `Recordset` `Open` method must specify the `adCmdTableDirect` option, and the `CursorType` property must be set to `adOpenStatic` or `adOpenDynamic`. The following Visual Basic code shows how this task is done:

```

' obConnection is an open Connection object.
Dim obRecordset As New ADODB.Recordset

obRecordset.LockType = adLockPessimistic
obRecordset.CursorType = adOpenDynamic
obRecordset.Open "mylib.standard", obConnection, , , adCmdTableDirect

Dim lt As ADODB.LockTypeEnum
lt = obRecordset.LockType

' Check the lock type to make sure the Recordset can be updated.
' Then, lock the first record in the underlying data source
' and modify the fields in the local record buffer.

If (lt <> adLockReadOnly) Then
    rs!age=19
    rs!sat=808
    rs!degree="BS"
    rs!homest="NC"
End If

' The record is locked, updated, and then unlocked.
obRecordset.Update

```

Locking Records for a Minimal Amount of Time

When you want to minimize the amount of time a record is locked or if updates will be made infrequently, you can use the `adLockOptimistic` lock type. This lock type does not lock the open `Recordset` until the `Update` method is called. When `Update` is called, the provider compares the record in the local record buffer against the underlying data source to ensure that it has not been changed by another process. If a record has been changed, the update will fail. The `Recordset` `Open` method must specify the `adCmdTableDirect` option and that `CursorType` property must be set to `adOpenStatic` or `adOpenDynamic`. The following Visual Basic code shows how this task is done:

```

Dim obConnection As ADODB.Connection
Dim obRecordset As New ADODB.Recordset

obRecordset.Open "mylib.standard", obConnection, adOpenDynamic, adLockOptimistic, adCmdTableDirect

' Modify the fields in the local buffer for the first record.
' The record is NOT locked in the underlying data set.

```

```

rs!age=19 'record is now locked
rs!sat=808
rs!degree="BS"
rs!homest="NC"

' The record is locked, updated, and then unlocked.
obRecordset.Update

```

Locking Records in Order to Improve Performance

Batch updating can improve the performance of your application. SAS permits batch updates when the data set has been opened with an exclusive member lock. To enable an exclusive member lock, you set the customized Recordset property SAS Optimistic Locking to **False**, as shown in the sample code.

After granting exclusive member-level access to one user, you can use the `adLockBatchOptimistic` lock type, which does not lock the open Recordset until the `UpdateBatch` method is called. When `UpdateBatch` is called, the provider compares each record in the local buffer against the underlying data source to ensure that it has not been changed by another process. If a record has been changed, the batch update will fail. The Recordset Open must specify the `adCmdTableDirect` option, and the `CursorType` property must be set to `adOpenStatic` or `adOpenDynamic`. The following Visual Basic code shows how this task is done:

```

' obConnection is an open Connection object.
Dim obRecordset As New ADODB.Recordset

' Set exclusive member access.
obRecordset.ActiveConnection = obConnection
obRecordset.Properties("SAS Optimistic Locking") = False

' Set the maximum number of rows for batch update.
obRecordset.Properties("Maximum Open Rows") = 1000
obRecordset.LockType = adLockBatchOptimistic
obRecordset.CursorType = adOpenStatic
obRecordset.Open "mydata.customers", , , , adCmdTableDirect

' Change all of the records.
While Not obRecordset.EOF
    rs!balance = 0
    obRecordset.MoveNext
Wend

' The record is locked, updated, and then unlocked.
obRecordset.UpdateBatch

```

Note: If a data set has not been opened with an exclusive member lock, then the `UpdateBatch` method ignores the batch request and updates the records one at a time, and there will be no performance gain.

Update Restrictions

A SAS data set can be opened concurrently for update access only from within a single SAS process. If one SAS process opens a data set for an update, all other processes are restricted to read-only access of that data set. This restriction applies to the IOM provider and supersedes any explicitly set Recordset properties. Therefore, if your application spans multiple SAS processes, you will not be able to open one data set

concurrently even if you specify an ADO lock type of `adLockPessimistic`, `adLockOptimistic`, or `adLockBatchOptimistic`.

This restriction does not apply to the SAS/SHARE and Base SAS providers because multiple requests to open the same data set are serviced by the same SAS process.

Part 5

Tips and Best Practices

<i>Chapter 12</i>	
<i>Tuning the Providers for Performance</i>	<i>119</i>
<i>Chapter 13</i>	
<i>Writing Code That Returns Provider Information</i>	<i>123</i>

Chapter 12

Tuning the Providers for Performance

Properties That Affect Performance	119
How the "CacheSize" Property Affects Performance	120
How the "Maximum Open Rows" Property Affects Performance	120
How the SAS Page Size Property Affects Performance	121
How the SAS Data Set Options Property Affects Performance	122

Properties That Affect Performance

The SAS providers support the following ADO and OLE DB properties that affect the performance of your application.

Table 12.1 Property Names and Supported Providers

ADO Property Name	OLE DB Property ID	Support
"CacheSize"	none	Supported in ADO by all providers
"Maximum Open Rows"	DBPROP_MAXOPENROWS	Supported by all providers
"SAS Page Size"	DBPROP_SAS_PAGESIZE	Supported by the IOM provider
"SAS Data Set Options"	DBPROP_SAS_DATASETOPTS	Supported by the IOM provider

These properties determine how much data is retrieved and cached at different architectural levels. They will affect performance for better or for worse, depending on how you use them in your application. For example, significant performance improvement occurs when each accessed row is relatively close to the previously accessed row. However, if your application randomly accesses rows or only accesses a few rows, then increased data caching might degrade performance. To optimize performance in this case, you can take the following two steps:

- If you have enough RAM to hold the entire data set in memory and your application accesses most of the rows in the data set, then increase the amount of data cached to match the amount of data in the data set.

- If you do not have enough RAM or your application needs to access only a few rows in the data set, then lower the amount of cached data to minimize the transfer of unused rows.

TIP When you develop your application, you should test different values for these properties to determine which values will result in the best performance for your target environment.

Note: The IOM provider also supports optimized accessors as described in the *OLE DB Programmer's Reference and Data Access SDK*. This feature enables OLE DB consumers to specify which columns should be included in the provider's cache.

TIP The sample code in this chapter assumes that an ADO Connection object is open. For more information, see “[Basic Connection Recipes](#)” on page 37.

How the "CacheSize" Property Affects Performance

The Recordset object property "CacheSize" indicates how many rows should be cached in the ADO local memory. The following code shows how to set this property:

```
' obConnection is an open Connection object.
Dim obRecordset As New ADODB.Recordset

obRecordset.CacheSize = 55
obRecordset.Open "seashell.mediate", obConnection, adOpenDynamic, adLockOptimistic, adCmdTableDirect
```

How the "Maximum Open Rows" Property Affects Performance

The "Maximum Open Rows" property, which is supported by all of the providers, determines how many rows can be active at any one time. This property affects performance because of the way it interacts with ADO and the "CacheSize" property.

ADO Recordset objects do not release open rows held in memory until they request more data from the data provider. All data requests performed after the first request can retrieve only "Maximum Open Rows" minus "CacheSize" rows. For example, if you create an application that iterated in sequence through each recordset row in a cache and you set the "Maximum Open Rows" value to 10 and the CacheSize value to 8, performance would be affected in the following ways:

1. On the first request for data, ADO requests eight rows from the provider.
2. After the application iterates through those eight rows, ADO initiates its next request for eight rows. However, because "Maximum Open Rows" is set to 10 and ADO still has eight rows open, the provider returns only two rows.
3. When the two rows are returned, ADO releases the first two rows in its local cache, returning the number held in memory to eight.

Although this behavior does not affect the ability of an application to iterate through a recordset, it might eliminate the performance gain that would otherwise be expected. In order for the Recordset object to retrieve the number of rows indicated by the value of

"CacheSize", you should set "Maximum Open Rows" to at least twice the value of "CacheSize". The following code shows how this task can be done:

```
' obConnection is an open Connection object.
Dim obRecordset As New ADODB.Recordset

' Set the active connection here instead of on the Open method.
obRecordset.ActiveConnection = obConnection
obRecordset.CacheSize = 55
obRecordset.Properties("Maximum Open Rows") = 110

If InStr(LCase(obConnection.Provider), "iomprovider") <> 0 Then
  obRecordset.Properties("SAS Page Size") = 55
End If

obRecordset.Open "seashell.mediate", , adOpenDynamic, adLockOptimistic, adCmdTableDirect
```

How the SAS Page Size Property Affects Performance

The SAS Page Size property, which is supported only by the IOM provider, determines how many rows will be retrieved from the SAS Workspace Server at one time. It serves a function similar to the "CacheSize" property. However, the SAS Page Size property determines how much data is cached in the provider itself rather than in the Recordset object.

There are a few special values for the SAS Page Size property.

- If the SAS Page Size property is set to -1, then the provider attempts to cache the entire data set on its first request for data. Your application must check that there is enough RAM for this operation to succeed.
- If the SAS Page Size property is set to 0, then caching is disabled.

```
' obConnection is an open Connection object.
Dim obRecordset As New ADODB.Recordset

' Set the active connection here instead of on the Open method.
obRecordset.ActiveConnection = obConnection
obRecordset.CacheSize = 55
obRecordset.Properties("Maximum Open Rows") = 110

If InStr(LCase(obConnection.Provider), "iomprovider") <> 0 Then
  obRecordset.Properties("SAS Page Size") = 55
End If

obRecordset.Open "seashell.mediate", , adOpenDynamic, adLockOptimistic, adCmdTableDirect
```

How the SAS Data Set Options Property Affects Performance

The SAS Data Set Options property is supported only by the IOM Provider. This property can be used to set a variety of data set options that affects how the SAS Workspace Server accesses a data set. This property is similar to the SAS Page Size property in that the property affects the SAS Workspace Server rather than affecting the client. One performance-related use of this property is to set the number of rows that the SAS Workspace Server retrieves from a data set. Using the OBS= data set option improves performance by directing the SAS Workspace Server to subset data when retrieving data. For more information about SAS data set options, see “SAS Data Set Options” in *SAS Language Reference: Dictionary*.

The following code shows how to use the SAS data set options with the OBS= data set option:

```
' obConnection is an open Connection object.
Dim obRecordset As New ADODB.Recordset

' Set the active connection here instead of on the Open method.
obRecordset.ActiveConnection = obConnection

If InStr(LCase(obConnection.Provider), "iomprovider") <> 0 Then
  ' Limit results to 250 rows
  obRecordset.Properties("SAS Data Set Options") = "obs=250"
  obRecordset.Open "seashell.mediate", , adOpenForwardOnly, adLockReadOnly, adCmdTableDirect

End If
```

Chapter 13

Writing Code That Returns Provider Information

How to Generate a List of Supported ADO Properties	123
How to Retrieve Version Information for a Provider	125
Retrieve Version Information for the Provider Implementation	125
Retrieve Version Information for a Server	126
Retrieve Version Information for the ADO Interface	126
Sample Code to Retrieve Version Information for a Provider, a Server, and the ADO Interface	126

How to Generate a List of Supported ADO Properties

The following sample code displays a list of ADO Connection properties and their defaults in the VB Immediate window. The example uses the latest version of the IOM provider, as indicated by the value of the Connection object's Provider property. To specify a different provider, replace "sas.IOMProvider.9.3" with the appropriate ProgID (see [“How to Identify the SAS Providers”](#) on page 26).

```
Option Explicit
Sub Main()
    Dim obConnection As New ADODB.Connection
    Dim obRecordset As New ADODB.Recordset
    Dim p As ADODB.Property
    obConnection.Provider = "SAS.IOMProvider.9.3"
    For Each p In obConnection.Properties
        Debug.Print p.Name & " = " & p.Value
    Next p
End Sub
```

Log 13.1 List of Properties for the IOM Provider

```

Password =
User ID =
Asynchronous Processing = 0
Data Source =
Window Handle = 0
Locale Identifier = 0
Location =
Mode = 19
Prompt = 4
Extended Properties =
SAS Cell Cache Size = 10000
SAS Logical Name =
SAS Machine DNS Name =
SAS Port = 0
SAS Protocol = 0
SAS Server Type = 1
SAS Service Name =
SAS Workspace Interface =
SAS Workspace ID =
SAS Workspace Init Script =
SAS Metadata User ID =
SAS Metadata Password =
SAS Metadata Location =
SAS Repository ID =
SAS Repository Name =

```

Log 13.2 List of Properties for the OLAP Provider

```

Password =
User ID =
Asynchronous Processing = 0
Data Source =
Window Handle = 0
Locale Identifier = 0
Location =
Mode = 19
Prompt = 4
Extended Properties =
SAS Cell Cache Size = 10000
SAS Logical Name =
SAS Machine DNS Name =
SAS Port = 5451
SAS Protocol = 2
SAS Server Type = 2
SAS Service Name =
SAS Workspace Interface =
SAS Workspace ID =
SAS Workspace Init Script =
SAS Metadata User ID =
SAS Metadata Password =
SAS Metadata Location =
SAS Repository ID =
SAS Repository Name =

```

Log 13.3 List of Properties for the Local Provider

```

Password =
User ID =
Data Source =
Window Handle = 0
Location =
Mode = 19
Prompt = 4
Extended Properties =
SAS File Format =

```

Log 13.4 List of Properties for the SAS/SHARE and Base SAS Providers

```

Password =
User ID =
Data Source =
Window Handle = 0
Location =
Mode = 19
Prompt = 4
Extended Properties =
SAS Local Server = 0
SAS Executable = C:\Program Files\SASHome\SASFoundation\9.3\sas.exe
SAS Parameters = -initstmt %sasodbc(sdplsrv) -icon -nologo -notutorialdlg
SAS Working Directory = C:\Program Files\SASHome\SASFoundation\9.3\
SAS Server Access Password =
SAS Server Release = 9

```

How to Retrieve Version Information for a Provider

Retrieve Version Information for the Provider Implementation

You can use either of these methods in order to retrieve version information for the provider implementation:

- After an ADO Connection object is open, use the "Provider Version" property from the Connection object Properties collection. The version appears in the form *MM.mm.rr.bbbb* where *MM* is the major release number, *mm* is the minor release number, *rr* is the revision number, and *bbbb* is the build number.
- If a Connection object is not open, then use the provider's .DLL file. Here are the steps:
 1. In Windows Explorer, navigate to your **C:\Program Files\SAS\SASProvidersForOLEDB\9.3** folder. The 32-bit versions of the providers are in the **32-bit** directory. On 64-bit machines, the 64-bit versions are in the **64-bit** directory.
 2. Right-click on the .DLL file associated with the provider that you want to check. Here are provider .DLL files for SAS 9.3:
 - sasaorio.dll for the IOM provider and the OLAP provider
 - sasafloc.dll for the local data provider

- sasafshr.dll for the SAS/SHARE provider and the Base SAS provider
3. Select **Properties** from the pop-up menu.
 4. In the Properties dialog box, click the **Version** tab to see the version number.

Retrieve Version Information for a Server

After an ADO Connection object is open, use the "DBMS Version" property in the Connection object Properties collection. The version appears in the form *MM.mm.rrrrfMMDDYY* where *MM* is the major release number, *mm* is the minor release number, *rrrr* is the maintenance release number, *f* is for Performance Image, and *MMDDYY* is the port date.

Note: The local provider does not have an associated server. If you request the server property from the local provider, it returns the provider implementation version.

Retrieve Version Information for the ADO Interface

After an ADO Connection object is open, use the Connection object "Version" property.

Sample Code to Retrieve Version Information for a Provider, a Server, and the ADO Interface

The following Visual Basic code shows you how to retrieve version information for a provider, a server, and the ADO interface. The code assumes that an ADO Connection object is open.

```
Dim strProviderVersion As String
Dim strServerVersion As String
Dim strADOVersion As String
If obConnection.State = adStateOpen Then
    strProviderVersion = obConnection.Properties("Provider Version")
    strServerVersion = obConnection.Properties("DBMS Version")
    strADOVersion = obConnection.Version
End If
```

Part 6

Troubleshooting

<i>Chapter 14</i>	
Handling Error Objects	129
<i>Chapter 15</i>	
Known Issues	133

Chapter 14

Handling Error Objects

Using ADO to Handle Errors	129
Using OLE DB to Handle Errors	131

Using ADO to Handle Errors

The OLE DB error objects that are returned by the SAS providers are presented in the ADO interface as Error objects. Because it is possible for more than one error to be returned, the errors are stored in the Errors Collection. To retrieve complete error information, you iterate through the Errors Collection and output the information. In most cases, the "Description" and "Number" properties of the Error object provide the most helpful information.

The Visual Basic, VBScript, and Active Server Pages (ASP) examples all show how errors can be handled by using ADO. The VBScript and ASP examples perform the same tasks as the Visual Basic example, but there are language differences between them. When reviewing the sample code, keep the following differences in mind:

- An ASP file typically embeds VBScript and delimits the VBScript with `<%` and `%>`.
- VBScript does not predefine the ADO Enumerated Constants, so you must code them as constants.
- In VBScript, you should create objects by using the `CreateObject` syntax.
- VBScript usually provides different output than Visual Basic.
- Usually, Visual Basic programmers handle errors by using the `On Error Goto` syntax. However, this syntax is not supported by VBScript, so the VBScript and ASP examples use a subroutine to handle error output.

Note: Although the examples use the `Debug.Print` and `MsgBox` methods to display error information, applications can use other methods such as `Document.Write`, or `Response.Write`. You can also write an application that traps errors instead of displaying error information.

The following examples use the local provider to attempt to open a table named `lostDataset`.

If `lostDataset` does not exist in the directory `c:\testdata`, then the application generates an error and outputs the error information. The "Description" property used in the examples is typically a string message that is composed by the SAS provider. However, it might also be a message from the server or some component that is used by the provider. The "Number" property is typically the `HRESULT` value that is returned by

the underlying OLE DB interface method. In the examples, the value is output in hexadecimal because that is the way that it is typically written in the header files such as WinError.h and OleDBErr.h.

Example Code 14.1 Visual Basic Error Handling

```
Sub Main()
    Dim obConnection As New ADODB.Connection
    Dim obRecordset As New ADODB.Recordset
    Dim errorObject As ADODB.Error

    On Error GoTo DisplayErrorInfo
    obConnection.Provider = "sas.LocalProvider"
    obConnection.Properties("Data Source") = "c:\testdata"

    obConnection.Open
    obRecordset.Open "lostDataset", obConnection, adOpenDynamic, adLockOptimistic, ADODB.adCmdTableDirect
    obRecordset.Close
    obConnection.Close

DisplayErrorInfo:
    For Each errorObject In obRecordset.ActiveConnection.Errors
        Debug.Print "Description :"; errorObject.Description
        Debug.Print "Number: "; Hex(errorObject.Number)
    Next
End Sub
```

Example Code 14.2 VBScript Error Handling

```
<HTML>
Example showing error handling by using VBScript.

<SCRIPT LANGUAGE=VBSCRIPT>
Const adOpenDynamic = 2
Const adLockOptimistic = 3
Const adCmdTableDirect = 512

Set obConnection = CreateObject("ADODB.Connection")
Set obRecordset = CreateObject("ADODB.Recordset")
Set errorObject = CreateObject("ADODB.Error")

On Error Resume Next
obConnection.Provider = "sas.LocalProvider"
obConnection.Properties("Data Source") = "c:\testdata"

obConnection.Open
obRecordset.Open "lostDataset", obConnection, adOpenDynamic, adLockOptimistic, adCmdTableDirect
DisplayErrorInfo
obRecordset.Close
obConnection.Close

sub DisplayErrorInfo()
    For Each errorObject In obRecordset.ActiveConnection.Errors
        MsgBox "Description: " & errorObject.Description & Chr(10) & Chr(13) & _
            "Number: " & Hex(errorObject.Number)
    Next
End Sub
```

```
</SCRIPT>
</HTML>
```

Example Code 14.3 ASP Error Handling

```
<HTML>
Example showing error handling by using VBScript.

<%
Const adOpenDynamic = 2
Const adLockOptimistic = 3
Const adCmdTableDirect = 512

Set obConnection = Server.CreateObject("ADODB.Connection")
Set obRecordset = Server.CreateObject("ADODB.Recordset")
Set errorObject = Server.CreateObject("ADODB.Error")

On Error Resume Next
obConnection.Provider = "sas.LocalProvider"
obConnection.Properties("Data Source") = "c:\testdata"

obConnection.Open
obRecordset.Open "lostDataset", obConnection, adOpenDynamic, adLockOptimistic, adCmdTableDirect
DisplayErrorInfo
obRecordset.Close
obConnection.Close

sub DisplayErrorInfo()
  For Each errorObject In obRecordset.ActiveConnection.Errors
    Response.Write "Description: " & errorObject.Description & Chr(10) & Chr(13) & _
      "Number: " & Hex(errorObject.Number)
  Next
End Sub
%>
</SCRIPT>
</HTML>
```

Using OLE DB to Handle Errors

The SAS providers support OLE DB error objects, which are extensions of automation error objects. Most interfaces implemented by the providers return error objects. For any component, use the `ISupportErrorInfo` interface to determine which interfaces on the component return error objects.

See Also

[“Standard OLE DB Interfaces” on page 216](#)

Chapter 15

Known Issues

Known Issues for All Providers	133
Known Issues for the IOM Provider	134
Known Issues for the Local Provider	135
Known Issues for the SAS/SHARE Provider	136

Known Issues for All Providers

Some ADO properties are missing in Active Server Pages (ASP) under Microsoft IIS 5.0.

When a server-side cursor is used, the following two properties cannot be accessed through the ADO interface from an ASP page:

- "Literal Row Identity"
- "IRowsetUpdate"

The "Literal Row Identity" property is read-only and therefore has a minor effect on the functionality of the providers. The value of the "Literal Row Identity" property is always **VARIANT_TRUE**.

The absence of the "IRowsetUpdate" property, however, can result in unexpected behavior. By default, the "IRowsetUpdate" property is **VARIANT_FALSE**, which means that recordsets are opened in immediate update mode. When recordsets are opened in immediate update mode, batch updating (via the UpdateBatch method) is not available when a server-side cursor is used.

TIP "Literal Row Identity" and "IRowsetUpdate" are available in Visual Basic applications and in ASP pages when you use a client-side cursor. For more information about cursor types and locations, see [“Working with Cursor and Lock Type Combinations” on page 139](#).

SAS customized properties cannot be accessed when you use an ADO client-side cursor with a Recordset or Connection object.

If your application uses an ADO client-side cursor, you see the following error when the application attempts to access SAS customized properties: Item cannot be found in the collection corresponding to the requested name or ordinal. This problem applies to the ADO Connection object and the ADO Recordset object. It occurs because ADO does not request information about properties from the provider when a client-side cursor is used.

To enable the application to access SAS customized properties, perform one of these tasks:

- Use a server-side cursor location instead of a client-side cursor location.
- Set the cursor location to server-side in order to access the SAS customized properties, and then set the cursor location to client-side.

For more information about cursor types and locations, see “[Working with Cursor and Lock Type Combinations](#)” on page 139.

Visual Basic DataBound Controls do not perform reliably with the SAS providers for OLE DB.

There is no resolution available. The data binding features of the Visual Basic DataBound Controls cannot be used reliably with the SAS providers for OLE DB. Do not use these controls with the providers.

Known Issues for the IOM Provider

The ADO Field object "DefinedSize" property must be multiplied by 2.

For character fields, the "DefinedSize" property is set to the maximum number of characters defined for that field. Because the IOM provider returns character data as wide character strings, the size of this field in bytes is twice the value of "DefinedSize." To get an accurate size in bytes, you must multiply "DefinedSize" by 2.

Note: The "ActualSize" property correctly returns the length in bytes.

The ADO Recordset "MaxRecords" property is not available.

There is no resolution to this issue. The SAS IOM provider does not currently implement the ADO "MaxRecords" property.

Accessing a local instance of the IOM provider in an ASP page that runs under Windows 2000 Professional returns EOF.

When ASP pages that are served from a machine running Windows 2000 Professional attempt to connect to a SAS Workspace Server on the same machine by using COM, all Recordset objects will report EOF immediately after being opened as if the recordsets contain no rows. This behavior occurs only under Windows 2000 Professional. You do not encounter this problem when the ASP page server and the SAS Workspace Server are on different machines.

As a workaround, use the dcomcnfg tool to change the "Authentication Level" of the "SAS: IOM DCOM Servers" to "(None)". For more information about using dcomcnfg, see **Turning Off Call Security** in the Microsoft Developer Network Library at <http://msdn2.microsoft.com/en-us/library/ms687216.aspx>.

Move methods fail if you use them to access V7 compressed data sets.

The following ADO Recordset methods will fail with Version 7 compressed data sets when the number of records to move is nonzero: Move, MoveNext, MoveFirst, and MovePrevious. In this situation, these methods result in random access operations, which are not supported in compressed SAS data sets before Version 8.

In SAS Version 8, when a compressed SAS data set is created with the options POINTOBS=YES and RESUSE=NO (the default values), the data set can be accessed

randomly, and the Move, MoveNext, MoveFirst, and MovePrevious methods are successful.

Known Issues for the Local Provider

Numeric variables are not converted correctly when you use the File Format property and access SAS data sets on platforms other than Windows.

When you use the OLE DB property DBPROP_SAS_INIT_FILEFORMAT (the ADO property "SAS File Format") and specify a value of "v8", the local provider does not properly convert numeric variables for SAS data sets that are created on platforms other than Windows. In this case, the local provider returns incorrect data for the numeric variable (although it does not return an error).

To ensure that numeric variables are converted properly for SAS data sets created on platforms other than Windows, use the default file format or "v9".

For more information about cross-platform support, see ["Data Sources and File Types Supported by the SAS Providers"](#) on page 5.

ADO Recordsets that are opened with a client-side cursor incorrectly appear to support updating.

If you use the read-only local provider to open a Recordset object with a server-side cursor, ADO attempts to read the values of the DBPROP_IRowsetUpdate and DBPROP_IRowsetChange properties. If both of these properties are false, the following Visual Basic statements return **FALSE**:

- Recordset.Properties("IRowsetUpdate")
- Recordset.Properties("IRowsetChange")
- Recordset.Supports(adUpdate)
- Recordset.Supports(adUpdateBatch)

In this case, any attempt to update the Recordset results in this error message: "Object or provider is not capable of performing requested operation." This response is consistent with a read-only provider.

However, when a Recordset object is opened with a client-side cursor, ADO does not ask the provider for the values of DBPROP_IRowsetChange and DBPROP_IRowsetUpdate. In this case, all four of the Visual Basic statements in the previous list incorrectly return **TRUE**. This return value makes it appear as if the opened recordset can be updated. If you attempt to perform the update, you see this error message: "Multiple-step operation generated errors. Check each status value."

The OLE DB Rowset RestartPosition method and the ADO Recordset MoveFirst method are not supported for transport files.

If you use the local provider to read transport files, you encounter a movement limitation. The engine used to read transport files is a forward-only, sequential single pass, which means that you can make only one pass through the file. As a result, the OLE DB Rowset RestartPosition method and the ADO Recordset MoveFirst method are not supported for the transport file format. In order to reposition the cursor at the first row, you must complete one of these tasks:

- When using OLE DB, you must close and then reopen the file.
- When using ADO, you must close and then reopen the Recordset.

Reopening a connection fails on Windows 7.

When an ADO connection object is opened, closed, and then reopened, the connection fails. SAS has noticed that the "Data Source" property is cleared when a connection object is closed in a Windows 7 operating environment.

To reuse a connection object, set the user-specified properties again before calling the open method.

```
Dim obConnection As New ADODB.Connection
obConnection.Provider = "sas.LocalProvider"
obConnection.Properties("Data Source") = "C:\v9data"
obConnection.Properties("SAS File Format") = "V9"
obConnection.Open()
...
obConnection.Close()
' Set the connection properties again before the call to Open()
obConnection.Properties("Data Source") = "C:\v9data"
obConnection.Properties("SAS File Format") = "V9"
obConnection.Open()
...
obConnection.Close()
```

Known Issues for the SAS/SHARE Provider

Reopening a connection fails on Windows 7.

When an ADO connection object is opened, closed, and then reopened, the connection fails. SAS has noticed that the "Data Source" property is cleared when a connection object is closed in a Windows 7 operating environment.

To reuse a connection object, set the user-specified properties again before calling the open method.

```
Dim obConnection As New ADODB.Connection
obConnection.Provider = "sas.ShareProvider"
obConnection.Properties("Data Source") = "C:\v9data"
obConnection.Properties("SAS File Format") = "V9"
obConnection.Open()
...
obConnection.Close()
' Set the connection properties again before the call to Open()
obConnection.Properties("Data Source") = "C:\v9data"
obConnection.Properties("SAS File Format") = "V9"
obConnection.Open()
...
obConnection.Close()
```

Part 7

Appendixes

<i>Appendix 1</i>	
ADO: Supported Cursor and Lock Type Combinations	139
<i>Appendix 2</i>	
ADO: Supported Methods and Properties	143
<i>Appendix 3</i>	
OLE DB Properties	147
<i>Appendix 4</i>	
OLE DB Interfaces	215
<i>Appendix 5</i>	
Schema Rowsets	229
<i>Appendix 6</i>	
OLE DB: Format Processing	253
<i>Appendix 7</i>	
OLE DB: Column Mapping and Binding	263
<i>Appendix 8</i>	
Customized User Help for the Data Link Properties Dialog Box . .	269

Appendix 1

ADO: Supported Cursor and Lock Type Combinations

Working with Cursor and Lock Type Combinations	139
Server-Side Cursor Combinations	139
Client-Side Cursor Combinations	141

Working with Cursor and Lock Type Combinations

The cursor and lock type combinations that are supported by the SAS providers depend on whether the cursor is located on the client side or the server side. If you request a combination that is not supported, then the providers will attempt to use the supported combination that most closely matches your choice. After the recordset is opened, verify the cursor and lock type.

When you specify properties and cursor location on a recordset, you must specify the properties after the cursor location is set. Certain properties can be specified only for the server. If the cursor location is specified after the properties, you might not see an error and you might have unexpected results. For more information, see the discussion of SAS customized properties in [“Known Issues for All Providers” on page 133](#).

Note: There are no recordsets in ADO MD, so this information does not apply to the OLAP provider.

See Also

[“Implementing a Locking Strategy” on page 113](#)

Server-Side Cursor Combinations

To set a server-side cursor location, you can use this line of code:

```
obRecordset.CursorLocation = adUseServer
```

The following tables list the server-side cursor combinations that are supported for the local, IOM, SAS/SHARE, and Base SAS providers.

Table A1.1 Server-Side Cursor Combinations Supported by the Local Provider

Cursor Type	Lock Type	Supported Features
adOpenDynamic	adLockReadOnly	adBookmark, adFind, adHoldRecords, adMovePrevious
adOpenForwardOnly	adLockReadOnly	adFind, adHoldRecords

Table A1.2 Server-Side Cursor Combinations Supported by the IOM Provider

Cursor Type	Lock Type	Supported Features
adOpenDynamic	adLockBatchOptimistic	adAddNew, adBookmark, adDelete, adFind, adHoldRecords, adMovePrevious, adUpdate, adUpdateBatch
	adLockOptimistic	adAddNew, adBookmark, adDelete, adFind, adHoldRecords, adMovePrevious, adUpdate, adUpdateBatch
	adLockPessimistic	adAddNew, adBookmark, adDelete, adFind, adHoldRecords, adMovePrevious, adUpdate, adUpdateBatch
	adLockReadOnly	adBookmark, adFind, adHoldRecords, adMovePrevious
adOpenForwardOnly	adLockBatchOptimistic	adAddNew, adDelete, adFind, adHoldRecords, adUpdate, adUpdateBatch
	adLockOptimistic	adAddNew, adDelete, adFind, adHoldRecords, adUpdate, adUpdateBatch
	adLockPessimistic	adAddNew, adDelete, adFind, adHoldRecords, adUpdate, adUpdateBatch
	adLockReadOnly	adFind, adHoldRecords

Table A1.3 Server-Side Cursor Combinations Supported by the SAS/SHARE and Base SAS Providers

Cursor Type	Lock Type	Supported Features
adOpenDynamic	adLockBatchOptimistic	adAddNew, adBookmark, adDelete, adFind, adHoldRecords, adMovePrevious, adUpdate, adUpdateBatch
	adLockOptimistic	adAddNew, adBookmark, adDelete, adFind, adHoldRecords, adMovePrevious, adUpdate, adUpdateBatch
	adLockReadOnly	adBookmark, adFind, adHoldRecords, adMovePrevious

Cursor Type	Lock Type	Supported Features
adOpenForwardOnly	adLockBatchOptimistic	adAddNew, adDelete, adFind, adHoldRecords, adUpdate, adUpdateBatch
	adLockOptimistic	adAddNew, adDelete, adFind, adHoldRecords, adUpdate, adUpdateBatch
	adLockReadOnly	adFind, adHoldRecords

Client-Side Cursor Combinations

To set a client-side cursor location, you can use this line of code:

```
obRecordset.CursorLocation = adUseClient
```

The only client-side combination that is supported by the SAS providers is adOpenStatic and adLockReadOnly. This combination supports adApproxPosition, adBookmark, adFind, adHoldRecords, adMovePrevious, adNotify, and adResync.

Appendix 2

ADO: Supported Methods and Properties

The following table identifies which methods and properties of ADO objects are supported and which are not supported by the SAS providers.

Note: The SAS providers do not support the ICommandWithParameters interface, so the Parameter object and Parameter collection are not supported and are not included in the table.

* MDX queries with the OLAP provider can be cancelled if the query is made asynchronously with the Execute method.

	SUPPORTED		UNSUPPORTED	
ADO Object	Methods	Properties	Methods	Properties
Connection	Close ConnectionString Open Execute OpenSchema	Attributes CursorLocation Mode Provider State	BeginTrans Cancel * CommitTrans RollbackTrans	CommandTimeout ConnectionTimeout DefaultDatabase IsolationLevel
Command	Execute	ActiveConnection CommandText CommandType Name Prepared State	Cancel * CreateParameter	CommandTimeout

	SUPPORTED		UNSUPPORTED	
ADO Object	Methods	Properties	Methods	Properties
Recordset	AddNew CacheSize CancelBatch CancelUpdate Close Clone Delete Find GetRows GetString Move MoveFirst MoveLast MoveNext MovePrevious Open Requery Save Supports Update UpdateBatch	AbsolutePage AbsolutePosition BOF Bookmark CursorLocation CursorType Data Source EditMode EOF Filter LockType MaxRecords PageCount PageSize RecordCount Sort State	Cancel CompareBookmarks NextRecordset Resync Seek	ActiveCommand DataMember Index MarshalOptions Status StayInSync Source
Error		Description Number Source		HelpContext HelpFile NativeError SQLState
Errors Collection	Clear Refresh	Count Item		
Field		ActualSize Attributes DefinedSize Name NumericScale Precision Type Value	AppendChunk GetChunk	DataFormat OriginalValue UnderlyingValue

	SUPPORTED		UNSUPPORTED	
ADO Object	Methods	Properties	Methods	Properties
Fields Collection	Append Refresh	Count Item	Delete	
Property		Attributes Name Type Value		
Properties Collection	Refresh	Count Item		

Appendix 3

OLE DB Properties

OLE DB Properties: Introduction	149
What Are OLE DB Properties?	149
Supported Property Groups	150
Supported Property Sets	150
Property-Related Methods	152
OLE DB Properties: Descriptions	153
DBPROP_APPENDONLY	153
DBPROP_AUTH_PASSWORD	153
DBPROP_AUTH_USERID	154
DBPROP_BOOKMARKS	154
DBPROP_BOOKMARKSKIPPED	155
DBPROP_BOOKMARKTYPE	155
DBPROP_BYREFACCESSORS	155
DBPROP_CANFETCHBACKWARDS	156
DBPROP_CANHOLDROWS	156
DBPROP_CANSROLLBACKWARDS	156
DBPROP_CLIENTCURSOR	157
DBPROP_CURRENTCATALOG	157
DBPROP_DATASOURCEREADONLY	157
DBPROP_DATASOURCE_TYPE	158
DBPROP_DBMSNAME	158
DBPROP_DBMSVER	158
DBPROP_IAccessor	159
DBPROP_IColumnsInfo	159
DBPROP_IConvertType	160
DBPROP_INIT_ASYNC	160
DBPROP_INIT_DATASOURCE	160
DBPROP_INIT_HWND	161
DBPROP_INIT_LCID	161
DBPROP_INIT_LOCATION	162
DBPROP_INIT_MODE	162
DBPROP_INIT_PROMPT	163
DBPROP_INIT_PROVIDERSTRING	163
DBPROP_IRowset	163
DBPROP_IRowsetChange	164
DBPROP_IRowsetIdentity	164
DBPROP_IRowsetInfo	164
DBPROP_IRowsetLocate	165
DBPROP_IRowsetUpdate	165
DBPROP_IRowsetView	165
DBPROP_ISupportErrorInfo	166

DBPROP_ IViewFilter	166
DBPROP_ IViewRowset	166
DBPROP_ IViewSort	167
DBPROP_ LITERALBOOKMARKS	167
DBPROP_ LITERALIDENTITY	167
DBPROP_ LOCKMODE	168
DBPROP_ MAXOPENROWS	168
DBPROP_ MAXORSINFILTER	168
DBPROP_ MAXPENDINGROWS	169
DBPROP_ MAXROWS	169
DBPROP_ MAXSORTCOLUMNS	169
DBPROP_ MSMD_MDX_CALCMEMB_EXTENSIONS	170
DBPROP_ ORDEREDBOOKMARKS	170
DBPROP_ OTHERINSERT	170
DBPROP_ OTHERUPDATEDELETE	171
DBPROP_ OWNINSERT	171
DBPROP_ OWNUPDATEDELETE	171
DBPROP_ PROVIDERVER	172
DBPROP_ REMOVEDLETED	172
DBPROP_ SAS_ALTERPASSWORD	172
DBPROP_ SAS_BLANKPADDING	173
DBPROP_ SAS_DATASET_ENCODING	173
DBPROP_ SAS_DATASET_LABEL	173
DBPROP_ SAS_DATASET_TYPE	174
DBPROP_ SAS_DATASET_WINDOWS_CODEPAGE	174
DBPROP_ SAS_DATASETOPTS	174
DBPROP_ SAS_DEFAULTFILEFORMAT	175
DBPROP_ SAS_ENGINE	175
DBPROP_ SAS_FMTERR	176
DBPROP_ SAS_FORMATS	176
DBPROP_ SAS_GET_MISSING_VALUES	177
DBPROP_ SAS_INFORMATS	177
DBPROP_ SAS_INIT_CELLCACHESIZE	178
DBPROP_ SAS_INIT_FILEFORMAT	178
DBPROP_ SAS_INIT_LOCALSERVER	179
DBPROP_ SAS_INIT_LOGICALNAME	180
DBPROP_ SAS_INIT_MACHINEDNSNAME	180
DBPROP_ SAS_INIT_PORT	180
DBPROP_ SAS_INIT_PROTOCOL	181
DBPROP_ SAS_INIT_SASEXE	181
DBPROP_ SAS_INIT_SASPARAMETERS	182
DBPROP_ SAS_INIT_SASWORKINGDIR	182
DBPROP_ SAS_INIT_SERVERPASSWORD	182
DBPROP_ SAS_INIT_SERVERRELEASE	183
DBPROP_ SAS_INIT_SERVERTYPE	183
DBPROP_ SAS_INIT_SERVICENAME	183
DBPROP_ SAS_INIT_WORKSPACE	184
DBPROP_ SAS_INIT_WORKSPACEID	184
DBPROP_ SAS_INIT_WORKSPACE_INIT_SCRIPT	185
DBPROP_ SAS_LIBOPTS	185
DBPROP_ SAS_MISSING_VALUES	185
DBPROP_ SAS_NLSOPTS	186
DBPROP_ SAS_OPTIMISTICLOCKING	186
DBPROP_ SAS_PAGESIZE	186
DBPROP_ SAS_PATH	187
DBPROP_ SAS_PHYSICALPOSITIONING	187

DBPROP_SAS_PT2DBPW	188
DBPROP_SAS_READPASSWORD	188
DBPROP_SAS_RESERVED_ROWSETFLAGS	189
DBPROP_SAS_SQLCONNECTIONSTRING	189
DBPROP_SAS_SQLENGINE	189
DBPROP_SAS_USE_TKMANAGER_SEARCHPATH	190
DBPROP_SAS_WHERE	190
DBPROP_SAS_WORKSPACE_INIT_LIST	191
DBPROP_SAS_WORKSPACE_INIT_LOG	191
DBPROP_SAS_WRITEPASSWORD	191
DBPROP_SESS_AUTOCOMMITISOLEVELS	192
DBPROP_SORTONINDEX	192
DBPROP_SUPPORTEDTXNISOLEVELS	192
DBPROP_UNIQUEROWS	193
DBPROP_UPDATABILITY	193
MDPROP_AGGREGATECELL_UPDATE	193
MDPROP_AXES	194
MDPROP_FLATTENING_SUPPORT	194
MDPROP_MDX_CASESUPPORT	195
MDPROP_MDX_DESCFLAGS	195
MDPROP_MDX_FORMULAS	195
MDPROP_MDX_JOINCUBES	196
MDPROP_MDX_MEMBER_FUNCTIONS	196
MDPROP_MDX_NUMERIC_FUNCTIONS	196
MDPROP_MDX_OBJQUALIFICATION	197
MDPROP_MDX_OUTERREFERENCE	197
MDPROP_MDX_QUERYBYPROPERTY	197
MDPROP_MDX_SET_FUNCTIONS	198
MDPROP_MDX_SLICER	198
MDPROP_MDX_STRING_COMPOP	199
MDPROP_NAMED_LEVELS	199
MDPROP_RANGEROWSET	199
OLE DB Properties: Sorted by ADO Name	200
OLE DB Properties: Sorted by Data Provider	203
Supported Properties in the IOM and OLAP Data Providers	203
Supported Properties in the Local Data Provider	206
Supported Properties in the SAS/SHARE and Base SAS Data Providers	208
OLE DB Properties: Sorted by Group	210
Supported Properties in the Data Source Group	210
Supported Properties in the Data Source Information Group	210
Supported Properties in the Initialization Group	211
Supported Properties in the Rowset Group	212
Supported Properties in the Session Group	213

OLE DB Properties: Introduction

What Are OLE DB Properties?

Properties are characteristics of an OLE DB object. Data sources, sessions, and rowsets all have associated properties. Every property has a description, a value, and a type, and

can be read-only or writable. Properties also have related methods that can be used to set object properties and retrieve object property information.

Properties are contained within sets that are contained within groups. A property set is a closely related collection of properties associated with one OLE DB object (for example, rowsets), or associated with one type of functionality (for example, initializing the data source component). A property group is a loosely knit group of property sets that are associated with a particular object or type of functionality. Every property belongs to a property set and, by virtue of its set, a property group.

Each property group name corresponds to an object name (like Rowset) or type of functionality (like Initialization). Property names and property set names can be used to determine whether a property is one of the OLE DB standard properties or a customized property specific to SAS.

- Properties specific to SAS begin with DBPROP_SAS_ (for example, DBPROP_SAS_FORMATS).
- Property sets specific to SAS begin with DBPROPSET_SAS_ (for example, DBPROPSET_SAS_DATASOURCEINFO).

You can also view properties sorted by group and set, provider, and ADO name.

Note: When you specify properties and cursor location on a recordset, you need to specify the properties after the cursor location is set. Certain properties can be specified only for the server. If the cursor location is specified after the properties, you might not see an error and you might have unexpected results. For more information, see the discussion of SAS customized properties in [“Known Issues for All Providers” on page 133](#).

Supported Property Groups

The following table lists the major property groups. The local, SAS/SHARE, IOM, and Base SAS providers support these groups; however, not all property sets within the groups are supported by all providers (see [“Supported Property Sets” on page 150](#)).

Table A3.1 Supported Property Groups

Property Group	Description
Data Source	Properties related to data sources.
Data Source Information	Properties that describe data sources. These properties are read-only and give unchanging information about the provider and data source.
Initialization	Properties for initializing the data source.
Rowset	Properties related to rowsets.
Session	Properties related to sessions.

Supported Property Sets

The following table lists the property sets (by their global unique identifiers (GUIDs)), associated groups, descriptions, and which providers support them.

Table A3.2 Supported Property Sets

Property Set GUID	Property Group and Description	Local	IOM	OLAP	SAS/SHA	Base SAS
DBPROPSET_DATASOURCE	Data Source Associated with data sources.	Yes	Yes	Yes	Yes	Yes
DBPROPSET_DATASOURCEINFO	Data Source Information Static information about data sources.	Yes	Yes	Yes	Yes	Yes
DBPROPSET_MDX_EXTENSION	Data Source Information Static information about features supported in MDX (Multidimensional Expressions). Calculated members are supported in an Excel pivot table.	no	Yes	Yes	no	no
DBPROPSET_SAS_DATASOURCEINFO	Data Source Information Static information about data sources that are specific to SAS providers.	Yes	no	no	no	no
DBPROPSET_DBINIT	Initialization Associated with initializing the data source.	Yes	Yes	Yes	Yes	Yes
DBPROPSET_SAS_DBINIT	Initialization Aspects of initializing the data source that are specific to SAS.	Yes	Yes	Yes	Yes	Yes
DBPROPSET_SAS_ROWSET	Rowset Aspects of rowsets that are specific to SAS.	Yes	Yes	Yes	Yes	Yes
DBPROPSET_ROWSET	Rowset Associated with rowsets.	Yes	Yes	Yes	Yes	Yes
DBPROPSET_SESSION	Session Associated with sessions.	Yes	Yes	Yes	Yes	Yes
DBPROPSET_SAS_SESSION	Session Aspects of sessions that are specific to SAS.	no	Yes	Yes	Yes	Yes

Property-Related Methods

OLE DB consumers use property-related methods to set properties on objects in order to force them to act in a certain way. For example, a consumer can set a property to enable an OLE DB rowset to support bookmarks. Consumers also use property-related methods to retrieve an OLE DB object's properties to find out what that object can do. For example, a consumer can get the value of a property that will reveal whether an OLE DB data source is read-only or writable.

The following table lists of OLE DB objects with their property-related methods. All five SAS providers for OLE DB support these methods. For more information about these methods, see the OLE DB documentation.

Table A3.3 OLE DB Object, Interface, Method, and Purpose

OLE DB Object	Interface	Method	Purpose
Data source	IDBProperties	GetPropertyInfo	Retrieves information about any property supported by a provider.
Data source	IDBProperties	GetProperties	For the specified data source object, retrieves the values currently set for properties that belong to these property groups: Data Source, Data Source Information, or Initialization.
Data source	IDBProperties	SetProperties	Sets Data Source or Initialization group properties for the specified data source object.
Session	IOpenRowset	OpenRowset	Sets properties on a rowset object created from all rows in a single base table.
Session	ISessionProperties	GetProperties	For the specified session object, retrieves the values currently set for properties that belong to the Session group.
Session	ISessionProperties	SetProperties	Sets Session group properties for the specified session object.
Session	IDBSchemaRowset	GetRowset	Retrieves information about a specified schema rowset and sets Rowset group properties for that schema rowset.
Session	ITableDefinition	CreateTable	Creates a new base table in the data source, and sets specific properties for the table's Column property group.
Command	ICommandProperties	GetProperties	For the specified rowset object, retrieves the Rowset group properties that have been set by the SetProperties method.
Command	ICommandProperties	SetProperties	Sets the Rowset group properties that must be supported by the rowset or rowsets that are returned after the command is executed.

OLE DB Object	Interface	Method	Purpose
Rowset	IRowsetInfo	GetProperties	Retrieves values for all properties that the specified rowset supports.

OLE DB Properties: Descriptions

DBPROP_APPENDONLY

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BOOL

Description (ADO property name): Append-Only Rowset

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_AUTH_PASSWORD

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Initialization

Property Set: DBPROPSET_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): Password

Notes:

Use this property to identify the password to use when connecting to a server. Use this property in conjunction with the DBPROP_AUTH_USERID property.

Local Provider: This property is ignored by the local provider.

IOM Provider: When connecting to a remote SAS Workspace Server, set this property to the user's log-in password. However, you do not need to set this property in these two circumstances:

- You are connecting to an existing workspace via the DBPROP_SAS_INIT_WORKSPACE property.
- You are instantiating a new local server by setting DBPROP_INIT_DATASOURCE to _LOCAL_.

SAS/SHARE Provider: Set this property to the user's log-in password when you connect to a SAS/SHARE server that requires user authentication. Consult your SAS/SHARE server administrator if you are not sure about your server's authentication requirements.

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_AUTH_USERID

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Initialization

Property Set: DBPROPSET_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): User ID

Notes:

When connecting to a server, use this property to identify the user's log-in ID. Use this property in conjunction with the DBPROP_AUTH_PASSWORD property.

Local Provider: This property is ignored by the local provider.

IOM Provider: When connecting to a remote SAS Workspace Server, set this property to the user's log-in ID. However, you do not need to set this property in these two circumstances:

- You are connecting to an existing workspace via the DBPROP_SAS_INIT_WORKSPACE property.
- You are instantiating a new local server by setting DBPROP_INIT_DATASOURCE to _LOCAL_.

SAS/SHARE Provider: Set this property to the user's log-in ID when connecting to a SAS/SHARE server that requires user authentication. Consult your SAS/SHARE server administrator if you are not sure about your server's authentication requirements.

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_BOOKMARKS

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BOOL

Description (ADO property name): Use Bookmarks

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_BOOKMARKSKIPPED

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: No

Column: False

Type: VT_BOOL

Description (ADO property name): Skip Deleted Bookmarks

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_BOOKMARKTYPE

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_I4

Description (ADO property name): Bookmark Type

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_BYREFACCESSORS

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_BOOL

Description (ADO property name): Pass By Ref Accessors

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_CANFETCHBACKWARDS

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BOOL

Description (ADO property name): Fetch Backwards

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_CANHOLDROWS

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BOOL

Description (ADO property name): Hold Rows

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_CANSCROLLBACKWARDS

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BOOL

Description (ADO property name): Scroll Backwards

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_CLIENTCURSOR

Supported in: SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BOOL

Description (ADO property name): ClientCursor

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.
This property supports client-side cursors in ADO applications.

DBPROP_CURRENTCATALOG

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source

Property Set: DBPROPSET_DATASOURCE

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): Current Catalog

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_DATASOURCEONLY

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_BOOL

Description (ADO property name): Read-Only Data Source

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_DATASOURCE_TYPE

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: Yes

Column: False

Type: VT_I4

Description (ADO property name): Data Source Type

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_DBMSNAME

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_BSTR

Description (ADO property name): DBMS Name

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_DBMSVER

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_BSTR

Description (ADO property name): DBMS Version

Notes:

Local Provider: Identifies the release version of the provider. Returns the same value as DBPROP_PROVIDERVER.

IOM and SAS/SHARE Providers: Identifies the release version of the connected server.

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_IAccessor

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: No

Column: False

Type: VT_BOOL

Description (ADO property name): IAccessor

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_IColumnsInfo

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: No

Column: False

Type: VT_BOOL

Description (ADO property name): IColumnsInfo

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_IConvertType

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: No

Column: False

Type: VT_BOOL

Description (ADO property name): IConvertType

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_INIT_ASYNC

Supported in: IOM Provider, OLAP Provider

Property Group: Initialization

Property Set: DBPROPSET_DBINIT

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): Asynchronous Processing

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_INIT_DATASOURCE

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Initialization

Property Set: DBPROPSET_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): Data Source

Notes:

Use this property to identify the server or physical disk location that you want to connect to.

Local Provider: Set this property to the directory location that contains your data. The directory specification can be a local drive, a mounted drive, or the fully qualified path to a network drive. You can also set this property to "_LOCAL_" if you want to specify a fully qualified file path when opening a rowset.

IOM Provider: Set this property to "_LOCAL_" when you want a SAS Workspace Server connection. When creating a remote SAS Workspace Server connection, you can set this property to any string that you want to associate with the connection.

SAS/SHARE Provider: Set this property to the name of the SAS/SHARE server to which you want to connect. Consult your SAS/SHARE server administrator if you do not know the server name. If you are using DBPROP_SAS_INIT_LOCALSERVER to start the server, then you must use the same Server ID passed in via the DBPROP_SAS_INIT_SASPARAMETERS property.

OLAP Provider: Set this property to the network DNS name of the OLAP server or the IP address of the OLAP server. When connecting to a remote OLAP server, use this property in conjunction with DBPROP_SAS_INIT_SERVICENAME, DBPROP_SAS_INIT_PROTOCOL, and DBPROP_SAS_INIT_PORT.

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_INIT_HWND

Supported in: IOM Provider, OLAP Provider, Local Provider

Property Group: Initialization

Property Set: DBPROPSET_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_I4

Description (ADO property name): Window Handle

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_INIT_LCID

Supported in: IOM Provider, OLAP Provider

Property Group: Initialization

Property Set: DBPROPSET_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_I4

Description (ADO property name): Locale Identifier

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_INIT_LOCATION

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Initialization

Property Set: DBPROPSET_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): Location

Notes:

Local Provider: This property is ignored by the Local Provider.

IOM Provider: This property is ignored by the IOM Provider.

SAS/SHARE Provider: When connecting to a server that is running on a different node than the client, set this property to the DNS name of the node that is hosting the server.

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_INIT_MODE

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Initialization

Property Set: DBPROPSET_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_I4

Description (ADO property name): Mode

Notes:

Use this property to specify access permissions to the connected data source.

All five data providers honor the following bitmasks:

- DB_MODE_READ
- DB_MODE_READWRITE
- DB_MODE_SHARE_DENY_NONE

Other bitmasks are ignored and the mode is degraded to (DB_MODE_READ | DB_MODE_SHARE_DENY_NONE).

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_INIT_PROMPT

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Initialization

Property Set: DBPROPSET_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_I2

Description (ADO property name): Prompt

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_INIT_PROVIDERSTRING

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Initialization

Property Set: DBPROPSET_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): Extended Properties

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_IRowset

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: No

Column: False

Type: VT_BOOL

Description (ADO property name): IRowset

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_IRowsetChange

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BOOL

Description (ADO property name): IRowsetChange

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_IRowsetIdentity

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: No

Column: False

Type: VT_BOOL

Description (ADO property name): IRowsetIdentity

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_IRowsetInfo

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: No

Column: False

Type: VT_BOOL

Description (ADO property name): IRowsetInfo

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_IRowsetLocate

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BOOL

Description (ADO property name): IRowsetLocate

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_IRowsetUpdate

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BOOL

Description (ADO property name): IRowsetUpdate

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_IRowsetView

Supported in: Local Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BOOL

Description (ADO property name): IRowsetView

Notes:

Experimental.

DBPROP_ISupportErrorInfo

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: No

Column: False

Type: VT_BOOL

Description (ADO property name): ISupportErrorInfo

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_IViewFilter

Supported in: Local Provider

Property Group: Rowset

Property Set: DBPROPSET_VIEW

Read: Yes

Write: Yes

Column: False

Type: VT_BOOL

Description (ADO property name): IViewFilter

Notes:

Experimental.

DBPROP_IViewRowset

Supported in: Local Provider

Property Group: Rowset

Property Set: DBPROPSET_VIEW

Read: Yes

Write: Yes

Column: False

Type: VT_BOOL

Description (ADO property name): IViewRowset

Notes:

Experimental.

DBPROP_IViewSort

Supported in: Local Provider

Property Group: Rowset

Property Set: DBPROPSET_VIEW

Read: Yes

Write: Yes

Column: False

Type: VT_BOOL

Description (ADO property name): IViewSort

Notes:

Experimental.

DBPROP_LITERALBOOKMARKS

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: No

Column: False

Type: VT_BOOL

Description (ADO property name): Literal Bookmarks

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_LITERALIDENTITY

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: No

Column: False

Type: VT_BOOL

Description (ADO property name): Literal Row Identity

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_LOCKMODE

Supported in: IOM Provider, OLAP Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_I4

Description (ADO property name): Lock Mode

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_MAXOPENROWS

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_I4

Description (ADO property name): Maximum Open Rows

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_MAXORSINFILTER

Supported in: Local Provider

Property Group: Rowset

Property Set: DBPROPSET_VIEW

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): Maximum OR Conditions

Notes:

Experimental.

DBPROP_MAXPENDINGROWS

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): Maximum Pending Rows

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_MAXROWS

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): Maximum Rows

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_MAXSORTCOLUMNS

Supported in: Local Provider

Property Group: Rowset

Property Set: DBPROPSET_VIEW

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): Maximum Sort Columns

Notes:

Experimental.

DBPROP_MSMD_MDX_CALCMEMB_EXTENSIONS

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source Information

Property Set: DBPROPSET_MDX_EXTENSIONS

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): MDX Calculated Members Extensions

DBPROP_ORDEREDBOOKMARKS

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BOOL

Description (ADO property name): Bookmarks Ordered

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_OTHERINSERT

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: No

Column: False

Type: VT_BOOL

Description (ADO property name): Others' Inserts Visible

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_OTHERUPDATEDELETE

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: No

Column: False

Type: VT_BOOL

Description (ADO property name): Others' Changes Visible

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_OWNINGINSERT

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: No

Column: False

Type: VT_BOOL

Description (ADO property name): Own Inserts Visible

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_OWNINGUPDATEDELETE

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: No

Column: False

Type: VT_BOOL

Description (ADO property name): Own Changes Visible

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_PROVIDERVER

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_BSTR

Description (ADO property name): Provider Version

Notes:

Use this property to identify the release version of the provider. The major release, minor release, revision number, and build number are encoded in the returned value.

Local Provider: Returns the same value as DBPROP_DBMSVER.

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_REMOVEDELETED

Supported in: Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_ROWSET

Read: Yes

Write: No

Column: False

Type: VT_BOOL

Description (ADO property name): Remove Deleted Rows

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_SAS_ALTERPASSWORD

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS Alter Password

Notes:

Use this property to access read-protected, write-protected, or alter-protected SAS files.

For more information about the SAS data set password facility, see "File Protection" in *SAS Language Reference: Concepts*.

DBPROP_SAS_BLANKPADDING

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BOOL

Description (ADO property name): SAS Preserve Trailing Blanks

Notes:

If the value of DBPROP_SAS_BLANKPADDING is VARIANT_TRUE when character data is read, then trailing blanks are preserved. Otherwise, trailing blanks are trimmed.

DBPROP_SAS_DATASET_ENCODING

Supported in: Local Provider

Property Group: Rowset

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: No

Column: False

Type: VT_UI4

Description (ADO property name): "SAS Data Set Encoding"

Notes:

Use this property to retrieve the data set encoding value from SAS data sets.

DBPROP_SAS_DATASET_LABEL

Supported in: Local Provider

Property Group: Rowset

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): "SAS Data Set Label"

Notes:

Use this property to set and retrieve the data set label value from SAS data sets.

DBPROP_SAS_DATASET_TYPE

Supported in: Local Provider

Property Group: Rowset

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): "SAS Data Set Type"

Notes:

Use this property to set and retrieve the data set type value from SAS data sets.

DBPROP_SAS_DATASET_WINDOWS_CODEPAGE

Supported in: Local Provider, SAS/SHARE Provider

Property Group: Rowset

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: No

Column: False

Type: VT_UI4

Description (ADO property name): "SAS Data Set Windows Code Page"

Notes:

Use this property to retrieve the Windows Code Page ID that is determined from the encoding value of the SAS data set. The default value of this property is 1252, which is the Western Latin 1 encoding.

DBPROP_SAS_DATASETOPTS

Supported in: IOM Provider

Property Group: Rowset

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): "SAS Data Set Options"

Notes:

Use this property to specify SAS data set options when directly opening a table. For more information about SAS data set options, see "SAS Data Set Options" in *SAS Language Reference: Dictionary*.

DBPROP_SAS_DEFAULTFILEFORMAT

Supported in: Local Provider

Property Group: Data Source Information

Property Set: DBPROPSET_SAS_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_BSTR

Description (ADO property name): SAS Default File Format

Notes:

Use this property to define the default value for DBPROP_SAS_INITFILEFORMAT. The value can change from one release of the provider to the next. In general, the default file format that is identified by this property corresponds to the data set format of the latest major SAS release.

DBPROP_SAS_ENGINE

Supported in: IOM Provider, OLAP Provider

Property Group: Initialization

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS Library Engine

Notes:

Use this property to specify the name of the engine to use when this rowset is opened.

The available engines depend on the host system, the version of SAS, and the site installation. This property is comparable to the engine name option in the SAS LIBNAME statement.

When this property is set, DBPROP_SAS_PATH must also be set. If DBPROP_SAS_ENGINE is not set and DBPROP_SAS_PATH is set, then the default engine is used.

DBPROP_SAS_FMTERR

Supported in: Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BOOL

Description (ADO property name): SAS Format Error

Note:

Use this property to control how the provider handles unknown format and informat names. When the property value is VARIANT_TRUE, the provider generates an error during row I/O operations if the associated format or informat is not found. When the property value is VARIANT_FALSE and the associated format or informat is not found, then the provider uses BESTw.d for numeric columns and \$CHARw. for character columns.

DBPROP_SAS_FORMATS

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS Formats

Notes:

Use this property to format columns of data. You can use one of the following string values:

- "_ALL_" , which indicates that all variables should be returned as formatted values.
- "+_ALL_" , which indicates that all variables should be returned as both formatted and unformatted values.
- a comma-separated list with items of the following form:

```
[+] COLUMN [=FORMAT [w] . [d]]
```

- The optional plus (+) modifier indicates that the variable should be returned as both formatted and unformatted values.

- **COLUMN** is the name of the variable that you want to format.
- **FORMATw.d** is the SAS format that you want to apply. If this value is not given, then the default format is used.

Use this property only for ADO consumers. OLE DB consumers should not use this property to format data.

DBPROP_SAS_GET_MISSING_VALUES

Supported in: Local Provider

Property Group: Rowset

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BOOL

Description (ADO property name): SAS Get Missing Values Grid

Notes:

Use this property to control whether a grid that represents special numeric missing values data is returned or standard data is returned to the user. This property can have the following values:

- True, to indicate that the SAS missing values grid is requested. In order to identify the special numeric missing values, the returned data set is the opposite of the standard data values. Numeric fields that have a value such as 15 are instead returned as Null. Fields that have a missing value return a number that identifies the SAS numeric missing value indicator. All character fields are returned with a length of zero.
- False, to indicate that standard data set values are requested. False is the default setting.

For more information about this property, see [“Reading Special Numeric Missing Values from a Data Set”](#) on page 105.

DBPROP_SAS_INFORMATS

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS Informat

Notes:

Use this property to apply informats to columns of data. You can use one of the following string values:

- "_ALL_" , which indicates that all variables should be output using informatted values.
- a comma-separated list with items of the form:

```
COLUMN [=INFORMAT[w] . [d] ]
```

- **COLUMN** is the name of the variable that you write with an informat.
- **INFORMATw.d** is the SAS informat that you want to apply. If this informat is not given, then the default informat is used.

Use this property only for ADO consumers. OLE DB consumers should not use this property to informat data.

DBPROP_SAS_INIT_CELLCACHE SIZE

Supported in: IOM Provider, OLAP Provider

Property Group: Initialization

Property Set: DBPROPSET_SAS_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_I4

Description (ADO property name): SAS Cell Cache Size

Notes:

This property defines the maximum number of cells that will be cached by the OLAP Provider at any one time. Cached cells are used only to handle requests for single cells (that is, calls to `IMDDataset::GetCellData` where `ulStartCell` and `ulEndCell` are equal). The actual number of cells cached in each `MDDataset` is based on the number of tuples along the column's axis. Usually, the number of tuples is equivalent to the number of cells in one row. The following formula can be used to determine the number of cached cells:

$$\text{floor}(\text{SAS Cell Cache Size} / \text{Number of cells per row}) * \text{Number of cells per row}$$

For example, assume that you have an `MDDataset` with five cells in a row. A SAS Cell Cache Size of 12 results in $\text{floor}(12 / 5) * 5 = \text{floor}(2.4) * 5 = 2 * 5 = 10$ cells actually cached. For this `MDDataset`, all values for SAS Cell Cache Size between 10 and 14 (inclusive) will yield 10 cached cells.

This property can have the following values:

- -1 indicates that the entire `MDDataset` will be cached in memory.
- 0 indicates that caching will be disabled.
- >0 indicates the maximum number of cells that will be cached.

DBPROP_SAS_INIT_FILEFORMAT

Supported in: Local Provider

Property Group: Initialization

Property Set: DBPROPSET_SAS_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS File Format

Notes:

Use this property to define the SAS file format associated with the data source. The Local Provider supports the following file formats:

- Version 6 data sets
- Versions 7, 8, and 9 data sets
- Version 5 transport files

This property accepts a string value that indicates which file format to associate with the directory that is given the DBPROP_INIT_DATASOURCE value. Valid strings for DBPROP_SAS_INIT_FILEFORMAT are as follows:

- "V9" for Version 9, Version 8, or Version 7 data sets
- "V8" for Version 8 or Version 7 data sets
- "V7" for Version 7 or Version 8 data sets
- "V6" for Version 6 data sets
- "XPT" for Version 5 transport files

"V7" and "V8" specify the equivalent file formats and can be used interchangeably.

If no value is set for this property, then the provider examines the data source directory to determine which file format to use. If a decision cannot be made based on the contents of the data source directory, then the provider uses the DBPROP_SAS_DEFAULTFILEFORMAT value as the default for DBPROP_SAS_INIT_FILEFORMAT.

Note: When you specify a value of "V8", the Local Provider does not properly convert numeric variables for SAS data sets created on platforms other than Windows. In this case, the Local Provider returns incorrect data for the numeric variable (although it does not return an error). To ensure that numeric variables are converted properly for SAS data sets on platforms other than Windows, use the default file format or "V9".

DBPROP_SAS_INIT_LOCALSERVER

Supported in: SAS/SHARE Provider, Base SAS Provider

Property Group: Initialization

Property Set: DBPROPSET_SAS_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_I4

Description (ADO property name): SAS Local Server

Note:

Use this property to determine whether the SAS/SHARE Provider will start a local server. The value for this property can be 0 or 1. Zero is the default value and indicates that the provider should not start a server. One indicates that the SAS/SHARE provider should start a local server.

DBPROP_SAS_INIT_LOGICALNAME

Supported in: IOM Provider, OLAP Provider

Property Group: Initialization

Property Set: DBPROPSET_SAS_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS Logical Name

Note:

This property is obsolete and is provided only for backward compatibility.

DBPROP_SAS_INIT_MACHINEDNSNAME

Supported in: IOM Provider, OLAP Provider

Property Group: Initialization

Property Set: DBPROPSET_SAS_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS Machine DNS Name

Notes:

Use this property to specify the network DNS name of the server or the IP address of the server.

When connecting to a remote SAS Workspace Server, use this property in conjunction with DBPROP_SAS_INIT_SERVICENAME, DBPROP_SAS_INIT_PROTOCOL, and DBPROP_SAS_INIT_PORT.

DBPROP_SAS_INIT_PORT

Supported in: IOM Provider, OLAP Provider

Property Group: Initialization

Property Set: DBPROPSET_SAS_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_I4

Description (ADO property name): SAS Port

Notes:

Use this property to identify the TCP/IP port number to use on the machine that is hosting the SAS Workspace Server.

When connecting to a remote SAS Workspace Server, use this property in conjunction with DBPROP_SAS_INIT_MACHINEDNSNAME and DBPROP_SAS_INIT_PROTOCOL. This property is functionally equivalent to DBPROP_SAS_INIT_SERVICENAME.

DBPROP_SAS_INIT_PROTOCOL

Supported in: IOM Provider, OLAP Provider

Property Group: Initialization

Property Set: DBPROPSET_SAS_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_I4

Description (ADO property name): SAS Protocol

Notes:

Use this property to indicate how to communicate with the SAS server. Its value is either ProtocolCom, ProtocolCorba, or ProtocolBridge.

When connecting to a remote SAS Workspace Server, use this property in conjunction with DBPROP_SAS_INIT_SERVICENAME, DBPROP_SAS_INIT_MACHINEDNSNAME, and DBPROP_SAS_INIT_PORT.

DBPROP_SAS_INIT_SASEXE

Supported in: SAS/SHARE Provider, Base SAS Provider

Property Group: Initialization

Property Set: DBPROPSET_SAS_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS Executable

Notes:

Use this property to specify the fully qualified path for the SAS executable file (sas.exe) that you use to start a SAS session. The default value might change from one release of the provider to the next. In general, the default path corresponds to the standard installation path of the latest major SAS release.

DBPROP_SAS_INIT_SASPARAMETERS

Supported in: SAS/SHARE Provider, Base SAS Provider

Property Group: Initialization

Property Set: DBPROPSET_SAS_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS Parameters

Note:

Use this property to specify the parameters that are used to invoke SAS. The parameters must include `-initstmt`, which executes a SAS macro to start the local server. An example is `-initstmt %sasodbc(sdplserv) -icon -nologo -notutorialdlg`.

DBPROP_SAS_INIT_SASWORKINGDIR

Supported in: SAS/SHARE Provider, Base SAS Provider

Property Group: Initialization

Property Set: DBPROPSET_SAS_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS Working Directory

Note:

Use this property to specify the fully qualified path for the directory that you want to use as the SAS working directory.

DBPROP_SAS_INIT_SERVERPASSWORD

Supported in: SAS/SHARE Provider, Base SAS Provider

Property Group: Initialization

Property Set: DBPROPSET_SAS_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS Server Access Password

Note:

Use this property to specify the server access password when one is needed. Consult your SAS/SHARE server administrator if you are unsure about your server's authentication requirements.

DBPROP_SAS_INIT_SERVERRELEASE

Supported in: SAS/SHARE Provider, Base SAS Provider

Property Group: Initialization

Property Set: DBPROPSET_SAS_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_I4

Description (ADO property name): SAS Server Release

Notes:

Use this property to specify the SAS release of the server that you intend to use. The value of this property can be 7, 8 or 9. The value should correspond to the major release number of the connected server. Consult your SAS/SHARE server administrator if you are unsure of your server's release number.

DBPROP_SAS_INIT_SERVERTYPE

Supported in: IOM Provider, OLAP Provider

Property Group: Initialization

Property Set: DBPROPSET_SAS_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_I4

Description (ADO property name): SAS Server Type

Notes:

Use this property to specify the type of SAS server that you want to connect to. This value can be DBPROPVAL_DST_TDP for a tabular data server or DBPROPVAL_DST_MDP for an OLAP server. Contact your system administrator if you are unsure about the server type.

DBPROP_SAS_INIT_SERVICENAME

Supported in: IOM Provider, OLAP Provider

Property Group: Initialization

Property Set: DBPROPSET_SAS_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS Service Name

Notes:

Use this property to identify the TCP/IP service name to use on the SAS Workspace Server.

This name is used to look up the port on the machine that is hosting the SAS Workspace Server. When connecting to a remote SAS Workspace Server, use this property in conjunction with DBPROP_SAS_INIT_MACHINEDNSNAME and DBPROP_SAS_INIT_PROTOCOL. This value is functionally equivalent to DBPROP_SAS_INIT_PORT.

DBPROP_SAS_INIT_WORKSPACE

Supported in: IOM Provider, OLAP Provider

Property Group: Initialization

Property Set: DBPROPSET_SAS_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_UNKNOWN

Description (ADO property name): SAS Workspace Interface

Notes:

This property is no longer supported. Use DBPROP_SAS_INIT_WORKSPACEID instead.

DBPROP_SAS_INIT_WORKSPACEID

Supported in: IOM Provider, OLAP Provider

Property Group: Initialization

Property Set: DBPROPSET_SAS_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS Workspace ID

Notes:

Use this property to specify the UniqueIdentifier for a SAS workspace.

When connecting to an existing SAS workspace, set this property equal to the UniqueIdentifier of the workspace. UniqueIdentifier is a property on the SAS IOM workspace object.

DBPROP_SAS_INIT_WORKSPACE_INIT_SCRIPT

Supported in: IOM Provider, OLAP Provider

Property Group: Initialization

Property Set: DBPROPSET_SAS_DBINIT

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS Workspace Init Script

Notes:

You can use this BSTR Data Source Initialization property to hold SAS code that will be submitted for execution to a SAS Workspace server immediately after a connection is to the server is established. The code can be any valid SAS code.

DBPROP_SAS_LIBOPTS

Supported in: IOM Provider, OLAP Provider

Property Group: Initialization

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS Library Options

Notes:

Use this property to specify any LIBNAME or engine options that should be used when opening this rowset.

When this property is set, DBPROP_SAS_PATH must also be set. If DBPROP_SAS_PATH is set and the libref is not specified as part of the rowset's TableID, then the IOM Provider assigns a libref on the SAS Workspace Server. The options specified by this property are applied to the libref assignment. The libref is unassigned when the rowset is released.

DBPROP_SAS_MISSING_VALUES

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BOOL

Description (ADO property name): SAS Missing Values

DBPROP_SAS_NLSOPTS

Supported in: Local Provider

Property Group: Rowset

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_I4

Description (ADO property name): SAS NLS Options

Notes:

Use this property to specify how strings will be handled by the provider.

DBPROP_SAS_OPTIMISTICLOCKING

Supported in: IOM Provider, OLAP Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BOOL

Description (ADO property name): SAS Optimistic Locking

Notes:

Use this property to control whether a data set is opened with SAS member-level access or SAS record-level access. A value of VARIANT_TRUE indicates record-level access; a value of VARIANT_FALSE indicates member-level access.

When a file is opened for record-level access, the server uses a record-locking strategy, which means that exclusive update privileges are acquired on a per record basis. This setting enables multiple users to access the same file simultaneously. When a file is opened with member-level control, the server grants exclusive access to the entire file (for example, data set), which prevents other users from updating the file.

DBPROP_SAS_PAGESIZE

Supported in: IOM Provider, OLAP Provider

Property Group: Rowset

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_I4

Description (ADO property name): SAS Page Size

Notes:

Use this property to set the number of rows in a single page of data retrieved from the server. A value of 0 disables paging. If the DBPROP_SAS_PAGESIZE property is set to -1, then the provider attempts to cache the entire data set when it reads the data set for the first time. Your application must ensure that there is enough RAM for this operation to succeed.

DBPROP_SAS_PATH

Supported in: IOM Provider, OLAP Provider

Property Group: Initialization

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS Physical Path

Notes:

Use this property to specify the host file system path for the location of the table referenced by this rowset (such as a directory).

The path form depends on the platform that is hosting the server and the conventions of the selected engine. The engine usually requires you to specify an existing location on the server machine.

When this property is set and the libref is not specified as part of the rowset's TableID, the provider assigns a libref on behalf of the client. The path of the libref is specified by this property. The optional properties DBPROP_SAS_ENGINE and DBPROP_SAS_LIBOPTS are also used to assign the new libref. The libref is unassigned when the rowset is released.

DBPROP_SAS_PHYSICALPOSITIONING

Supported in: IOM Provider, OLAP Provider

Property Group: Rowset

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BOOL

Description (ADO property name): SAS Physical Positioning

Notes:

This property determines which positioning model is used by the IOM provider. When this property is FALSE, then standard OLE DB logical positioning is used. When this property is TRUE, the SAS physical positioning model is used. By default, this property is FALSE.

When offsets are being calculated, positioning is handled in these ways:

- Logical positioning does not count deleted records.
- Physical positioning does count deleted records.

DBPROP_SAS_PT2DBPW

Supported in: SAS/SHARE Provider, Base SAS Provider

Property Group: Session

Property Set: DBPROPSET_SAS_SESSION

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS PT2DBPW

Notes:

Use this property to enable the processing of SQL queries on remote databases. Set it to the same value that is specified in the PT2DBPW option in the PROC SERVER statement.

This property must be set whenever the PT2DBPW option is specified in the PROC SERVER statement and you are using a DBPROP_SAS_SQLENGINE value other than the default of SQLVIEW.

DBPROP_SAS_READPASSWORD

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS Read Password

Notes:

Use this property to specify the read password when you access a read-protected SAS file.

For more information about the SAS data set password facility, see "File Protection" in *SAS Language Reference: Concepts*.

DBPROP_SAS_RESERVED_ROWSETFLAGS

Supported in: IOM Provider, OLAP Provider

Property Group: Rowset

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_I4

Description (ADO property name): SAS Reserved 1

Notes:

Reserved for future use.

DBPROP_SAS_SQLCONNECTIONSTRING

Supported in: IOM Provider, OLAP Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Session

Property Set: DBPROPSET_SAS_SESSION

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS SQL Engine Options

Notes:

When you are creating a Command object, use this property to specify SAS options to associate with the DBMS engine. Use this property when DBPROP_SAS_SQLENGINE is set to a value other than its default SQLVIEW value.

DBPROP_SAS_SQLENGINE

Supported in: IOM Provider, OLAP Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Session

Property Set: DBPROPSET_SAS_SESSION

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS SQL Engine

Notes:

When you are creating a Command object, use this property to identify which DBMS engine to use.

DBPROP_SAS_USE_TKMANAGER_SEARCHPATH

Supported in: Local Provider

Property Group: Data Source Information

Property Set: DBPROPSET_SAS_DATASOURCEINFO

Read: Yes

Write: Yes

Column: False

Type: VT_BOOL

Description (ADO property name): Use TK Manager Search Path

Notes:

This property indicates that the Threaded Kernel Path has been set in the TKMANAGER_SEARCH_PATH environment variable and that the Threaded Kernel Manager is to be used. TKMANAGER_SEARCH_PATH must contain the fully qualified path to the Threaded Kernel Manager DLL (tkmanager.dll) as the first path, followed by the fully qualified paths for all of the Threaded Kernel files required by the provider. This property is FALSE by default.

DBPROP_SAS_WHERE

Supported in: SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS Where

Notes:

Use this property to set a SAS WHERE clause on the rowset. When a WHERE clause is used, the rowset returns only rows that meet the condition that is specified by the WHERE clause. The WHERE clause is limited to forward-only access. It cannot be used with a client-side cursor.

This property is the most efficient way to subset a data set because the filtering process is done on the server. Only rows that match the WHERE-clause criteria are transmitted from the server to the client. Other rowset filtering methods are done in the client

process, which means that rows that do not meet the criteria are transmitted from the server to the client.

For more information about WHERE-clause processing, see *SAS Language Reference: Concepts* and the *Base SAS Procedures Guide*.

DBPROP_SAS_WORKSPACE_INIT_LIST

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source Information

Property Set: DBPROPSET_SAS_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_BSTR

Description (ADO property name): SAS Workspace Init List

Notes:

You can use this read-only BSTR Data Source property to capture the results of the SAS listing for SAS code that was submitted by using the DBPROP_SAS_INIT_WORKSPACE_INIT_SCRIPT (the "SAS Workspace Init Script") property.

DBPROP_SAS_WORKSPACE_INIT_LOG

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source Information

Property Set: DBPROPSET_SAS_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_BSTR

Description (ADO property name): SAS Workspace Init Log

Notes:

You can use this read-only BSTR Data Source property to capture the SAS log for SAS code that was submitted by using the DBPROP_SAS_INIT_WORKSPACE_INIT_SCRIPT (the "SAS Workspace Init Script") property.

DBPROP_SAS_WRITEPASSWORD

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Rowset

Property Set: DBPROPSET_SAS_ROWSET

Read: Yes

Write: Yes

Column: False

Type: VT_BSTR

Description (ADO property name): SAS Write Password

Notes:

Use this property to specify the write password when accessing a write-protected SAS file.

For more information about the SAS data set password facility, see "File Protection" in *SAS Language Reference: Concepts*.

DBPROP_SESS_AUTOCOMMITISOLEVELS

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Session

Property Set: DBPROPSET_SESSION

Read: Yes

Write: Yes

Column: False

Type: VT_I4

Description (ADO property name): Autocommit Isolation Levels

Notes:

For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_SORTONINDEX

Supported in: Local Provider

Property Group: Rowset

Property Set: DBPROPSET_VIEW

Read: Yes

Write: No

Column: False

Type: VT_BOOL

Description (ADO property name): Sort on Index

Notes:

Experimental.

DBPROP_SUPPORTEDTXNISOLEVELS

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider

Property Group: Data Source Information
Property Set: DBPROPSET_DATASOURCEINFO
Read: Yes
Write: No
Column: False
Type: VT_I4
Description (ADO property name): Isolation Levels
Notes:
For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

DBPROP_UNIQUEROWS

Supported in: SAS/SHARE Provider, Base SAS Provider
Property Group: Rowset
Property Set: DBPROPSET_ROWSET
Read: Yes
Write: Yes
Column: False
Type: VT_BOOL
Description (ADO property name): UniqueRows
Notes:
For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.
This property was added to support client-side cursors in ADO applications.

DBPROP_UPDATABILITY

Supported in: IOM Provider, OLAP Provider, Local Provider, SAS/SHARE Provider, Base SAS Provider
Property Group: Rowset
Property Set: DBPROPSET_ROWSET
Read: Yes
Write: Yes
Column: False
Type: VT_I4
Description (ADO property name): Updatability
Notes:
For more information, see the *OLE DB Programmer's Reference and Data Access SDK*.

MDPROP_AGGREGATECELL_UPDATE

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): Support for updating aggregated cells

Notes:

For more information, see the "OLE DB for Online Analytical Processing (OLAP)" section of the *OLE DB Programmer's Reference and Data Access SDK*. Also see the *SAS OLAP Server: MDX Guide*.

MDPROP_AXES

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): Number of axes in the data set

Notes:

For more information, see the "OLE DB for Online Analytical Processing (OLAP)" section of the *OLE DB Programmer's Reference and Data Access SDK*. Also see the *SAS OLAP Server: MDX Guide*.

MDPROP_FLATTENING_SUPPORT

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): Flattening Support

Notes:

For more information, see the "OLE DB for Online Analytical Processing (OLAP)" section of the *OLE DB Programmer's Reference and Data Access SDK*. Also see the *SAS OLAP Server: MDX Guide*.

MDPROP_MDX_CASESUPPORT

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): Support for MDX case statements

Notes:

For more information, see the "OLE DB for Online Analytical Processing (OLAP)" section of the *OLE DB Programmer's Reference and Data Access SDK*. Also see the *SAS OLAP Server: MDX Guide*.

MDPROP_MDX_DESCFLAGS

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): Support for various <desc flag> values in the DESCENDANTS function

Notes:

For more information, see the "OLE DB for Online Analytical Processing (OLAP)" section of the *OLE DB Programmer's Reference and Data Access SDK*. Also see the *SAS OLAP Server: MDX Guide*.

MDPROP_MDX_FORMULAS

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): Support for creation of named sets and calculated members

Notes:

For more information, see the "OLE DB for Online Analytical Processing (OLAP)" section of the *OLE DB Programmer's Reference and Data Access SDK*. Also see the *SAS OLAP Server: MDX Guide*.

MDPROP_MDX_JOINCUBES

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): Support for query joining multiple cubes

Notes:

For more information, see the "OLE DB for Online Analytical Processing (OLAP)" section of the *OLE DB Programmer's Reference and Data Access SDK*. Also see the *SAS OLAP Server: MDX Guide*.

MDPROP_MDX_MEMBER_FUNCTIONS

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): Support for various member functions

Notes:

For more information, see the "OLE DB for Online Analytical Processing (OLAP)" section of the *OLE DB Programmer's Reference and Data Access SDK*. Also see the *SAS OLAP Server: MDX Guide*.

MDPROP_MDX_NUMERIC_FUNCTIONS

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): Support for various numeric functions

Notes:

For more information, see the "OLE DB for Online Analytical Processing (OLAP)" section of the *OLE DB Programmer's Reference and Data Access SDK*. Also see the *SAS OLAP Server: MDX Guide*.

MDPROP_MDX_OBJQUALIFICATION

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): Provider's ability to qualify a cube name

Notes:

For more information, see the "OLE DB for Online Analytical Processing (OLAP)" section of the *OLE DB Programmer's Reference and Data Access SDK*. Also see the *SAS OLAP Server: MDX Guide*.

MDPROP_MDX_OUTERREFERENCE

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): Support for outer Reference in an MDX statement

Notes:

For more information, see the "OLE DB for Online Analytical Processing (OLAP)" section of the *OLE DB Programmer's Reference and Data Access SDK*. Also see the *SAS OLAP Server: MDX Guide*.

MDPROP_MDX_QUERYBYPROPERTY

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_BOOL

Description (ADO property name): Support for querying by property values in an MDX statement

Notes:

For more information, see the "OLE DB for Online Analytical Processing (OLAP)" section of the *OLE DB Programmer's Reference and Data Access SDK*. Also see the *SAS OLAP Server: MDX Guide*.

MDPROP_MDX_SET_FUNCTIONS

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): Support for various set functions

Notes:

For more information, see the "OLE DB for Online Analytical Processing (OLAP)" section of the *OLE DB Programmer's Reference and Data Access SDK*. Also see the *SAS OLAP Server: MDX Guide*.

MDPROP_MDX_SLICER

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): The capabilities in the WHERE clause of an MDX statement

Notes:

For more information, see the "OLE DB for Online Analytical Processing (OLAP)" section of the *OLE DB Programmer's Reference and Data Access SDK*. Also see the *SAS OLAP Server: MDX Guide*.

MDPROP_MDX_STRING_COMPOP

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): Support for string comparison operators other than equals and not-equals operators

Notes:

For more information, see the "OLE DB for Online Analytical Processing (OLAP)" section of the *OLE DB Programmer's Reference and Data Access SDK*. Also see the *SAS OLAP Server: MDX Guide*.

MDPROP_NAMED_LEVELS

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): Support for named levels

Notes:

For more information, see the "OLE DB for Online Analytical Processing (OLAP)" section of the *OLE DB Programmer's Reference and Data Access SDK*. Also see the *SAS OLAP Server: MDX Guide*.

MDPROP_RANGEROWSET

Supported in: IOM Provider, OLAP Provider

Property Group: Data Source Information

Property Set: DBPROPSET_DATASOURCEINFO

Read: Yes

Write: No

Column: False

Type: VT_I4

Description (ADO property name): Support for cell updates

Notes:

For more information, see the "OLE DB for Online Analytical Processing (OLAP)" section of the *OLE DB Programmer's Reference and Data Access SDK*. Also see the *SAS OLAP Server: MDX Guide*.

OLE DB Properties: Sorted by ADO Name

Note: Properties specific to SAS begin with SAS (for example, SAS Formats).

- ["Append-Only Rowset" on page 153](#)
- ["Asynchronous Processing" on page 160](#)
- ["Autocommit Isolation Levels" on page 192](#)
- ["Bookmark Type" on page 155](#)
- ["Bookmarks Ordered" on page 170](#)
- ["ClientCursor" on page 157](#)
- ["Current Catalog" on page 157](#)
- ["DBMS Name" on page 158](#)
- ["DBMS Version" on page 158](#)
- ["Data Source" on page 160](#)
- ["Data Source Type" on page 158](#)
- ["Extended Properties" on page 163](#)
- ["Fetch Backwards" on page 156](#)
- ["Flattening Support" on page 194](#)
- ["Hold Rows" on page 156](#)
- ["IAccessor" on page 159](#)
- ["IColumnsInfo" on page 159](#)
- ["IConvertType" on page 160](#)
- ["IRowset" on page 163](#)
- ["IRowsetChange" on page 164](#)
- ["IRowsetIdentity" on page 164](#)
- ["IRowsetInfo" on page 164](#)
- ["IRowsetLocate" on page 165](#)
- ["IRowsetUpdate" on page 165](#)
- ["IRowsetView" on page 165](#)
- ["ISupportErrorInfo" on page 166](#)
- ["IViewFilter" on page 166](#)
- ["IViewRowset" on page 166](#)

- "IViewSort" on page 167
- "Isolation Levels" on page 192
- "Literal Bookmarks" on page 167
- "Literal Row Identity" on page 167
- "Locale Identifier" on page 161
- "Location" on page 162
- "Lock Mode" on page 168
- "MDX Calculated Members Extensions" on page 170
- "Maximum OR Conditions" on page 168
- "Maximum Open Rows" on page 168
- "Maximum Pending Rows" on page 169
- "Maximum Rows" on page 169
- "Maximum Sort Columns" on page 169
- "Mode" on page 162
- "Number of axes in the data set" on page 194
- "Others' Changes Visible" on page 171
- "Others' Inserts Visible" on page 170
- "Own Changes Visible" on page 171
- "Own Inserts Visible" on page 171
- "Pass By Ref Accessors" on page 155
- "Password" on page 153
- "Prompt" on page 163
- "Provider Version" on page 172
- "Provider's ability to qualify a cube name" on page 197
- "Read-Only Data Source" on page 157
- "Remove Deleted Rows" on page 172
- "SAS Alter Password" on page 172
- "SAS Cell Cache Size" on page 178
- "SAS Data Set Encoding" on page 173
- "SAS Data Set Label" on page 173
- "SAS Data Set Type" on page 174
- "SAS Data Set Windows Code Page" on page 174
- "SAS Data Set Options" on page 174
- "SAS Default File Format" on page 175
- "SAS Executable" on page 181
- "SAS File Format" on page 178
- "SAS Format Error" on page 176

- "SAS Formats" on page 176
- "SAS Get Missing Values Grid" on page 177
- "SAS Informats" on page 177
- "SAS Library Engine" on page 175
- "SAS Library Options" on page 185
- "SAS Local Server" on page 179
- "SAS Logical Name" on page 180
- "SAS Machine DNS Name" on page 180
- "SAS Missing Values" on page 185
- "SAS NLS Options" on page 186
- "SAS Optimistic Locking" on page 186
- "SAS PT2DBPW" on page 188
- "SAS Page Size" on page 186
- "SAS Parameters" on page 182
- "SAS Physical Path" on page 187
- "SAS Physical Positioning" on page 187
- "SAS Port" on page 180
- "SAS Preserve Trailing Blanks" on page 173
- "SAS Protocol" on page 181
- "SAS Read Password" on page 188
- "SAS Reserved 1" on page 189
- "SAS SQL Engine" on page 189
- "SAS SQL Engine Options" on page 189
- "SAS Server Access Password" on page 182
- "SAS Server Release" on page 183
- "SAS Server Type" on page 183
- "SAS Service Name" on page 183
- "SAS Where" on page 190
- "SAS Working Directory" on page 182
- "SAS Workspace ID" on page 184
- "SAS Workspace Init List" on page 191
- "SAS Workspace Init Log" on page 191
- "SAS Workspace Init Script" on page 185
- "SAS Workspace Interface" on page 184
- "SAS Write Password" on page 191
- "Scroll Backwards" on page 156
- "Skip Deleted Bookmarks" on page 155

- "Sort on Index" on page 192
- "Support for MDX case statements on page 195
- "Support for cell updates" on page 199
- "Support for creation of named sets and calculated members" on page 195
- "Support for named levels" on page 199
- "Support for outer reference in an MDX statement" on page 197
- "Support for query joining multiple cubes" on page 196
- "Support for querying by property values in an MDX statement on page 197
- "Support for string comparison operators other than equals and not-equals operators" on page 199
- "Support for updating aggregated cells" on page 193
- "Support for various <desc flag> values in the DESCENDANTS function" on page 195
- "Support for various member functions" on page 196
- "Support for various numeric functions" on page 196
- "Support for various set functions" on page 198
- "The capabilities in the WHERE clause of an MDX statement" on page 198
- "UniqueRows" on page 193
- "Updatability" on page 193
- "Use Bookmarks" on page 154
- "Use TK Manager Search Path" on page 190
- "User ID" on page 154
- "Window Handle" on page 161

OLE DB Properties: Sorted by Data Provider

Supported Properties in the IOM and OLAP Data Providers

- "DBPROP_APPENDONLY" on page 153
- "DBPROP_AUTH_PASSWORD" on page 153
- "DBPROP_AUTH_USERID" on page 154
- "DBPROP_BOOKMARKS" on page 154
- "DBPROP_BOOKMARKSKIPPED" on page 155
- "DBPROP_BOOKMARKTYPE" on page 155
- "DBPROP_BYREFACCESSORS" on page 155
- "DBPROP_CANFETCHBACKWARDS" on page 156
- "DBPROP_CANHOLDROWS" on page 156
- "DBPROP_CANSCROLLBACKWARDS" on page 156

- “DBPROP_CURRENTCATALOG” on page 157
- “DBPROP_DATASOURCEREADONLY” on page 157
- “DBPROP_DATASOURCE_TYPE” on page 158
- “DBPROP_DBMSNAME” on page 158
- “DBPROP_DBMSVER” on page 158
- “DBPROP_IAccessor” on page 159
- “DBPROP_IColumnsInfo” on page 159
- “DBPROP_IConvertType” on page 160
- “DBPROP_INIT_ASYNC” on page 160
- “DBPROP_INIT_DATASOURCE” on page 160
- “DBPROP_INIT_HWND” on page 161
- “DBPROP_INIT_LCID” on page 161
- “DBPROP_INIT_LOCATION” on page 162
- “DBPROP_INIT_MODE” on page 162
- “DBPROP_INIT_PROMPT” on page 163
- “DBPROP_INIT_PROVIDERSTRING” on page 163
- “DBPROP_IRowset” on page 163
- “DBPROP_IRowsetChange” on page 164
- “DBPROP_IRowsetIdentity” on page 164
- “DBPROP_IRowsetInfo” on page 164
- “DBPROP_IRowsetLocate” on page 165
- “DBPROP_IRowsetUpdate” on page 165
- “DBPROP_ISupportErrorInfo” on page 166
- “DBPROP_LITERALBOOKMARKS” on page 167
- “DBPROP_LITERALIDENTITY” on page 167
- “DBPROP_LOCKMODE” on page 168
- “DBPROP_MAXOPENROWS” on page 168
- “DBPROP_MAXPENDINGROWS” on page 169
- “DBPROP_MAXROWS” on page 169
- “DBPROP_MSMD_MDX_CALCMEMB_EXTENSIONS” on page 170
- “DBPROP_ORDEREDBOOKMARKS” on page 170
- “DBPROP_OTHERINSERT” on page 170
- “DBPROP_OTHERUPDELETEDELETE” on page 171
- “DBPROP_OWNINSERT” on page 171
- “DBPROP_OWNUPDELETEDELETE” on page 171
- “DBPROP_PROVIDERSERVER” on page 172
- “DBPROP_SAS_ALTERPASSWORD” on page 172

- “DBPROP_SAS_BLANKPADDING” on page 173
- “DBPROP_SAS_DATASETOPTS” on page 174
- “DBPROP_SAS_ENGINE” on page 175
- “DBPROP_SAS_FORMATS” on page 176
- “DBPROP_SAS_INFORMATS” on page 177
- “DBPROP_SAS_INIT_CELLCACHE SIZE” on page 178
- “DBPROP_SAS_INIT_LOGICALNAME” on page 180
- “DBPROP_SAS_INIT_MACHINEDNSNAME” on page 180
- “DBPROP_SAS_INIT_PORT” on page 180
- “DBPROP_SAS_INIT_PROTOCOL” on page 181
- “DBPROP_SAS_INIT_SERVERTYPE” on page 183
- “DBPROP_SAS_INIT_SERVICENAME” on page 183
- “DBPROP_SAS_INIT_WORKSPACE” on page 184
- “DBPROP_SAS_INIT_WORKSPACEID” on page 184
- “DBPROP_SAS_INIT_WORKSPACE_INIT_SCRIPT” on page 185
- “DBPROP_SAS_LIBOPTS” on page 185
- “DBPROP_SAS_MISSING_VALUES” on page 185
- “DBPROP_SAS_OPTIMISTICLOCKING” on page 186
- “DBPROP_SAS_PAGESIZE” on page 186
- “DBPROP_SAS_PATH” on page 187
- “DBPROP_SAS_PHYSICALPOSITIONING” on page 187
- “DBPROP_SAS_READPASSWORD” on page 188
- “DBPROP_SAS_RESERVED_ROWSETFLAGS” on page 189
- “DBPROP_SAS_SQLCONNECTIONSTRING” on page 189
- “DBPROP_SAS_SQLENGINE” on page 189
- “DBPROP_SAS_WORKSPACE_INIT_LIST” on page 191
- “DBPROP_SAS_WORKSPACE_INIT_LOG” on page 191
- “DBPROP_SAS_WRITEPASSWORD” on page 191
- “DBPROP_SESS_AUTOCOMMITISOLEVELS” on page 192
- “DBPROP_SUPPORTEDTXNISOLEVELS” on page 192
- “DBPROP_UPDATABILITY” on page 193
- “MDPROP_AGGREGATECELL_UPDATE” on page 193
- “MDPROP_AXES” on page 194
- “MDPROP_FLATTENING_SUPPORT” on page 194
- “MDPROP_MDX_CASESUPPORT” on page 195
- “MDPROP_MDX_DESCFLAGS” on page 195
- “MDPROP_MDX_FORMULAS” on page 195

- “MDPROP_MDX_JOINCUBES” on page 196
- “MDPROP_MDX_MEMBER_FUNCTIONS” on page 196
- “MDPROP_MDX_NUMERIC_FUNCTIONS” on page 196
- “MDPROP_MDX_OBJQUALIFICATION” on page 197
- “MDPROP_MDX_OUTERREFERENCE” on page 197
- “MDPROP_MDX_QUERYBYPROPERTY” on page 197
- “MDPROP_MDX_SET_FUNCTIONS” on page 198
- “MDPROP_MDX_SLICER” on page 198
- “MDPROP_MDX_STRING_COMPOP” on page 199
- “MDPROP_NAMED_LEVELS” on page 199
- “MDPROP_RANGEROWSET” on page 199

Supported Properties in the Local Data Provider

- “DBPROP_APPENDONLY” on page 153
- “DBPROP_AUTH_PASSWORD” on page 153
- “DBPROP_AUTH_USERID” on page 154
- “DBPROP_BOOKMARKS” on page 154
- “DBPROP_BOOKMARKSKIPPED” on page 155
- “DBPROP_BOOKMARKTYPE” on page 155
- “DBPROP_BYREFACCESSORS” on page 155
- “DBPROP_CANFETCHBACKWARDS” on page 156
- “DBPROP_CANHOLDROWS” on page 156
- “DBPROP_CANSCROLLBACKWARDS” on page 156
- “DBPROP_DATASOURCEREADONLY” on page 157
- “DBPROP_DBMSNAME” on page 158
- “DBPROP_DBMSVER” on page 158
- “DBPROP_IAccessor” on page 159
- “DBPROP_IColumnsInfo” on page 159
- “DBPROP_IConvertType” on page 160
- “DBPROP_INIT_DATASOURCE” on page 160
- “DBPROP_INIT_LOCATION” on page 162
- “DBPROP_INIT_MODE” on page 162
- “DBPROP_INIT_PROMPT” on page 163
- “DBPROP_INIT_PROVIDERSTRING” on page 163
- “DBPROP_IRowset” on page 163
- “DBPROP_IRowsetChange” on page 164
- “DBPROP_IRowsetIdentity” on page 164

- “DBPROP_IRowsetInfo” on page 164
- “DBPROP_IRowsetLocate” on page 165
- “DBPROP_IRowsetUpdate” on page 165
- “DBPROP_IRowsetView” on page 165
- “DBPROP_ISupportErrorInfo” on page 166
- “DBPROP_IViewFilter” on page 166
- “DBPROP_IViewRowset” on page 166
- “DBPROP_IViewSort” on page 167
- “DBPROP_LITERALBOOKMARKS” on page 167
- “DBPROP_LITERALIDENTITY” on page 167
- “DBPROP_MAXOPENROWS” on page 168
- “DBPROP_MAXORSINFILTER” on page 168
- “DBPROP_MAXPENDINGROWS” on page 169
- “DBPROP_MAXROWS” on page 169
- “DBPROP_MAXSORTCOLUMNS” on page 169
- “DBPROP_ORDEREDBOOKMARKS” on page 170
- “DBPROP_OTHERINSERT” on page 170
- “DBPROP_OTHERUPDATEDELETE” on page 171
- “DBPROP_OWNINSERT” on page 171
- “DBPROP_OWNUPDATEDELETE” on page 171
- “DBPROP_PROVIDERVER” on page 172
- “DBPROP_REMOVEDELETED” on page 172
- “DBPROP_SAS_ALTERPASSWORD” on page 172
- “DBPROP_SAS_BLANKPADDING” on page 173
- “DBPROP_SAS_DATASET_ENCODING” on page 173
- “DBPROP_SAS_DATASET_LABEL” on page 173
- “DBPROP_SAS_DATASET_TYPE” on page 174
- “DBPROP_SAS_DATASET_WINDOWS_CODEPAGE” on page 174
- “DBPROP_SAS_DEFAULTFILEFORMAT” on page 175
- “DBPROP_SAS_FMTERR” on page 176
- “DBPROP_SAS_FORMATS” on page 176
- “DBPROP_SAS_GET_MISSING_VALUES” on page 177
- “DBPROP_SAS_INFORMATS” on page 177
- “DBPROP_SAS_INIT_FILEFORMAT” on page 178
- “DBPROP_SAS_MISSING_VALUES” on page 185
- “DBPROP_SAS_NLSOPTS” on page 186
- “DBPROP_SAS_READPASSWORD” on page 188

- “DBPROP_SAS_USE_TKMANAGER_SEARCHPATH” on page 190
- “DBPROP_SAS_WRITEPASSWORD” on page 191
- “DBPROP_SESS_AUTOCOMMITISOLEVELS” on page 192
- “DBPROP_SORTONINDEX” on page 192
- “DBPROP_SUPPORTEDTXNISOLEVELS” on page 192
- “DBPROP_UPDATABILITY” on page 193

Supported Properties in the SAS/SHARE and Base SAS Data Providers

- “DBPROP_APPENDONLY” on page 153
- “DBPROP_AUTH_PASSWORD” on page 153
- “DBPROP_AUTH_USERID” on page 154
- “DBPROP_BOOKMARKS” on page 154
- “DBPROP_BOOKMARKSKIPPED” on page 155
- “DBPROP_BOOKMARKTYPE” on page 155
- “DBPROP_BYREFACCESSORS” on page 155
- “DBPROP_CANFETCHBACKWARDS” on page 156
- “DBPROP_CANHOLDROWS” on page 156
- “DBPROP_CANSROLLBACKWARDS” on page 156
- “DBPROP_CLIENTCURSOR” on page 157
- “DBPROP_DATASOURCEREADONLY” on page 157
- “DBPROP_DBMSNAME” on page 158
- “DBPROP_DBMSVER” on page 158
- “DBPROP_IAccessor” on page 159
- “DBPROP_IColumnsInfo” on page 159
- “DBPROP_IConvertType” on page 160
- “DBPROP_INIT_DATASOURCE” on page 160
- “DBPROP_INIT_LOCATION” on page 162
- “DBPROP_INIT_MODE” on page 162
- “DBPROP_INIT_PROMPT” on page 163
- “DBPROP_INIT_PROVIDERSTRING” on page 163
- “DBPROP_IRowset” on page 163
- “DBPROP_IRowsetChange” on page 164
- “DBPROP_IRowsetIdentity” on page 164
- “DBPROP_IRowsetInfo” on page 164
- “DBPROP_IRowsetLocate” on page 165
- “DBPROP_IRowsetUpdate” on page 165

- “DBPROP_I\SupportErrorInfo” on page 166
- “DBPROP_LITERALBOOKMARKS” on page 167
- “DBPROP_LITERALIDENTITY” on page 167
- “DBPROP_MAXOPENROWS” on page 168
- “DBPROP_MAXPENDINGROWS” on page 169
- “DBPROP_MAXROWS” on page 169
- “DBPROP_ORDEREDBOOKMARKS” on page 170
- “DBPROP_OTHERINSERT” on page 170
- “DBPROP_OTHERUPDELETEDELETE” on page 171
- “DBPROP_OWNINSERT” on page 171
- “DBPROP_OWNUPDELETEDELETE” on page 171
- “DBPROP_PROVIDERVER” on page 172
- “DBPROP_SAS_ALTERPASSWORD” on page 172
- “DBPROP_SAS_BLANKPADDING” on page 173
- “DBPROP_SAS_DATASET_WINDOWS_CODEPAGE” on page 174
- “DBPROP_SAS_FMTERR” on page 176
- “DBPROP_SAS_FORMATS” on page 176
- “DBPROP_SAS_INFORMATS” on page 177
- “DBPROP_SAS_INIT_LOCALSERVER” on page 179
- “DBPROP_SAS_INIT_SASEXE” on page 181
- “DBPROP_SAS_INIT_SASPARAMETERS” on page 182
- “DBPROP_SAS_INIT_SASWORKINGDIR” on page 182
- “DBPROP_SAS_INIT_SERVERPASSWORD” on page 182
- “DBPROP_SAS_INIT_SERVERRELEASE” on page 183
- “DBPROP_SAS_MISSING_VALUES” on page 185
- “DBPROP_SAS_OPTIMISTICLOCKING” on page 186
- “DBPROP_SAS_PT2DBPW” on page 188
- “DBPROP_SAS_READPASSWORD” on page 188
- “DBPROP_SAS_SQLCONNECTIONSTRING” on page 189
- “DBPROP_SAS_SQLENGINE” on page 189
- “DBPROP_SAS_WHERE” on page 190
- “DBPROP_SAS_WRITEPASSWORD” on page 191
- “DBPROP_SESS_AUTOCOMMITISOLEVELS” on page 192
- “DBPROP_SUPPORTEDTXNISOLEVELS” on page 192
- “DBPROP_UNIQUEROWS” on page 193
- “DBPROP_UPDATABILITY” on page 193

OLE DB Properties: Sorted by Group

Supported Properties in the Data Source Group

- DBPROPSET_DATASOURCE
 - “DBPROP_CURRENTCATALOG” on page 157

Supported Properties in the Data Source Information Group

- DBPROPSET_DATASOURCEINFO
 - “DBPROP_BYREFACCESSORS” on page 155
 - “DBPROP_DATASOURCEREADONLY” on page 157
 - “DBPROP_DATASOURCE_TYPE” on page 158
 - “DBPROP_DBMSNAME” on page 158
 - “DBPROP_DBMSVER” on page 158
 - “DBPROP_PROVIDERVER” on page 172
 - “DBPROP_SUPPORTEDTXNISOLEVELS” on page 192
 - “MDPROP_AGGREGATECELL_UPDATE” on page 193
 - “MDPROP_AXES” on page 194
 - “MDPROP_FLATTENING_SUPPORT” on page 194
 - “MDPROP_MDX_CASESUPPORT” on page 195
 - “MDPROP_MDX_DESCFLAGS” on page 195
 - “MDPROP_MDX_FORMULAS” on page 195
 - “MDPROP_MDX_JOINCUBES” on page 196
 - “MDPROP_MDX_MEMBER_FUNCTIONS” on page 196
 - “MDPROP_MDX_NUMERIC_FUNCTIONS” on page 196
 - “MDPROP_MDX_OBJQUALIFICATION” on page 197
 - “MDPROP_MDX_OUTERREFERENCE” on page 197
 - “MDPROP_MDX_QUERYBYPROPERTY” on page 197
 - “MDPROP_MDX_SET_FUNCTIONS” on page 198
 - “MDPROP_MDX_SLICER” on page 198
 - “MDPROP_MDX_STRING_COMPOP” on page 199
 - “MDPROP_NAMED_LEVELS” on page 199
 - “MDPROP_RANGEROWSET” on page 199
- DBPROPSET_MDX_EXTENSIONS
 - “DBPROP_MSMD_MDX_CALCMEMB_EXTENSIONS” on page 170

- DBPROPSET_SAS_DATASOURCEINFO
 - “DBPROP_SAS_DEFAULTFILEFORMAT” on page 175
 - “DBPROP_SAS_USE_TKMANAGER_SEARCHPATH” on page 190
 - “DBPROP_SAS_WORKSPACE_INIT_LIST” on page 191
 - “DBPROP_SAS_WORKSPACE_INIT_LOG” on page 191

Supported Properties in the Initialization Group

- DBPROPSET_DBINIT
 - “DBPROP_AUTH_PASSWORD” on page 153
 - “DBPROP_AUTH_USERID” on page 154
 - “DBPROP_INIT_ASYNC” on page 160
 - “DBPROP_INIT_DATASOURCE” on page 160
 - “DBPROP_INIT_HWND” on page 161
 - “DBPROP_INIT_LCID” on page 161
 - “DBPROP_INIT_LOCATION” on page 162
 - “DBPROP_INIT_MODE” on page 162
 - “DBPROP_INIT_PROMPT” on page 163
 - “DBPROP_INIT_PROVIDERSTRING” on page 163
- DBPROPSET_SAS_DBINIT
 - “DBPROP_SAS_INIT_CELLCACHESIZE” on page 178
 - “DBPROP_SAS_INIT_FILEFORMAT” on page 178
 - “DBPROP_SAS_INIT_LOCALSERVER” on page 179
 - “DBPROP_SAS_INIT_LOGICALNAME” on page 180
 - “DBPROP_SAS_INIT_MACHINEDNSNAME” on page 180
 - “DBPROP_SAS_INIT_PORT” on page 180
 - “DBPROP_SAS_INIT_PROTOCOL” on page 181
 - “DBPROP_SAS_INIT_SASEXE” on page 181
 - “DBPROP_SAS_INIT_SASPARAMETERS” on page 182
 - “DBPROP_SAS_INIT_SASWORKINGDIR” on page 182
 - “DBPROP_SAS_INIT_SERVERPASSWORD” on page 182
 - “DBPROP_SAS_INIT_SERVERRELEASE” on page 183
 - “DBPROP_SAS_INIT_SERVERTYPE” on page 183
 - “DBPROP_SAS_INIT_SERVICENAME” on page 183
 - “DBPROP_SAS_INIT_WORKSPACE” on page 184
 - “DBPROP_SAS_INIT_WORKSPACEID” on page 184
 - “DBPROP_SAS_INIT_WORKSPACE_INIT_SCRIPT” on page 185
- DBPROPSET_SAS_ROWSET

- “DBPROP_SAS_ENGINE” on page 175
- “DBPROP_SAS_LIBOPTS” on page 185
- “DBPROP_SAS_PATH” on page 187

Supported Properties in the Rowset Group

- DBPROPSET_ROWSET
 - “DBPROP_APPENDONLY” on page 153
 - “DBPROP_BOOKMARKS” on page 154
 - “DBPROP_BOOKMARKSKIPPED” on page 155
 - “DBPROP_BOOKMARKTYPE” on page 155
 - “DBPROP_CANFETCHBACKWARDS” on page 156
 - “DBPROP_CANHOLDROWS” on page 156
 - “DBPROP_CANSROLLBACKWARDS” on page 156
 - “DBPROP_CLIENTCURSOR” on page 157
 - “DBPROP_IAccessor” on page 159
 - “DBPROP_IColumnsInfo” on page 159
 - “DBPROP_IConvertType” on page 160
 - “DBPROP_IRowset” on page 163
 - “DBPROP_IRowsetChange” on page 164
 - “DBPROP_IRowsetIdentity” on page 164
 - “DBPROP_IRowsetInfo” on page 164
 - “DBPROP_IRowsetLocate” on page 165
 - “DBPROP_IRowsetUpdate” on page 165
 - “DBPROP_IRowsetView” on page 165
 - “DBPROP_ISupportErrorInfo” on page 166
 - “DBPROP_LITERALBOOKMARKS” on page 167
 - “DBPROP_LITERALIDENTITY” on page 167
 - “DBPROP_LOCKMODE” on page 168
 - “DBPROP_MAXOPENROWS” on page 168
 - “DBPROP_MAXPENDINGROWS” on page 169
 - “DBPROP_MAXROWS” on page 169
 - “DBPROP_ORDEREDBOOKMARKS” on page 170
 - “DBPROP_OTHERINSERT” on page 170
 - “DBPROP_OTHERUPDATEDELETE” on page 171
 - “DBPROP_OWNINSERT” on page 171
 - “DBPROP_OWNUPDATEDELETE” on page 171
 - “DBPROP_REMOVEDELETED” on page 172

- “DBPROP_UNIQUEROWS” on page 193
- “DBPROP_UPDATABILITY” on page 193
- DBPROPSET_SAS_ROWSET
 - “DBPROP_SAS_ALTERPASSWORD” on page 172
 - “DBPROP_SAS_BLANKPADDING” on page 173
 - “DBPROP_SAS_DATASET_ENCODING” on page 173
 - “DBPROP_SAS_DATASET_LABEL” on page 173
 - “DBPROP_SAS_DATASET_TYPE” on page 174
 - “DBPROP_SAS_DATASET_WINDOWS_CODEPAGE” on page 174
 - “DBPROP_SAS_DATASETOPTS” on page 174
 - “DBPROP_SAS_FMTERR” on page 176
 - “DBPROP_SAS_FORMATS” on page 176
 - “DBPROP_SAS_GET_MISSING_VALUES” on page 177
 - “DBPROP_SAS_INFORMATS” on page 177
 - “DBPROP_SAS_MISSING_VALUES” on page 185
 - “DBPROP_SAS_NLSOPTS” on page 186
 - “DBPROP_SAS_OPTIMISTICLOCKING” on page 186
 - “DBPROP_SAS_PAGESIZE” on page 186
 - “DBPROP_SAS_PHYSICALPOSITIONING” on page 187
 - “DBPROP_SAS_READPASSWORD” on page 188
 - “DBPROP_SAS_RESERVED_ROWSETFLAGS” on page 189
 - “DBPROP_SAS_WHERE” on page 190
 - “DBPROP_SAS_WRITEPASSWORD” on page 191
- DBPROPSET_VIEW
 - “DBPROP_IViewFilter” on page 166
 - “DBPROP_IViewRowset” on page 166
 - “DBPROP_IViewSort” on page 167
 - “DBPROP_MAXORSINFILTER” on page 168
 - “DBPROP_MAXSORTCOLUMNS” on page 169
 - “DBPROP_SORTONINDEX” on page 192

Supported Properties in the Session Group

- DBPROPSET_SAS_SESSION
 - “DBPROP_SAS_PT2DBPW” on page 188
 - “DBPROP_SAS_SQLCONNECTIONSTRING” on page 189
 - “DBPROP_SAS_SQLENGINE” on page 189

- DBPROPSET_SESSION
 - “DBPROP_SESS_AUTOCOMMITISOLEVELS” on page 192

Appendix 4

OLE DB Interfaces

About OLE DB Interfaces	215
What Are OLE DB Interfaces?	215
Standard OLE DB Interfaces	216
Command Object	216
Data Source Object	216
Rowset Object	216
Session Object	217
OLE DB for OLAP Interfaces	217
Dataset Object	217
Custom Interfaces	218
About Custom Interfaces	218
ISASColumnsInfo Custom Interface	218
ISASDataSetInfo Custom Interface	220
ISASDataSetInfo90 Custom Interface	223
Data Set Management Using the ITableDefinition Interface	225
About Creating and Deleting SAS Data Sets	225
Creating a Data Set	225
Deleting Data Sets	227

About OLE DB Interfaces

What Are OLE DB Interfaces?

An OLE DB data provider is a COM component composed of a collection of interfaces. Each OLE DB interface defines a set of related methods that OLE DB clients can call. The OLE DB specification defines a set of standard OLE DB interfaces. The OLAP provider also implements the OLE DB for OLAP interfaces. In addition, the following custom rowset interfaces supplement the standard OLE DB interfaces.

- ISASColumnsInfo Custom Interface
- ISASDataSetInfo Custom Interface
- ISASDataSetInfo90 Custom Interface

The following topics explain how to use the interfaces:

- [“Standard OLE DB Interfaces” on page 216](#)

- “OLE DB for OLAP Interfaces” on page 217
- “Custom Interfaces” on page 218

Standard OLE DB Interfaces

Command Object

The Command object implementation does not support command processing. The implementation accommodates a limitation in ADO support for custom rowset properties. All of the command interface methods, except for those methods on ICommandProperties, return E_FAIL. Here are the Command object interfaces:

- IColumnsInfo
- ICommand¹
- ICommandPrepare
- ICommandProperties
- ICommandText¹
- IConvertType
- ISupportErrorInfo

¹ The ICommand and ICommandText interfaces support the Cancel method for the OLAP Provider. This support is limited to MDX queries that are made asynchronously and with the Execute method.

Data Source Object

These interfaces are supported on the Data Source object:

- IDBCreateSession
- IDBInitialize
- IDBProperties
- IPersist
- ISupportErrorInfo

Rowset Object

These interfaces are supported on the Rowset object with the following restrictions:

- For the SAS/SHARE, IOM, and Base SAS providers, the IRowsetLocate, IRowsetChange, and IRowsetUpdate interfaces are not supported on Rowset objects that are returned by Command objects. They are supported only on Rowset objects that are returned by IOpenRowset::OpenRowset.
- For the SAS/SHARE and Base SAS providers, this same restriction is true for IRowsetIdentity.
- IAccessor

- IColumnsInfo
- IConvertType
- IRowset
- IRowsetChange
- IRowsetIdentity
- IRowsetInfo
- IRowsetLocate
- IRowsetUpdate
- ISupportErrorInfo

Session Object

These interfaces are supported on the Session object:

- IDBCreateCommand

Note: IDBCreateCommand is present only to accommodate a limitation in ADO support for custom rowset properties. IDBCreateCommand methods return E_FAIL.

- IDBSchemaRowset
- IGetDataSource
- IOpenRowset
- ISessionProperties
- ISupportErrorInfo
- ITableDefinition

See Also

[“Data Set Management Using the ITableDefinition Interface” on page 225](#)

OLE DB for OLAP Interfaces

Dataset Object

These interfaces are supported only by the OLAP provider:

- IAccessor
- IColumnsInfo
- IConvertType
- IMDDataset
- IMDFind

Custom Interfaces

About Custom Interfaces

SAS provides three custom interfaces that can be used to return data set metadata that does not directly map to OLE DB constructs.

- The ISASColumnsInfo interface has one method.
- The ISASDataSetInfo interface has two methods.
- The ISASDataSetInfo90 interface is an extension to the ISASDataSetInfo interface. The ISASDataSetInfo90 interface has four methods. Two are shared with the ISASDataSetInfo interface and two are unique to the ISASDataSetInfo90 interface.

ISASColumnsInfo Custom Interface

The GetColumnInfo Method

The single ISASColumnsInfo interface method is called GetColumnInfo.

GetColumnInfo supplements the standard OLE DB IColumnsInfo interface methods. It returns SAS column metadata that is not supported by the GetColumnInfo method in the standard IColumnsInfo interface.

```
HRESULT GetColumnInfo(
    DBORDINAL * pcColumns,
    SASCOLUMNINFO ** prgInfo,
    OLECHAR ** ppStringsBuffer );
```

GetColumnInfo returns a fixed set of column metadata in an array of SASCOLUMNINFO structures, one structure per column. The structures appear in the same order in which the columns appear in the rowset (column ordinal order). This order is determined by the order in which the columns are returned from IColumnsInfo::GetColumnInfo. Bookmark columns are never included in the output of this method. Here is the definition of the SASCOLUMNINFO structure:

```
typedef struct tagSASCOLUMNINFO {
    LPOLESTR pwszColDesc;
    LPOLESTR pwszFmtName;
    LPOLESTR pwszIFmtName;
    ULONG iOrdinal;
    SHORT iFmtLength;
    SHORT iFmtDecimal;
    SHORT iIFmtLength;
    SHORT iIFmtDecimal;
    SHORT iSortInfo;
    BOOL fIndexed;
} SASCOLUMNINFO;
```

The elements (members) of the SASCOLUMNINFO structure are described in the following table.

Table A4.1 Elements (Members) of the SASCOLUMNINFO Structure

Element	Description
pwszColDesc	Pointer to the column description (the SAS variable label). If no label is associated with this column, this member is NULL.
pwszFmtName	Pointer to the persisted format name. If no format is associated with this column, this member is NULL.
pwszIFmtName	Pointer to the persisted informat name. If no informat is associated with this column, this member is NULL.
iOrdinal	The ordinal of the column. This value corresponds to the SAS variable number.
iFmtLength	The width of the formatted data. This member is valid only when pwszFmtName is not NULL.
iFmtDecimal	The decimal width of the format data. This member is valid only when pwszFmtDecimal is not NULL.
iIFmtLength	The width of the informatted data. This member is valid only when pwszIFmtName is not NULL.
iIFmtDecimal	The decimal width of the informatted data. This member is valid only when pwszIFmtName is not NULL.
iSortInfo*	A signed short value that indicates the column's position in any applied sorting hierarchy. Positive values indicate ascending sort order, and negative values indicate descending sort order. The absolute value of the signed short value describes the position of the variable in the sorting hierarchy. Zero (0) indicates that the column does not participate in sorting.
fIndexed	True when this column is an indexed column.

* For the SAS/SHARE, Local, and Base SAS providers, this member is valid. For the SAS IOM provider, this member is not valid. That is, it always contains 0 whether the column participates in the sorting.

Here are the parameters for the GetColumnInfo method:

pcColumns

[out] A pointer to the memory where the number of columns in the rowset will be returned. This number will not include the bookmark column even if there is one. If this method terminates as the result of an error, *pcColumns is set to 0.

prgInfo

[out] A pointer to the memory where an array of SASCOLUMNINFO structures will be returned. One structure is returned for each column that is in the rowset. The provider allocates memory for the structures and returns the address of the memory location. When the memory is no longer needed by the column information, you use IMalloc::Free to release the memory. If *pcColumns is 0 when it is calculated or this method terminates as the result of an error, the provider does not allocate any memory and ensures that *prgInfo is a NULL pointer.

ppStringsBuffer

[out] A pointer to the memory where column string names will be returned. All of the column string values (names) are stored within a single allocation block. If no returned columns have string names or if this method terminates as the result of an

error, *ppStringsBuffer will be set to NULL. If one or more columns has a string name, then the specified memory location contains the string values (names) for the columns. When the names are no longer needed, you use IMalloc::Free to release the memory. If *pcColumns is 0 when calculated or this method terminates as the result of an error, the provider does not allocate any memory and ensures that *ppStringsBuffer is a NULL pointer. Each string value stored in this buffer is terminated by a null-termination character.

Here are the return codes for the GetColumnInfo method:

S_OK

indicates that the method succeeded.

E_FAIL

indicates that a provider-specific error occurred.

E_INVALIDARG

indicates that pcColumns, prgInfo, or ppStringsBuffer was a null pointer.

E_OUTOFMEMORY

indicates that the provider was unable to allocate sufficient memory for the column information structures.

TIP The GetColumnInfo method provides a quick alternative to the GetColumnsRowset method. While the GetColumnsRowset method returns all available column metadata, it does so in a rowset. To get the metadata, the consumer must create the column metadata rowset, create one or more accessors, get each row in the rowset, and get the data from the rowset.

ISASDataSetInfo Custom Interface

The GetDataSetInfo Method

The GetDataSetInfo method returns SAS file metadata that is not supported by predefined OLE DB constructs.

```
HRESULT GetDataSetInfo(
    SASDATASETINFO ** ppDataSetInfo,
    OLECHAR ** ppStringsBuffer );
```

The GetDataSetInfo method makes no logical change to the state of the object. The GetDataSetInfo method returns metadata about the file in a SASDATASETINFO structure.

```
typedef struct tagSASDATASETINFO {
    LONG lLogicalRecordCount;
    LONG lPhysicalRecordCount;
    LONG lRecordLength;
    DATE dDateCreated;
    DATE dDateModified;
    LPOLESTR pwszLabel;
    LPOLESTR pwszCompressionRoutine;
    BOOL fIsIndexed;
} SASDATASETINFO;
```

The elements (members) of the SASDATASETINFO structure are described in the following table:

Table A4.2 Elements (Members) of the SASDATASETINFO Structure

Element	Description
lLogicalRecordCount	The number of logical records in the data set. This value is the number of records that are returned if the caller sequentially reads through the entire data set. If this value is not immediately available, a value of -1 is returned in this field. This situation occurs when the data set is not a physical file (for example, a SAS data view or a WHERE clause). In such cases, you can use the GetRecordCounts method to computationally determine this value.
lPhysicalRecordCount	The number of physical records in the data set. This number can be greater than the number of logical records and can indicate the magnitude of the physical file size. If this value is not immediately available, a value of -1 is returned in this field. This situation occurs when the data set is not a physical SAS file (for example, a SAS data view or a SAS/ACCESS table). In such cases, you can use the GetRecordCounts method to computationally determine this value.
lRecordLength	The physical record length in bytes.
dDateCreated	The date that the data set was created.
dDateModified	The date that the data set was last modified.
pwszLabel	The data set label. NULL if none is set.
pwszCompressionRoutine	The name of the compression algorithm used. NO if none is set.
flsIndexed	TRUE if an index exists on the data set.

Under some circumstances, the actual values for the lLogicalRecordCount and lPhysicalRecordCount members of the SASDATASETINFO structure are not immediately available. This situation occurs when the underlying data set is something other than a physical SAS data file (for example, a SAS data view, a WHERE clause, or a SAS/ACCESS file). In those cases, the members return a -1. To determine the actual values, call ISASDataSetInfo::GetRecordCounts.

Here are the parameters for the GetDataSetInfo method:

ppDataSetInfo

[out] A pointer to the memory where a SASDATASETINFO structure will be returned. If *ppDataSetInfo is NULL when GetDataSetInfo is executed, then the provider allocates memory for this structure and returns the address to the memory location specified by this pointer. When the memory is no longer needed, you use IMalloc::Free to free the memory. As an alternative to depending on the provider to allocate memory for the SASDATASETINFO structure, the consumer can allocate memory for this structure and pass in the address as *ppDataSetInfo.

ppStringsBuffer

[out] A pointer to the memory where column string names will be returned. All of the column string values (names) are stored within a single allocation block. If no returned columns have string names, or if this method terminates as the result of an error, *ppStringsBuffer will be set to NULL. If one or more columns has a string name, then the specified memory location contains the string values (names) for the columns. When the names are no longer needed, you use IMalloc::Free to release the memory. If *pcColumns is 0 when calculated or this method terminates as the result

of an error, the provider does not allocate any memory and ensures that *ppStringsBuffer is a NULL pointer. Each string value stored in this buffer is terminated by a null-termination character.

Here are the return codes for the GetDataSetInfo method:

S_OK

indicates that the method succeeded.

E_FAIL

indicates that a provider-specific error occurred.

E_INVALIDARG

indicates that the ppDataSetInfo or the ppStringsBuffer was a null pointer.

E_OUTOFMEMORY

indicates that the provider was unable to allocate sufficient memory for the data set information structures.

The GetRecordCounts Method

The GetRecordCounts method determines the number of logical and physical records in a data set by forcing a sequential read of the data set. Call this method only when GetDataSetInfo returns -1 in the SASDATASETINFO ILogicalRecordCount and IPhysicalRecordCount fields.

```
HRESULT GetRecordCounts (
    LONGLONG * pLogicalRecordCount,
    LONGLONG * pPhysicalRecordCount );
```

CAUTION:

Executing this method on a very large data set might incur a significant performance cost.

Here are the parameters for the GetRecordCounts method:

pLogicalRecordCount

[out]

A pointer to the memory where the logical number of records in the data set will be returned. This number indicates how many rows are returned if every row is read once. If this value exceeds a LONGLONG, a value of -1 is returned in this field.

pPhysicalRecordCount

[out]

A pointer to the memory where the physical number of records in the data set will be returned. This number might be greater than the number of logical records. It indicates how many records are allocated in the data set. This value multiplied by the length of an individual record (IRecordLength returned by GetDataSetInfo) approximates the magnitude of the physical size of the data set. If this value exceeds a LONGLONG, a value of -1 is returned in this field.

Here are the return codes for the GetRecordCounts method:

S_OK

indicates that the method succeeded.

E_FAIL

indicates that a provider-specific error occurred.

E_INVALIDARG

indicates that pLogicalRecordCount or pPhysicalRecordCount was a null pointer.

ISASDataSetInfo90 Custom Interface

The GetDataSetInfo_2 Method

The GetDataSetInfo_2 method returns file metadata that is specific to SAS. It is a revised version of GetDataSetInfo that permits larger record counts.

```
HRESULT GetDataSetInfo_2(
    SASDATASETINFO90 ** ppDataSetInfo,
    OLECHAR ** ppStringsBuffer );
```

Note: See “The GetDataSetInfo Method” on page 220.

This method makes no logical change to the state of the [SAS file?] object. The GetDataSetInfo_2 method returns file metadata in a SASDATASETINFO90 structure.

```
typedef struct tagSASDATASETINFO90 {
    LONGLONG llLogicalRecordCount;
    LONGLONG llPhysicalRecordCount;
    LONG lRecordLength;
    DATE dDateCreated;
    DATE dDateModified;
    LPOLESTR pwszLabel;
    LPOLESTR pwszCompressionRoutine;
    BOOL fIsIndexed;
} SASDATASETINFO90;
```

The elements (members) of the SASDATASETINFO90 structure are described in the following table:

Table A4.3 Elements (Members) of the SASDATASETINFO90 Structure

Element	Description
llLogicalRecordCount	The number of logical records in the data set. This value is the number of records that are returned if the caller reads sequentially through the entire data set. If this value is not immediately available, a value of -1 is returned in this field. This situation occurs when the data set is not a physical file (for example, a SAS data view or a WHERE clause). In such cases, you can use the GetRecordCounts_2 method to computationally determine this value.
llPhysicalRecordCount	The number of physical records in the data set. This number can be greater than the number of logical records and can indicate the magnitude of the physical file size. If this value is not immediately available, a value of -1 is returned in this field. This situation occurs when the data set is not a physical SAS file (for example, a SAS data view or a SAS/ACCESS table). In such cases, you can use the GetRecordCounts_2 method to computationally determine this value.
lRecordLength	The physical record length in bytes.
dDateCreated	The date that the data set was created.
dDateModified	The date that the data set was last modified.
pwszLabel	The data set label. NULL if none is set.
pwszCompressionRoutine	The name of the compression algorithm used. NO if none is set.

Element	Description
flsIndexed	TRUE if an index exists on the data set.

Under some circumstances, the actual values for the `lLogicalRecordCount` and `lPhysicalRecordCount` members of the `SASDATASETINFO90` structure are not immediately available. This situation occurs when the underlying data set is something other than a physical SAS data file (for example, a SAS data view, a WHERE clause, or a SAS/ACCESS file). In those cases, the members return a -1. To determine the actual values, call `ISASDataSetInfo90::GetRecordCounts_2`.

Here are the parameters for the `GetDataSetInfo_2` method:

`ppDataSetInfo`

[out] A pointer to the memory where a `SASDATASETINFO90` structure will be returned. If `*ppDataSetInfo` is NULL when `GetDataSetInfo_2` is executed, then the provider allocates memory for this structure and returns the address to the memory location specified by this pointer. When the memory is no longer needed, you use `IMalloc::Free` to free the memory. As an alternative to depending on the provider to allocate memory for the `SASDATASETINFO90` structure, the consumer can allocate memory for this structure and pass in the address as `*ppDataSetInfo`.

`ppStringsBuffer`

[out] A pointer to the memory where column string names will be returned. All of the column string values (names) are stored within a single allocation block. If no returned columns have string names, or if this method terminates as the result of an error, `*ppStringsBuffer` will be set to NULL. If one or more columns has a string name, then the specified memory location contains the string values (names) for the columns. When the names are no longer needed, you use `IMalloc::Free` to release the memory. If `*pcColumns` is 0 when calculated or this method terminates as the result of an error, the provider does not allocate any memory and ensures that `*ppStringsBuffer` is a NULL pointer. Each string value stored in this buffer is terminated by a null-termination character.

Here are the return codes for the `GetDataSetInfo_2` method:

`S_OK`

indicates that the method succeeded.

`E_FAIL`

indicates that a provider-specific error occurred.

`E_INVALIDARG`

indicates that `ppDataSetInfo` or `ppStringsBuffer` was a null pointer.

`E_OUTOFMEMORY`

indicates that the provider was unable to allocate sufficient memory for the data set information structures.

The `GetRecordCounts_2` Method

The `GetRecordCounts_2` method determines a count of the number of logical and physical records in a data set by forcing a sequential read of the data set. Call this method only when `GetDataSetInfo_2` returns -1 in the `SASDATASETINFO90` `lLogicalRecordCount` and `lPhysicalRecordCount` fields.

```
HRESULT GetRecordCounts_2(
    LONGLONG * pllLogicalRecordCount,
    LONGLONG * pllPhysicalRecordCount );
```

Note: See “The GetRecordCounts Method” on page 222.

CAUTION:

Executing this method on a very large data set might incur a significant performance cost.

Here are the parameters for the GetRecordCounts_2 method:

pIILogicalRecordCount

[out] A pointer to the memory where the logical number of records in the data set will be returned. This number indicates how many rows are returned if every row is read once.

pIIPhysicalRecordCount

[out] A pointer to the memory where the physical number of records in the data set will be returned. This number might be greater than the number of logical records. It indicates how many records are allocated in the data set. This value multiplied by the length of an individual record (IRecordLength returned by GetDataSetInfo_2) approximates the magnitude of the physical size of the data set.

Here are the return codes for the GetRecordCounts_2 method:

S_OK

indicates that the method succeeded.

E_FAIL

indicates that a provider-specific error occurred.

E_INVALIDARG

indicates that pIILogicalRecordCount or pIIPhysicalRecordCount was a null pointer.

Data Set Management Using the ITableDefinition Interface

About Creating and Deleting SAS Data Sets

The SAS/SHARE, IOM, and Base SAS providers implement the CreateTable and DropTable methods that are available on the OLE DB ITableDefinition interface. These methods enable you to create new SAS data sets and delete existing ones.

Note: SAS does not support the ITableDefinition AddColumn or DropColumn methods.

Creating a Data Set

Process Overview

To create a SAS data set, you must provide the following information:

- the name of the new SAS data set (table)
- a description of the SAS variables (columns in the table)

You must also indicate whether you want to only create the physical data set or if you also want to return a rowset on that data set. If you want a rowset to be returned, you must provide property values for that rowset.

Naming the Data Set

Note: The form of the name must be *libname.membername*, where *libname* is a library defined by the server and *membername* is the data file or data view that you want to open within the library.

To specify the name of the new data set, you use the DBID structure defined by OLE DB. The following code shows how to set up a DBID structure for a new data set named MYDATA in a library named MYLIB.

```
DBID TableID;
TableID.eKind = DBKIND_NAME;
TableID.uName.pwszName = L"MYLIB.MYDATA";
```

The DBID structure uses two members, `eKind` and `pwszName`, which are the only two members supported by the SAS providers.

- The `eKind` member indicates where the provider should look within the DBID to find the name to use when the physical data set is created.
- `DBKIND_NAME` tells the provider to look only at the `pwszName` member of the `uName` union, which contains the data set name (with library reference, if applicable). The `pwszName` value is not case-sensitive.

Identifying the Variables

In this example data set, the following three variables (columns) are defined:

- FRIEND, a 20-byte character variable
- BIRTHDAY, a SAS date variable
- PHONE, an 8-byte character variable.

FRIEND and PHONE are represented as SAS character variables. BIRTHDAY must be represented as a double because that is how dates are stored in SAS data sets. SAS variable types are mapped to OLE DB DBTYPE indicators as follows:

- SAS numeric data is returned as a `DBTYPE_R8`.
- SAS character data is returned as a `DBTYPE_STR`.

Note: For more information about data types, see [“PROVIDER_TYPES Schema Rowset” on page 247](#).

The following code shows how to use the variable information to fill an array of `DBCOLUMNDESC` structures. Variable (column) names are specified by using a `DBID` structure in the same way that a data set (table) name is specified. The `dbcid` member of the `DBCOLUMNDESC` structure is where the variable (column) name is defined. The other members of the `DBCOLUMNDESC` structure are filled out based on the rules outlined by the OLE DB specification.

```
DBCOLUMNDESC rgColumnDescs[3];
memset( rgColumnDescs, '0', sizeof( rgColumnDescs ) ); // defensive programming
rgColumnDescs[0].dbcid.uName.pwszName = L"FRIEND";
rgColumnDescs[0].dbcid.eKind = DBKIND_NAME;
rgColumnDescs[0].wType = DBTYPE_STR;
rgColumnDescs[0].ulColumnSize = 20;
rgColumnDescs[1].dbcid.uName.pwszName = L"BIRTHDAY";
rgColumnDescs[1].dbcid.eKind = DBKIND_NAME;
rgColumnDescs[1].wType = DBTYPE_R8;
rgColumnDescs[1].bPrecision = 15;
rgColumnDescs[2].dbcid.uName.pwszName = L"PHONE";
rgColumnDescs[2].dbcid.eKind = DBKIND_NAME;
```

```
rgColumnDescs[2].wType = DBTYPE_STR;
rgColumnDescs[2].ulColumnSize = 8;
```

Returning a Rowset

To create a rowset on the new data set, you must specify the following information:

- the IID of the interface you want returned
- the address of an interface pointer

The following code completes this example. It can be used to create and return a sequential rowset—an approach that is useful if you want to create and populate the data set at the same time, but you do not intend to immediately update the new rows.

```
IDBCreateSession * pCreateSession;
ITableDefinition * pTableDefinition;
IUnknown * pRowset;
DBPROP prop;
DBPROPSET PropertySet;
pDSO->QueryInterface( IID_IDBCreateSession, );
pCreateSession->CreateSession( NULL, // no aggregation
    IID_ITableDefinition, // ID of the interface we want on the session object
    ); // address of pointer to session
// specify sequential access
prop.dwPropertyID = DBPROP_IRowsetLocate;
prop.dwOptions = DBPROPOPTIONS_OPTIONAL;
prop.colid = DB_NULLID;
prop.vValue.vt = VT_BOOL;
prop.vValue.bool = VARIANT_FALSE;
PropertySet.guidPropertySet = DBPROPSET_ROWSET;
PropertySet.cProperties = 1;
PropertySet.rgProperties =
pTableDefinition->CreateTable( NULL,
    // no aggregation
    ,
    3, // 3 columns
    rgColumnDescs, // column descriptions
    IID_IRowset, // id of interface we want on the rowset
    1, // we're passing in the rowset property set
    ,
    NULL, // ignored by the providers
    ); // address of pointer to rowset
```

Deleting Data Sets

You can use the DropTable method to delete a SAS data set from its associated data source. The definition of data source depends on the provider that is being used:

- For the Base SAS provider, a data source consists of all library names that are known to a local installation of Base SAS.
- For the SAS/SHARE provider, a data source consists of all library names that are known to a SAS/SHARE server.
- For the IOM provider, a data source consists of all library names that are known to a SAS Workspace Server.

CAUTION:

Be careful when you design an application that implements this method. If you give users the ability to erase data sets, you should also build in safeguards to prevent them from accidentally deleting data sets.

The DropTable method's single parameter is an instance of the DBID structure, which is used to specify the name of the data set (table) and its variables (columns). The following code shows how to set up a DBID structure to delete a data set named MYDATA in a library named MYLIB.

```
DBID TableID;
TableID.eKind = DBKIND_NAME;
TableID.uName.pwszName = L"MYLIB.MYDATA";
```

The following code shows how you would use this DBID structure to delete the data set:

```
IDBCreateSession * pCreateSession;
ITableDefinition * pTableDefinition;
pDSO->QueryInterface( IID_IDBCreateSession, );
pCreateSession->CreateSession( NULL, // no aggregation
    IID_ITableDefinition, // ID of the interface we want on the session object
    ); // address of pointer to session
pTableDefinition->DropTable( );
```

Appendix 5

Schema Rowsets

About Schema Rowsets	230
What Are Schema Rowsets?	230
Supported Schema Rowsets	230
How Are Schema Rowsets Obtained?	231
How to Restrict the Rows That Are Returned	232
CATALOGS Schema Rowset	232
Standard Mapping	232
COLUMNS Schema Rowset	232
Standard Mapping	232
Rowset Extensions	234
Additional Details	235
CUBES Schema Rowset	236
Standard Mapping	236
DIMENSIONS Schema Rowset	237
Standard Mapping	237
FUNCTIONS Schema Rowset	238
Standard Mapping	238
HIERARCHIES Schema Rowset	239
Standard Mapping	239
LEVELS Schema Rowset	240
Standard Mapping	240
MEASURES Schema Rowset	242
Standard Mapping	242
MEMBERS Schema Rowset	244
Standard Mapping	244
The Tree Operation Restriction	245
PROPERTIES Schema Rowset	246
Standard Mapping	246
PROVIDER_TYPES Schema Rowset	247
Standard Mapping	247
SETS Schema Rowset	249
Standard Mapping	249
TABLES Schema Rowset	250
Standard Mapping	250
Rowset Extensions	251

About Schema Rowsets

What Are Schema Rowsets?

Schema rowsets provide metadata about a data source. The definition of data source depends on the provider being used:

- For the local provider, a data source is a single directory on disk, which corresponds to a single library.
- For the SAS/SHARE provider, a data source consists of all library names that are known to a SAS/SHARE server.
- For the IOM provider, a data source consists of all library names that are known to a SAS Workspace Server.
- For the OLAP provider, a data source consists of all cubes that are known to a SAS OLAP Server.
- For the Base SAS provider, a data source consists of all library names that are known to a local installation of Base SAS.

Supported Schema Rowsets

The SAS providers support the following three schema rowsets:

Table A5.1 Schema Rowsets Supported by All Providers

Schema Rowset	Description
TABLES	Provides information about which tables are available to the data source.
COLUMNS	Provides information about all of the columns within those tables.
PROVIDER_TYPES	Indicates which data types the provider uses to expose SAS data.

In addition, the OLAP Provider also supports the following OLE DB for OLAP schema rowsets:

Table A5.2 Schema Rowsets Supported by the OLAP Provider

Schema Rowset	Description
CATALOGS	Provides information about the catalogs that are available to the data source.
CUBES	Provides information about the cubes that are available to the data source.

Schema Rowset	Description
DIMENSIONS	Provides information about the dimensions that are available in a specified cube.
HIERARCHIES	Provides information about the hierarchies that are available in a specified dimension.
LEVELS	Provides information about the levels that are available in a specified dimension.
MEASURES	Provides information about the measures that are available to the data source.
PROPERTIES	Provides information about the properties that are available in each level and cell.
MEMBERS	Provides information about the members that are available to the data source.
FUNCTIONS	Provides information about the functions that are available to the data source.
SETS	Provides information about any currently defined sets.

For more information about each schema rowset, see the following topics:

- [“CATALOGS Schema Rowset”](#) on page 232
- [“COLUMNS Schema Rowset”](#) on page 232
- [“CUBES Schema Rowset”](#) on page 236
- [“DIMENSIONS Schema Rowset”](#) on page 237
- [“FUNCTIONS Schema Rowset”](#) on page 238
- [“HIERARCHIES Schema Rowset”](#) on page 239
- [“LEVELS Schema Rowset”](#) on page 240
- [“MEASURES Schema Rowset”](#) on page 242
- [“MEMBERS Schema Rowset ”](#) on page 244
- [“PROPERTIES Schema Rowset”](#) on page 246
- [“PROVIDER_TYPES Schema Rowset”](#) on page 247
- [“SETS Schema Rowset”](#) on page 249
- [“TABLES Schema Rowset”](#) on page 250

Note: The tables in each schema rowset topic list type indicators. For definitions of the type indicators, see the *OLE DB Programmer’s Reference and Data Access SDK*.

How Are Schema Rowsets Obtained?

You can obtain schema rowsets in the following ways:

- If you are using ADO, you can pass one of the elements of the ADO SchemaEnum enumeration to the OpenSchema method of an open Connection object.
- If you are using OLE DB, you can pass the GUID that represents the desired schema to the IDBSchemaRowset::GetRowset method on an open Session object.

How to Restrict the Rows That Are Returned

You can use DBSchemaRowset::GetRowset to restrict the rows that are returned in a schema rowset. A restriction is a value for a specific column. When a restriction is passed to IDBSchemaRowset::GetRowset, all rows that are returned in the schema rowset must match the restriction value on the specified column. Restriction values do not support pattern matching or wildcards. For example, the restriction value **D_F** matches **D_F** but not **DEF**.

TIP To determine which columns support restrictions, write an OLE DB consumer that calls IDBSchemaRowset::GetSchemas.

CATALOGS Schema Rowset

Standard Mapping

The CATALOGS schema rowset returns metadata that indicates which catalogs can be accessed by the provider. Each row in the CATALOGS schema rowset corresponds to one catalog available in the provider.

The following table describes the columns that are available in the CATALOGS schema rowset.

Table A5.3 Standard Mapping

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
CATALOG_NAME	DBTYPE_WSTR	Name of the catalog. Cannot be NULL.	Yes
DESCRIPTION	DBTYPE_WSTR	Description of the catalog.	No

Note: For more information about restrictions, see [“How to Restrict the Rows That Are Returned”](#) on page 232.

COLUMNS Schema Rowset

Standard Mapping

The COLUMNS schema rowset returns metadata about all of the columns within the tables that are available to the current data source. Each row in the COLUMNS schema

rowset corresponds to one column in a table. In the case of a SAS data set, a column is a variable.

The following table lists the 11 columns (of the 28 columns in the OLE DB specification) that contain data.

Table A5.4 Standard Mapping

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
TABLE_NAME	DBTYPE_WSTR	Table name. This value is either <i>libname.memname</i> (SAS/SHARE, IOM, and Base SAS providers) or the data set name (local provider).	Yes
COLUMN_NAME	DBTYPE_WSTR	Column name that corresponds to the SAS variable (column) name.	No
ORDINAL_POSITION	DBTYPE_UI4	Column ordinal that corresponds to the SAS variable (column) number. Column numbers begin at 1.	No
COLUMN_HASDEFAULT	DBTYPE_BOOL	VARIANT_TRUE if the column has a default value. VARIANT_FALSE if there is no default value or if the existence of a default value is unknown.	No
COLUMN_FLAGS	DBTYPE_BOOL	A bitmask that describes the column. (See “Possible COLUMN_FLAGS Values” on page 235.)	No
IS_NULLABLE	DBTYPE_UI4	VARIANT_TRUE if the column can be set to NULL. VARIANT_FALSE if the column cannot be set to NULL (which corresponds to a SAS missing value).	No
DATA_TYPE	DBTYPE_UI2	Column data type.	No
CHARACTER_MAXIMUM_LENGTH_TYPE	DBTYPE_UI4	The maximum possible length in characters of the column value.	No
CHARACTER_OCTET_LENGTH	DBTYPE_UI4	The maximum length in bytes of the column value.	No

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
NUMERIC_PRECISION	DBTYPE_UI2	The maximum precision of the column.	No
DESCRIPTION	DBTYPE_WSTR	The SAS variable (column) label.	No

Note: For more information about restrictions, see [“How to Restrict the Rows That Are Returned” on page 232](#).

Rowset Extensions

All the SAS providers extend the COLUMNS schema rowset to include metadata that is specific to SAS data sets. These custom schema rowset columns include the same information that is returned by the ISASColumnsInfo interface. However, there are some differences in how the information is made available. Specifically, the ISASColumnsInfo interface returns SAS metadata one data set at a time and is limited to OLE DB consumers. By contrast, the COLUMNS schema rowset extensions include all columns (variables) that are in all tables (data sets) in a data source. They are also available to both OLE DB and ADO consumers.

Each custom column maps to a member of the SASCOLUMNINFO structure that is returned by ISASColumnsInfo::GetColumnInfo. A ninth member in that structure, pwszColDesc, does not have a corresponding custom column because it returns a variable label that is included in the OLE DB specification's standard DESCRIPTION column.

The following table lists the columns that are added to the COLUMNS schema rowset. The columns are returned following the standard columns in the OLE DB specification and in the order in which they appear in the following table.

Table A5.5 Rowset Extensions

Column Name	Type Indicator	Mapped Member	Description
FORMAT_NAME	DBTYPE_WSTR	pwszFmtName	Stored format name. If no format is associated with this column, this member is NULL.
FORMAT_LENGTH	DBTYPE_I2	iFmtLength	Stored width of the formatted data.
FORMAT_DECIMAL	DBTYPE_I2	iFmtDecimal	Stored decimal width of the formatted data.
INFORMAT_NAME	DBTYPE_WSTR	pwszIFmtName	Stored informat name.
INFORMAT_LENGTH	DBTYPE_I2	iIFmtLength	Stored width to use when applying the default informat.
INFORMAT_DECIMAL	DBTYPE_I2	iIFmtDecimal	Stored decimal width to use when applying the default informat.

Column Name	Type Indicator	Mapped Member	Description
SORT_ORDER	DBTYPE_I2	iSortInfo	A signed short value that indicates the column's position in any applied sorting hierarchy. This member is not valid for all providers, and it is applicable only if the data set is the result of a SORT procedure. (See “What SORT_ORDER Means” on page 235.)
DBTYPE_INDEXED	DBTYPE_BOOL	fIndexed	VARIANT_TRUE when the variable is indexed; otherwise, it is VARIANT_FALSE.

Additional Details

Possible COLUMN_FLAGS Values

The COLUMN_FLAGS column is a bitmask that describes the column. The DBCOLUMNFLAGS enumerated type lists all of the possible bits that could be set in the bitmask.

DBCOLUMNFLAGS_ISFIXEDLENGTH	Set only if all data in the column has the same length. This flag is set for columns that contain numeric data. If DBPROP_SAS_BLANKPADDING is True, then this flag is set for columns containing character data. Otherwise, this flag is not set.
DBCOLUMNFLAGS_MAYBENULL	Set if the column can contain NULL values (which correspond to missing values). This flag is used by OLE DB consumers that read data to determine whether they might encounter a missing value.
DBCOLUMNFLAGS_WRITE	Set only if IRowsetChange::SetData can be called for the column.

Note: For more information about DBCOLUMNFLAGS, see the discussion of IColumnsInfo::GetColumnInfo in the OLE DB specification.

What SORT_ORDER Means

If a data set has been produced by a SORT procedure (PROC SORT), then the SORT_ORDER custom column contains a signed short value. This value indicates the column's position in the applied sorting hierarchy specified in the BY statement. Positive values indicate ascending sort order, and negative values indicate descending sort order. The absolute value of the signed short value describes the position of the variable in the sorting hierarchy. Zero (0) indicates that the column does not participate in sorting.

The meaning of the value in the SORT_ORDER custom column depends on the data provider and the consumer:

- The local provider supports this column for both OLE DB and ADO consumers.

- The SAS/SHARE and Base SAS providers support this column only for OLE DB consumers. When used with an ADO consumer, this column contains zero when it participates in sorting.
- The IOM provider does not support this column at all, so the column always contains zero when it participates in sorting.

CUBES Schema Rowset

Standard Mapping

The CUBES schema rowset returns metadata about the SAS cubes that can be accessed by the SAS OLAP Provider. Each row in the CUBES schema rowset corresponds to one cube that is available in the provider.

The following table describes the columns that are available in the CUBES schema rowset.

Table A5.6 Standard Mapping

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
CATALOG_NAME	DBTYPE_WSTR	Name of the catalog that contains this cube.	Yes
SCHEMA_NAME	DBTYPE_WSTR	Name of the schema that contains this cube.	Yes
CUBE_NAME	DBTYPE_WSTR	Name of this cube.	Yes
CUBE_GUID	DBTYPE_GUID	GUID of this cube.	No
CREATED_ON	DBTYPE_DBTIMESTAMP	Date and time that this cube was created.	No
LAST_SCHEMA_UPDATE	DBTYPE_DBTYPESTAMP	Date and time that this cube's schema was last updated.	No
SCHEMA_UPDATED_BY	DBTYPE_WSTR	User ID of the person who last updated this cube's schema.	No
LAST_DATA_UPDATE	DBTYPE_DBTIMESTAMP	Date and time that this cube was last updated.	No
DATA_UPDATED_BY	DBTYPE_WSTR	User ID of the person who last updated this cube.	No
DESCRIPTION	DBTYPE_WSTR	Description of this cube.	No

Note: For more information about restrictions, see [“How to Restrict the Rows That Are Returned”](#) on page 232.

DIMENSIONS Schema Rowset

Standard Mapping

The DIMENSIONS schema rowset returns metadata about the dimensions that are available on each cube that can be accessed by the provider. Each row in the DIMENSIONS schema rowset corresponds to a dimension that is available on a particular cube.

The following table describes the columns that are available in the DIMENSIONS schema rowset.

Table A5.7 Standard Mapping

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
CATALOG_NAME	DBTYPE_WSTR	Name of the catalog that contains this dimension.	Yes
SCHEMA_NAME	DBTYPE_WSTR	Name of the schema that contains this dimension.	Yes
CUBE_NAME	DBTYPE_WSTR	Name of the cube that contains this dimension.	Yes
DIMENSION_NAME	DBTYPE_WSTR	Name of this dimension.	Yes
DIMENSION_UNIQUE_NAME	DBTYPE_WSTR	Unique name of this dimension.	Yes
DIMENSION_GUID	DBTYPE_GUID	GUID of this dimension.	No
DIMENSION_CAPTION	DBTYPE_WSTR	Caption of this dimension. Same as DIMENSION_NAME if no caption exists for this dimension.	No
DIMENSION_ORDINAL	DBTYPE_UI4	Ordinal number of this dimension. This ordinal is relative only to the other dimensions in the same cube.	No
DIMENSION_TYPE	DBTYPE_I2	Type of this dimension. There are four types <ul style="list-style-type: none"> • MD_DIMTYPE_TIME, which is a time dimension. • MD_DIMTYPE_MEASURE, which is a measures dimension. • MD_DIMTYPE_OTHER, which is neither a time nor a measures dimension. • MD_DIMTYPE_UNKNOWN, which is an unknown type. 	No

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
DIMENSION_CARDINALITY	DBTYPE_UI4	Number of members contained by this dimension.	No
DEFAULT_HIERARCHY	DBTYPE_WSTR	Default hierarchy of this dimension.	No
DESCRIPTION	DBTYPE_WSTR	Description of this dimension.	No

Note: For more information about restrictions, see [“How to Restrict the Rows That Are Returned”](#) on page 232.

FUNCTIONS Schema Rowset

Standard Mapping

The FUNCTIONS schema rowset returns metadata that indicates which functions are available on the cubes that can be accessed by the provider. Each row in the FUNCTIONS schema rowset corresponds to one function that is available in the provider. The default sort order is ORIGIN, INTERFACE_NAME, and FUNCTION_NAME.

The following table describes the columns that are available in the FUNCTIONS schema rowset.

Table A5.8 Standard Mapping

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
FUNCTION_NAME	DBTYPE_WSTR	Name of the function.	Yes
DESCRIPTION	DBTYPE_WSTR	Description of the function.	No
PARAMETER_LIST	DBTYPE_WSTR	Comma-delimited list of parameters.	No
RETURN_TYPE	DBTYPE_14	Variable type of the return data type of the function.	No
ORIGIN	DBTYPE_14	One of the following: <ul style="list-style-type: none"> • built-in MDX and SAS functions • user-defined or external functions 	Yes
INTERFACE_NAME	DBTYPE_WSTR	Name of the interface for the user-defined function, as well as the group name for the MDX functions.	Yes
LIBRARY_NAME	DBTYPE_WSTR	Name of the type library for user-defined functions.	Yes

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
DLL_NAME	DBTYPE_WSTR	Name of the .dll or .exe in which this function is implemented.	No
HELP_FILE	DBTYPE_WSTR	Name of the help file that documents this function.	No
HELP_CONTEXT	DBTYPE_14	Help context ID for this function in the help file.	No
OBJECT	DBTYPE_WSTR	Object to which this function applies such as a dimension or level.	No
PARAM_LIST	DBTYPE_WSTR	Comma-delimited list of parameters.	No
CAPTION	DBTYPE_WSTR	Name of the function.	Yes

Note: For more information about restrictions, see [“How to Restrict the Rows That Are Returned”](#) on page 232.

HIERARCHIES Schema Rowset

Standard Mapping

The HIERARCHIES schema rowset returns metadata that indicates which hierarchies are available on the dimensions that can be accessed by the provider. Each row in the HIERARCHIES schema rowset corresponds to one hierarchy that is available in the provider.

The following table describes the columns available in the HIERARCHIES schema rowset.

Table A5.9 Standard Mapping

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
CATALOG_NAME	DBTYPE_WSTR	Name of the catalog that contains this hierarchy.	Yes
SCHEMA_NAME	DBTYPE_WSTR	Name of the schema that contains this hierarchy.	Yes
CUBE_NAME	DBTYPE_WSTR	Name of the cube that contains this hierarchy.	Yes
DIMENSION_UNIQUE_NAME	DBTYPE_WSTR	Unique name of the dimension that contains this hierarchy.	Yes

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
HIERARCHY_NAME	DBTYPE_WSTR	Name of this hierarchy. NULL if the parent dimension has no hierarchies.	Yes
HIERARCHY_UNIQUE_NAME	DBTYPE_WSTR	Unique name of this hierarchy. Same as the name of the parent dimension if that dimension has zero or one hierarchy.	Yes
HIERARCHY_GUID	DBTYPE_GUID	GUID of this hierarchy.	No
HIERARCHY_CAPTION	DBTYPE_WSTR	The caption of this hierarchy. Same as HIERARCHY_NAME if no caption exists for this hierarchy.	No
DIMENSION_TYPE	DBTYPE_I2	The type of the parent dimension of this hierarchy. One of four types: <ul style="list-style-type: none"> • MD_DIMTYPE_TIME, which is a time dimension. • MD_DIMTYPE_MEASURE, which is a measures dimension. • MD_DIMTYPE_OTHER, which is neither a time nor a measures dimension. • MD_DIMTYPE_UNKNOWN, which is an unknown type. 	No
HIERARCHY_CARDINALITY	DBTYPE_UI4	The number of members contained by this hierarchy.	No
DEFAULT_MEMBER	DBTYPE_WSTR	Default member of this hierarchy.	No
[ALL]_MEMBER	DBTYPE_WSTR	Member at the highest level of rollup in this hierarchy.	No
DESCRIPTION	DBTYPE_WSTR	Description of this hierarchy.	No

Note: For more information about restrictions, see [“How to Restrict the Rows That Are Returned”](#) on page 232.

LEVELS Schema Rowset

Standard Mapping

The LEVELS schema rowset returns metadata about the levels that are available on the dimensions that can be accessed by the provider. Each row in the LEVELS schema rowset corresponds to one level that is available in the provider.

The following table describes the columns that are available in the LEVELS schema rowset.

Table A5.10 Standard Mapping

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
CATALOG_NAME	DBTYPE_WSTR	Name of the catalog that contains this level.	Yes
SCHEMA_NAME	DBTYPE_WSTR	Name of the schema that contains this level.	Yes
CUBE_NAME	DBTYPE_WSTR	Name of the cube that contains this level.	Yes
DIMENSION_UNIQUE_NAME	DBTYPE_WSTR	Unique name of the dimension that contains this level.	Yes
HIERARCHY_UNIQUE_NAME	DBTYPE_WSTR	Unique name of the hierarchy that contains this level.	Yes
LEVEL_NAME	DBTYPE_WSTR	Name of this level.	Yes
LEVEL_UNIQUE_NAME	DBTYPE_WSTR	Unique name of this level.	Yes
LEVEL_GUID	DBTYPE_GUID	GUID of this level.	No
LEVEL_CAPTION	DBTYPE_WSTR	Caption of this level. Same as LEVEL_NAME if this level has no caption.	No
LEVEL_NUMBER	DBTYPE_UI4	The distance from this level to the root of the hierarchy. The LEVEL_NUMBER of the root level is 0.	No
LEVEL_CARDINALITY	DBTYPE_UI4	The number of members contained by this level.	No

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
LEVEL_TYPE	DBTYPE_I4	The type of this level. These are the types: <ul style="list-style-type: none"> • MDLEVEL_TYPE_REGULAR • MDLEVEL_TYPE_ALL • MDLEVEL_TYPE_CALCULATED • MDLEVEL_TYPE_TIME • MDLEVEL_TYPE_TIME_YEARS • MDLEVEL_TYPE_TIME_HALF_YEARS • MDLEVEL_TYPE_QUARTERS • MDLEVEL_TYPE_MONTHS • MDLEVEL_TYPE_WEEKS • MDLEVEL_TYPE_DAYS • MDLEVEL_TYPE_HOURS • MDLEVEL_TYPE_MINUTES • MDLEVEL_TYPE_SECONDS • MDLEVEL_TYPE_UNDEFINED • MDLEVEL_TYPE_UNKNOWN 	No
DESCRIPTION	DBTYPE_WSTR	Description of this level.	No

Note: For more information about restrictions, see [“How to Restrict the Rows That Are Returned”](#) on page 232.

MEASURES Schema Rowset

Standard Mapping

The MEASURES schema rowset returns metadata about the measures that can be accessed by the provider. Each row in the MEASURES schema rowset corresponds to one measure that is available in the provider.

The following table describes the columns that are available in the MEASURES schema rowset.

Table A5.11 Standard Mapping

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
CATALOG_NAME	DBTYPE_WSTR	Name of the catalog that contains this measure.	Yes

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
SCHEMA_NAME	DBTYPE_WSTR	Name of the schema that contains this measure.	Yes
CUBE_NAME	DBTYPE_WSTR	Name of the cube that contains this measure.	Yes
MEASURE_NAME	DBTYPE_WSTR	Name of this measure.	Yes
MEASURE_UNIQUE_NAME	DBTYPE_WSTR	Unique name of this measure.	Yes
MEASURE_CAPTION	DBTYPE_WSTR	Caption of this measure. Same as MEASURE_NAME if this measure has no caption.	No
MEASURE_GUID	DBTYPE_GUID	GUID of this measure.	No
MEASURE_AGGREGATOR	DBTYPE_I4	Indicator of how this measure was derived. The indicators are as follows: <ul style="list-style-type: none"> MDMEASURE_AGGR_SUM MDMEASURE_AGGR_COUNT MDMEASURE_AGGR_MIN MDMEASURE_AGGR_MAX MDMEASURE_AGGR_AVG MDMEASURE_AGGR_VAR MDMEASURE_AGGR_STD MDMEASURE_AGGR_CALCULATED MDMEASURE_AGGR_UNKNOWN 	No
DATA_TYPE	DBTYPE_UI2	Data type of this measure.	No
NUMERIC_PRECISION	DBTYPE_UI2	Maximum numeric precision for numeric measures. NULL for all other measures.	No
NUMERIC_SCALE	DBTYPE_I2	Number of digits to the right of the decimal point for numeric measures. NULL for all other measures.	No
MEASURE_UNITS	DBTYPE_WSTR	Unit of this measure.	No
DESCRIPTION	DBTYPE_WSTR	Description of this measure.	No

Note: For more information about restrictions, see [“How to Restrict the Rows That Are Returned”](#) on page 232.

MEMBERS Schema Rowset

Standard Mapping

The MEMBERS schema rowset returns metadata about the members that are available on each cube that can be accessed by the provider. Each row in the MEMBERS schema rowset corresponds to one member that is available in the provider.

The following table describes the columns that are available in the MEMBERS schema rowset.

Table A5.12 Standard Mapping

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
CATALOG_NAME	DBTYPE_WSTR	Name of the catalog that contains this member.	Yes
SCHEMA_NAME	DBTYPE_WSTR	Name of the schema that contains this member.	Yes
CUBE_NAME	DBTYPE_WSTR	Name of the cube that contains this member.	Yes
DIMENSION_UNIQUE_NAME	DBTYPE_WSTR	Unique name of the dimension that contains this member.	Yes
HIERARCHY_UNIQUE_NAME	DBTYPE_WSTR	Unique name of the hierarchy that contains this member.	Yes
LEVEL_UNIQUE_NAME	DBTYPE_WSTR	Unique name of the level that contains this member.	Yes
LEVEL_NUMBER	DBTYPE_UI4	Distance from this member to the root of the hierarchy. The LEVEL_NUMBER of the root level is 0.	Yes
MEMBER_ORDINAL	DBTYPE_UI4	Ordinal number of this member.	No
MEMBER_NAME	DBTYPE_WSTR	Name of this member.	Yes
MEMBER_UNIQUE_NAME	DBTYPE_WSTR	Unique name of this member.	Yes

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
MEMBER_TYPE	DBTYPE_WSTR	Type of this member. The types are as follows: <ul style="list-style-type: none"> • MDMEMBER_TYPE_REGULAR • MDMEMBER_TYPE_ALL • MDMEMBER_TYPE_FORMULA • MDMEMBER_TYPE_MEASURE • MDMEMBER_TYPE_UNKNOWN 	Yes
MEMBER_GUID	DBTYPE_GUID	GUID of this member.	No
MEMBER_CAPTION	DBTYPE_WSTR	Caption of this member. Same as MEMBER_NAME if no caption exists for this member.	Yes
CHILDREN_CARDINALITY	DBTYPE_UI4	Number of children contained by this member.	No
PARENT_LEVEL	DBTYPE_UI4	Distance from this member's parent to the root level of the hierarchy. The LEVEL_NUMBER of the root level is 0.	No
PARENT_UNIQUE_NAME	DBTYPE_UI4	Unique name of this member's parent.	No
PARENT_COUNT	DBTYPE_UI4	Number of parents of this member.	No
DESCRIPTION	DBTYPE_WSTR	Description of this member.	No

Note: For more information about restrictions, see [“How to Restrict the Rows That Are Returned”](#) on page 232.

The Tree Operation Restriction

The MEMBERS schema rowset has a 12th restriction, called tree operation, that does not correspond to an actual column. The tree operation restriction identifies the relationship between the returned members and the member specified in the MEMBER_UNIQUE_NAME restriction. These values can be bitwise-combined by using OR to form a bitmask that represents all returned rows. The following table displays the tree operation values and the relationships that they identify.

Table A5.13 Relationship of Returned Members to the Specified MEMBER_UNIQUE_NAME

Tree Operation (Returned Members)	Relationship to Specified MEMBER_UNIQUE_NAME
MDTREEOP_ANCESTORS	Ancestors of MEMBER_UNIQUE_NAME
MDTREEOP_CHILDREN	Immediate children of MEMBER_UNIQUE_NAME
MDTREEOP_SIBLINGS	Members on the same level as MEMBER_UNIQUE_NAME

Tree Operation (Returned Members)	Relationship to Specified MEMBER_UNIQUE_NAME
MDTREEOP_PARENT	Immediate parent of MEMBER_UNIQUE_NAME
MDTREEOP_SELF	MEMBER_UNIQUE_NAME itself
MDTREEOP_DESCENDANTS	All of the descendants of MEMBER_UNIQUE_NAME

PROPERTIES Schema Rowset

Standard Mapping

The PROPERTIES schema rowset returns metadata about the properties that are available on each level and the member that can be accessed by the provider. Each row in the PROPERTIES schema rowset corresponds to one property that is available in the provider.

The following table describes the columns that are available in the PROPERTIES schema rowset.

Table A5.14 Standard Mapping

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
PROPERTY_TYPE	DBTYPE_I2	Type of this property. Either MDPROP_MEMBER or MDPROP_CELL.	Yes
CATALOG_NAME	DBTYPE_WSTR	Name of the catalog that contains this property.	Yes
SCHEMA_NAME	DBTYPE_WSTR	Name of the schema that contains this property.	Yes
CUBE_NAME	DBTYPE_WSTR	Name of the cube that contains this property.	Yes
DIMENSION_UNIQUE_NAME	DBTYPE_WSTR	Unique name of the dimension that contains this property.	Yes
HIERARCHY_UNIQUE_NAME	DBTYPE_WSTR	Unique name of the hierarchy that contains this property.	Yes
LEVEL_UNIQUE_NAME	DBTYPE_WSTR	Unique name of the level that contains this property.	Yes
MEMBER_UNIQUE_NAME	DBTYPE_WSTR	Unique name of the member that contains this property.	Yes
PROPERTY_NAME	DBTYPE_WSTR	Name of this property.	Yes

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
PROPERTY_CAPTION	DBTYPE_WSTR	Caption of this property.	No
DATA_TYPE	DBTYPE_UI2	Data type of this property.	No
CHARACTER_MAXIMUM_LENGTH	DBTYPE_UI4	Maximum length of this property for character, binary, and bit properties if one of these properties is defined. The value is 0 for character, binary, and bit properties that do not have a defined maximum length. The value is NULL for all other properties.	No
CHARACTER_OCTET_LENGTH	DBTYPE_UI4	Maximum length in octets of this property for character, binary, and bit properties if one of these properties is defined. The value is 0 for character, binary, and bit properties that do not have a defined maximum length. The value is NULL for all other properties.	No
NUMERIC_PRECISION	DBTYPE_UI2	Maximum numeric precision for numeric properties. NULL for all other properties.	No
NUMERIC_SCALE	DBTYPE_I2	Number of digits to the right of the decimal point for numeric properties. NULL for all other properties.	No
DESCRIPTION	DBTYPE_WSTR	Description of this property.	No

Note: For more information about restrictions, see [“How to Restrict the Rows That Are Returned”](#) on page 232.

PROVIDER_TYPES Schema Rowset

Standard Mapping

The PROVIDER_TYPES schema rowset returns metadata that indicates which data types are returned by the provider. Each row in the PROVIDER_TYPES schema rowset corresponds to one data type.

The SAS providers return two data types (numeric data and character data), which means that the PROVIDER_TYPES schema rowset contains only two rows. Numeric data is returned as a DBTYPE_R8 type indicator. Character data is returned as a DBTYPE_STR type indicator.

The following table lists the 16 columns (of the 20 columns in the OLE DB specification) that contain data.

Table A5.15 Standard Mapping

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
TYPE_NAME	DBTYPE_WSTR	Data type name. Value is either num or char.	No
DATA_TYPE	DBTYPE_UI2	The DBTYPE value that corresponds to the current data type (numeric or character).	Yes
COLUMN_SIZE	DBTYPE_UI4	For numeric data, the maximum precision of the data type. For character data, the maximum length of the column in characters.	No
LITERAL_PREFIX	DBTYPE_WSTR	Character used to prefix a literal of the data type in a text command.	No
LITERAL_SUFFIX	DBTYPE_WSTR	Character used to create a suffix for a literal of the data type in a text command.	No
IS_NULLABLE	DBTYPE_BOOL	VARIANT_TRUE if the data type can be set to NULL; VARIANT_FALSE if the data type cannot be set to NULL.	No
CASE_SENSITIVE	DBTYPE_BOOL	VARIANT_TRUE if the data type is character data and also case-sensitive. VARIANT_FALSE if the data type is not character data or is not case sensitive.	No
SEARCHABLE	DBTYPE_UI4	Indicates whether the data type is searchable. Value is DB_SEARCHABLE if the provider supports ICommandText and the data type can be used with any relative operator in a WHERE clause. Value is NULL if the provider does not support ICommandText.	No
UNSIGNED_ATTRIBUTE	DBTYPE_BOOL	Indicates whether the data type is signed or not. Value is VARIANT_TRUE when the data type is signed; value is VARIANT_FALSE when the data type is not signed. Value is NULL if the signed or unsigned status is not applicable to the data type.	No
FIXED_PREC_SCALE	DBTYPE_BOOL	VARIANT_TRUE if precision and scale are fixed for the data type; VARIANT_FALSE if there is not a fixed precision and scale for the data type.	No

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
AUTO_UNIQUE_VALUE	DBTYPE_BOOL	VARIANT_TRUE if values of the data type can automatically increment; otherwise, VARIANT_FALSE.	No
GUID	DBTYPE_GUID	The global unique identifier (GUID) for the data type.	No
TYPELIB	DBTYPE_WSTR	The type library for the data type.	No
IS_LONG	DBTYPE_BOOL	VARIANT_TRUE if this type is a BLOB that contains very long data; otherwise, VARIANT_FALSE.	No
BEST_MATCH	DBTYPE_BOOL	VARIANT_TRUE if this data type is the best match between the OLE DB data type indicated by the value in the DATA_TYPE column and all types in the data source. VARIANT_FALSE if this data type is not the best match.	No
IS_FIXEDLENGTH	DBTYPE_BOOL	VARIANT_TRUE if all data of this type has the same length; otherwise, VARIANT_FALSE.	No

Note: For more information about restrictions, see [“How to Restrict the Rows That Are Returned”](#) on page 232.

SETS Schema Rowset

Standard Mapping

The SETS rowset returns metadata about the sets that are available on each schema or catalog that can be accessed by the provider. Each row in the SETS schema rowset corresponds to one set that is available in the provider. The default sort order is CATALOG_NAME, SCHEMA_NAME, CUBE_NAME, SET_NAME.

The following table describes the columns that are available in the SETS schema rowset.

Table A5.16 Standard Mapping

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
CATALOG_NAME	DBTYPE_WSTR	Name of the catalog to which the set belongs.	Yes
SCHEMA_NAME	DBTYPE_WSTR	Name of the schema that contains this set.	Yes
CUBE_NAME	DBTYPE_WSTR	Name of the cube that contains this set.	Yes

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
SET_NAME	DBTYPE_WSTR	Name of the set.	Yes
SCOPE	DBTYPE_14	Scope of the set. One of the following types: <ul style="list-style-type: none"> • MDSET_SCOPE_GLOBAL • MDSET_SCOPE_SESSION 	Yes
DESCRIPTION	DBTYPE_WSTR	Description of this set.	No

Note: For more information about restrictions, see [“How to Restrict the Rows That Are Returned”](#) on page 232.

TABLES Schema Rowset

Standard Mapping

The TABLES schema rowset returns metadata about all of the tables in the current data source. Each row in the TABLES schema rowset corresponds to an individual table. Each SAS data set is considered to be a table.

The following table lists the five of the nine columns in the OLE DB specification that contain data.

Table A5.17 Standard Mapping

Column Name	Type Indicator	Mapped Value	Restrictions Supported?
TABLE_NAME	DBTYPE_WSTR	Table name. The value is either <i>libname.memname</i> (SAS/SHARE, IOM, and Base SAS providers) or <i>memname</i> (local provider).	Yes
TABLE_TYPE	DBTYPE_WSTR	Table type. The value is either TABLE (all providers) or VIEW (SAS/SHARE, IOM, and Base SAS providers).	No
DESCRIPTION	DBTYPE_WSTR	Table description. In the case of a SAS data set, this value is the data set's label.	No
DATE_CREATED	DBTYPE_DATE	Table creation date.	No
DATE_MODIFIED	DBTYPE_DATE	Most recent modification date.	No

Note: For more information about restrictions, see [“How to Restrict the Rows That Are Returned”](#) on page 232.

Rowset Extensions

All of the SAS providers extend the TABLES schema rowset to include metadata that is specific to SAS data sets. These custom schema rowset columns include the same information that is returned by the ISASDataSetInfo interface. However, there are some differences in how the information is made available. Specifically, the ISASDataSetInfo interface returns SAS metadata one data set at a time and is limited to OLE DB consumers. By contrast, the TABLES schema rowset extensions include all tables (data sets) in a data source at one time and are available to both OLE DB and ADO consumers.

Each custom column maps to a member of the SASDATASETINFO structure that is returned by ISASDataSetInfo::GetDataSetInfo. A fifth member in that structure, pwszLabel, does not have a corresponding custom column because it returns a data set label that is included in the OLE DB specification's standard DESCRIPTION column.

The following table lists the columns that are added to the TABLES schema rowset. The columns are returned following the standard columns in the OLE DB specification, and in the order in which they appear in the following table.

Table A5.18 Rowset Extensions

Column Name	Type Indicator	Mapped Member	Description
LOGICAL_RECORD_COUNT	DBTYPE_I4	lLogicalRecordCount	The logical number of records in the SAS data set, which is the number of records you would encounter if you positioned at the beginning of the table and read sequentially until you encountered "end of file." If the number of records is unknown, -1 is returned.
PHYSICAL_RECORD_COUNT	DBTYPE_I4	lPhysicalRecordCount	The physical number of records that are in the SAS data set, which indicates the number of slots physically allocated for records in the data set. This value can be greater than the number of logical records. If the number of records is unknown, -1 is returned.
RECORD_LENGTH	DBTYPE_I4	lRecordLength	The number of bytes that are required to physically store a row of data in the SAS data set. The record length multiplied by the physical record count indicates the magnitude of the data set that is on disk but not the specific size.
COMPRESSED	DBTYPE_WSTR	pwszCompressionRoutine	The name of the compression algorithm used. If no compression is set, then the value is "NO."

Column Name	Type Indicator	Mapped Member	Description
INDEXED	DBTYPE_BOOL	fIsIndexed	This column is set to VARIANT_TRUE when an index exists on the SAS data set; otherwise, it is VARIANT_FALSE.
SAS_DATASET_TYPE	DBTYPE_WSTR	wszDataSetType	The SAS data set type.
SAS_DATASET_LABEL	DBTYPE_WSTR	wszDataSetLabel	A user-defined attribute of up to 200 characters that is used for documenting the SAS data set.
SAS_DATASET_ENCODING	DBTYPE_I4	lEncoding	The encoding value from the SAS data set.
SAS_DATASET_WINDOWS_CODEPAGE	DBTYPE_I4	lWinCP	The Windows code page value for the SAS data set.

Appendix 6

OLE DB: Format Processing

About Format and Informat Processing with OLE DB	253
What Are Formats and Informats?	253
Supported SAS Formats	254
Determining Persisted (Default) Formatting Information	254
How to Specify Format Processing When Binding Columns	255
Using Formats for Input Operations	256
How to Keep Default Formats	256
Sample Code for Using Default Formats	256
Overriding Formats for Input Operations	258
How to Override Default Formats	258
Sample Code for Overriding Default Formats	258
Processing Informats for Output Operations	261
How to Simultaneously Bind Columns to Formats and Informats	261

About Format and Informat Processing with OLE DB

What Are Formats and Informats?

A SAS format is a pattern or set of instructions that SAS uses to determine how the values of a variable (or column) should be written or displayed. A SAS informat is a pattern or set of instructions that SAS uses to determine how data values in an input file should be interpreted. SAS provides a set of standard formats and informats and also enables you to define your own.

OLE DB applications use customizations to the OLE DB interfaces in order to access SAS formatting features. The following topics explain how to use the customizations in your application:

- [“How to Specify Format Processing When Binding Columns” on page 255](#)
- [“Using Formats for Input Operations” on page 256](#)
- [“Overriding Formats for Input Operations” on page 258](#)
- [“Processing Informats for Output Operations” on page 261](#)

Note: ADO applications support SAS formatting features through the use of the "SAS Formats" and "SAS Informats" properties. For more information, see [“Using SAS Formats When You Read Data”](#) on page 92 and [“Using SAS Informats When You Write Data”](#) on page 94.

Supported SAS Formats

The level of support for SAS formats depends on the provider:

- The IOM provider supports both user-written formats and SAS formats that are defined within the context of the currently active IOM workspace.
- The local provider, OLAP provider, and SAS/SHARE provider do not support user-written formats; however, they do support most SAS formats.

Note: The OLAP provider requires that the formats are persisted with the cube on the OLAP server. For more information, see [“Determining Persisted \(Default\) Formatting Information”](#) on page 254.

Here is the list of supported SAS formats:

\$ASCII	DOWNAME	NUMX	S370FZDT
\$BINARY	DTDATE	OCTAL	S370FZDU
\$CHAR	DTMONYY	PD	TIME
\$EBCDIC	DTYEAR	PERCENT	TIMEAMPM
\$HEX	DTYYQC	PIB	TOD
\$OCTAL	E	PIBR	WEEKDATE
\$QUOTE	FLOAT	PK	WEEKDATX
\$REVERJ	FRACT	PVALUE	WEEKDAY
\$REVERS	HEX	QTR	WORDDATE
\$UPCASE	HHMM	QTRR	WORDDATX
\$XPORCH	HOUR	RB	WORDF
BEST	IB	ROMAN	WORDS
BINARY	IEEE	SSN	XYYMMDD
COMMA	JULIAN	S370FF	YEAR
COMMAX	MINGUO	S370FIB	YEN
D	MMDDYY	S370FIBU	YYMM
DATEAMPM	MMSS	S370FPD	YYMMDD
DATETIME	MMYY	S370FPDU	YYMMN
DATE	MONNAME	S370FPIB	YYMON
DAY	MONTH	S370FRB	YYQ
DDMMYY	MONYY	S370FZD	YYQR
DOLLAR	NEGPAREN	S370FZDL	Z
DOLLARX	NENGO	S370FZDS	ZD

For more information about SAS formats, see the *SAS Language Reference: Dictionary*.

Determining Persisted (Default) Formatting Information

In a SAS data set, a variable (column) can have persisted (default) formatting information. There are two methods that you can use to determine what, if any, default

formatting information exists in a data set. Use the method that is most appropriate for your application.

- Along with some other column metadata that is specific to SAS, the `ISASColumnsInfo` customized rowset interface returns the names, lengths, and decimal widths of formats and informats that are stored in the data set. In order to use the interface, the data set must be open. For more information about how to use the interface, see [“ISASColumnsInfo Custom Interface” on page 218](#).

Note: A format or informat name that is returned by `ISASColumnsInfo::GetColumnsInfo` might not be supported by a particular provider or server. By default, you will receive an error if you attempt to apply format or informat services with a format that is not supported. To change this default behavior, set the `DBPROP_SAS_FMTERR` property to `VARIANT_FALSE`.

- The `COLUMNS` schema rowset also returns format and informat metadata. Consider using this method if your application needs to determine metadata for columns that exist in a table that is not open. For more information, see [“COLUMNS Schema Rowset” on page 232](#).

How to Specify Format Processing When Binding Columns

When you write code to bind columns during read, write, and update operations, you can also provide formatting instructions. For example, you can specify that you want to keep or override default formatting for specific columns.

To provide formatting instructions, you use the `SASFORMAT` structure (defined by `SASExtensions.idl`). The following code shows the `SASFORMAT` structure:

```
typedef enum
{
    SASFMT_FORMAT,
    SASFMT_INFORMAT
} SASFMTENUM;

typedef struct tagSASFORMAT
{
    SASFMTENUM wType;
    LPOLESTR pwszName;
    SHORT iLength;
    SHORT iDecimals;
} SASFORMAT;
```

The elements (members) of the `SASFORMAT` structure are described in the following table.

Table A6.1 Elements (Members) of the `SASFORMAT` Structure

Element	Description
<code>wType</code>	Indicates whether a particular instance of a <code>SASFORMAT</code> structure applies to a column's format (<code>SASFMT_FORMAT</code>) or informat (<code>SASFMT_INFORMAT</code>).

Element	Description
pwszName	Pointer to the name (as a string) of the format or informat that should be used to override the column's default format.
iLength	Specifies the width to apply to the format or informat. If this value is nonzero, it will override the column's default format.
iDecimals	Specifies the number of significant decimal places that should be used when applying numeric formats or informats. If this value is nonzero, it will override the associated column's default format.

To perform the binding operation, you call the `IAccessor::CreateAccessor` method. For information about using the `IAccessor::CreateAccessor` method on rowsets returned by `IOpenRowset::OpenRowset` and `ICommand::Execute`, see the following topics:

- [“Using Formats for Input Operations” on page 256](#)
- [“Processing Informats for Output Operations” on page 261](#)
- [“Overriding Formats for Input Operations” on page 258](#)

Using Formats for Input Operations

How to Keep Default Formats

In a SAS data set, a variable (column) can have persisted (default) formatting information, which you have the option to keep with the returned data.

Note: If there is no persisted formatting information, then the provider applies a format that best fits the variable type. For more information about persisted formats, see [“Determining Persisted \(Default\) Formatting Information” on page 254](#).

Sample Code for Using Default Formats

The sample data set has three columns: SALEDATE, QUANTITY, and PRICE. Each column has a persisted format. The following sample code shows you how to specify the default formats when you write code to bind each column.

```
#include "SASExtensions.h" 1

SASFORMAT formatOverride; 2

memset( &formatOverride, 0, sizeof( formatOverride ) ); 3

formatOverride.wType = SASFMT_FORMAT; 4

DBBINDEXT bindingExtension; 5
bindingExtension.pExtension = (BYTE *) &formatOverride
```

```

bindingExtension.ulExtension = 1;

DBBINDING rgBindings[3]; 6
struct ACOLUMN
{
    DWORDstatus;
    BSTRvalue;
};

memset( rgBindings, 0, sizeof( rgBindings ) ); 7

for( ULONG i =0; i < sizeof(rgBindings); i++) 8
{

    rgBindings[i].iOrdinal = i+1; 9
    rgBindings[i].dwPart = DBPART_VALUE | DBPART_STATUS;
    rgBindings[i].obValue = (i * sizeof(struct ACOLUMN)) + offsetof(struct ACOLUMN, value);
    rgBindings[i].obStatus = (i * sizeof(struct ACOLUMN)) + offsetof(struct ACOLUMN, status);

    rgBindings[i].dwMemOwner = DBMEMOWNER_CLIENTOWNED; 10

    rgBindings[i].wType = 11

    DBTYPE_BSTR; 12
    rgBindings[i].pBindExt = &bindingExtension
}

```

- 1 enables the application to access SAS extensions to OLE DB. Access is specifically required in order to define the SASFORMAT structure.
- 2 creates a SASFORMAT structure in order to request the default format. The same instance of this structure is reused for all three columns.
- 3 All members are initialized to NULL or zero.
- 4 sets the override type to "formatting".
- 5 creates a DBBINDEXT structure to link the SASFORMAT structure to the DBBINDING structures. The DBBINDEXT structure explains how to organize the data that is returned by the provider. The DBBINDEXT structure is reused for all three columns.
- 6 ties the bindingExtension to each column that will be formatted. There is a DBBINDING structure for each bound column.
- 7 As a matter of good programming, the structure is initialized. Columns are bound in order. However, because it cannot be formatted, the self bookmark is skipped.
- 8 a loop that binds all of the columns as DBTYPE_BSTR and applies the default format as specified by the SASFORMAT structure.

- 9 returns the status because applying a format to a particular data item could fail. The column length is not returned because the data is being returned as BSTR, which has a fixed length.
- 10 frees each data item as the client finishes with it.
- 11 accepts the default character type. If you use a different character type, then specify a maximum column width. Base the value on the width of the format that is being applied.
- 12 supplies the formatting information. You can reuse the same DBBINDEXT and SASFORMAT structure multiple times to use the default format.

With such an array of DBBINDING structures, a call can be made to IAccessor::CreateAccessor to create an HACCESSOR that will obtain columns using SAS formats.

See Also

- [“Overriding Formats for Input Operations” on page 258](#)
- [“How to Specify Format Processing When Binding Columns” on page 255](#)

Overriding Formats for Input Operations

How to Override Default Formats

In a SAS data set, a variable (column) can have persisted formatting information, which you have the option to override. To override the persisted (default) format values, you specify a format length (width) and number of decimals (for numeric formats).

Sample Code for Overriding Default Formats

The example data set has three columns: SALEDATE, QUANTITY, and PRICE. Each column has a persisted (default) format. You want to override the default formatting for the SALEDATE and PRICE columns. Specifically, you want to format the SALEDATE column using MMDDYY8. formatting, and you want to format the PRICE column using DOLLAR8.2.

You also want to perform some calculations on the PRICE column. To perform the calculations, you must write additional code that binds PRICE as a numeric value. So, even though the data set has only three columns, this example requires the following four bindings:

- SALEDATE bound as DBTYPE_BSTR with MMDDYY8. formatting
- QUANTITY bound as a DBTYPE_R8
- PRICE bound as DBTYPE_BSTR with DOLLAR8.2 formatting
- PRICE bound as DBTYPE_R8

```
DBBINDING rgBindings[4]; 1
DBBINDEXT rgBindingExtensions[2];
SASFORMAT rgFormatOverrides[2];
```

```

ULONG dwOffset; 2

typedef struct 3
{
    DWORD status;
    BYTE value[1];
} COLUMNDEF;

memset( rgBindings, 0, sizeof( rgBindings ) ); 4
memset( rgBindingExtensions, 0, sizeof( rgBindingExtensions ) );
memset( rgFormatOverrides, 0, sizeof( rgFormatOverrides ) );

dwOffset = 0; 5

rgBindings[0].iOrdinal = 1; 6
rgBindings[0].dwPart = DBPART_VALUE | DBPART_STATUS;
rgBindings[0].obValue =dwOffset + offsetof( COLUMNDEF, value );
rgBindings[0].obStatus =dwOffset + offsetof( COLUMNDEF, status );
rgBindings[0].dwMemOwner = DBMEMOWNER_CLIENLOWNER;
rgBindings[0].wType = DBTYPE_BSTR;

rgFormatOverrides[0].wType = SASFMT_FORMAT; 7
rgFormatOverrides[0].pwszName = L"MMYYDD";
rgFormatOverrides[0].iLength = 8;

rgBindingExtensions[0].pExtension = (BYTE *) &(rgFormatOverrides[0]); 8
rgBindingExtensions[0].ulExtension = 1;
rgBindings[0].pBindExt = &(rgBindingExtensions[0]);

dwOffset += sizeof(BSTR) + offsetof( COLUMNDATA, value ); 9

rgBindings[1].iOrdinal = 2; 10
rgBindings[1].dwPart = DBPART_VALUE | DBPART_STATUS;
rgBindings[1].obValue =dwOffset + offsetof( COLUMNDEF, value );
rgBindings[1].obStatus =dwOffset + offsetof( COLUMNDEF, status );
rgBindings[1].wType = DBTYPE_R8;
dwOffset += sizeof(double) + offsetof( COLUMNDATA, value );

rgBindings[2].iOrdinal = 3; 11
rgBindings[2].dwPart = DBPART_VALUE | DBPART_STATUS;
rgBindings[2].obValue =dwOffset + offsetof( COLUMNDEF, value );
rgBindings[2].obStatus =dwOffset + offsetof( COLUMNDEF, status );
rgBindings[2].dwMemOwner = DBMEMOWNER_CLIENLOWNER;
rgBindings[2].wType = DBTYPE_BSTR;

rgFormatOverrides[1].wType = SASFMT_FORMAT;
rgFormatOverrides[1].pwszName = L"DOLLAR";

```

```

rgFormatOverrides[1].iLength = 8;
rgFormatOverrides[1].iDecimals = 2;

rgBindingExtensions[1].pExtension = (BYTE *) &(rgFormatOverrides[0]);
rgBindingExtensions[1].ulExtension = 1;
rgBindings[2].pBindExt = &(rgBindingExtensions[0]);
dwOffset += sizeof(BSTR) + offsetof( COLUMNDATA, value );

rgBindings[3].iOrdinal = 3; 12
rgBindings[3].dwPart = DBPART_VALUE | DBPART_STATUS;
rgBindings[3].obValue =dwOffset + offsetof( COLUMNDEF, value );
rgBindings[3].obStatus =dwOffset + offsetof( COLUMNDEF, status );
rgBindings[3].wType = DBTYPE_R8;

dwOffset += sizeof(double) + offsetof( COLUMNDATA, value ); 13

```

- 1 creates an array of DBBINDING structures. There are four bindings, two of which have formatting overrides.
- 2 tracks the offset of a bound data point into an I/O buffer.
- 3 After each binding is created, lays out a status and value part.
- 4 As a matter of good programming, all structures are initialized.
- 5 Parts are laid into the buffer at offset 0. When the bindings are created, dwOffset is the total number of bytes that are needed to hold one row in memory.
- 6 binds SALEDATE as DBTYPE_BSTR with the MMYDD8. format applied. The basic OLE DB binding information comes first.
- 7 prepares the format override for SALEDATE.
- 8 links the SASFORMAT structure to the DBBINDING structure.
- 9 sets the offset to increase beyond the data points for SALEDATE.
- 10 binds QUANTITY as DBTYPE_R8. Only basic OLE DB binding information is needed.
- 11 binds PRICE as DBTYPE_BSTR with the DOLLAR8.2 format applied. This action is similar to the binding for SALEDATE.
- 12 binds PRICE as DBTYPE_R8.
- 13 After this code is executed, dwOffset is the number of bytes that the application needs in order to store a row of data that is retrieved by using this binding information.

See Also

- [“Using Formats for Input Operations” on page 256](#)
- [“How to Specify Format Processing When Binding Columns” on page 255](#)

Processing Informats for Output Operations

You can use the SASFORMAT structure in order to specify to use an informat with a column. You can also choose to override default informatting information. The instructions for using and overriding informats are similar to the instructions for using and overriding formats. Here is the procedure:

1. Set the wType member of the SASFORMAT structure to SASFMT_INFORMAT, so that the instance of the structure applies to an informat rather than a format.
2. Set pwszName to an informat name to override the default, or set it to zero (0) or NULL to use the default.
3. Set iLength to a width to override the default, or set it to zero (0) or NULL to use the default.
4. Set iDecimals to a decimal value to override the default, or set it to zero (0) or NULL to use the default.
5. Set the wType member of your DBBINDING structure to one of these OLE DB character types: DBTYPE_STR, DBTYPE_WSTR, or DBTYPE_BSTR. (This action is the same as the format setting.)

See Also

- [“Using Formats for Input Operations” on page 256](#)
- [“Overriding Formats for Input Operations” on page 258](#)

How to Simultaneously Bind Columns to Formats and Informats

A column can be bound with both a format and an informat simultaneously. To perform this task, set the ulExtension member of the DBBINDEXT to 2, and set the pExtension to the address of an array of two SASFORMAT structures. One of the structures in this array should define the format to apply and the other structure should define the informat.

CAUTION:

The SAS providers do not validate that formats and informats are paired logically on the same column binding. The providers also do not prevent you from binding the same column more than once per accessor, nor do they prevent you from using that accessor on output operations. If you use an accessor in this way and if the accessor you use is not consistent with the operation being performed, you might receive data integrity errors.

Appendix 7

OLE DB: Column Mapping and Binding

About the Mapping and Binding Process	263
Returning Column Metadata	263
Mapping to SAS Constructs	264
Binding to Rowset Columns	265

About the Mapping and Binding Process

Before an OLE DB consumer can read, update, or insert a row into a table, the following two steps must be taken:

1. The SAS providers must use OLE DB constructs to give to the application the information about the columns in the data store. See [“Returning Column Metadata” on page 263](#).
2. The client application (the consumer) must give to the data provider the following specific information about the columns:
 - which columns will be manipulated
 - what type they will be accessed as
 - where they will be stored in the client's memory

The process of communicating this information to the provider is called column binding. Column binding is done by using the OLE DB accessors. See [“Binding to Rowset Columns” on page 265](#).

Returning Column Metadata

You can retrieve column metadata by using either the COLUMNS schema rowset or the IColumnsInfo::GetColumnsInfo() method. Choose the method that best fits your current needs:

- You can use the COLUMNS schema rowset to return metadata about all of the columns in every table in the data source component. You can also filter the returned data by a subset of the tables. Filtering enables you to view many details about the data store without explicitly opening all of the tables in the data source component.

- The `IColumnsInfo::GetColumnsInfo()` method is available when you have a rowset that is opened on a specific table. For the opened table, the `GetColumnsInfo` method returns the same kind of information that is found in the `COLUMNS` schema rowset. The information is returned in an array of `DBCOLUMNINFO` structures that can be accessed more efficiently than a schema rowset.

Mapping to SAS Constructs

In OLE DB, a rowset is a type of cursor over a table that consists of rows and columns. The intersection of each row and column identifies a cell of data. Every cell in a specific column is of the same data type. The OLE DB concept of a table corresponds directly to the SAS data set.

A data set consists of variables and observations. A variable defines a cell of data in an observation. Variables are like columns, and observations are like rows. In fact, these OLE DB terms and SAS terms can be used interchangeably.

The following table compares the members of the `DBCOLUMNINFO` structure with the `COLUMNS` schema rowset columns and describes how they all correspond to SAS constructs:

Table A7.1 *DBCOLUMNINFO Structure Members Compared with COLUMNS Schema Rowset Columns and the Corresponding SAS Constructs*

DBCOLUMNINFO Member	COLUMNS Schema Rowset Column	SAS Data Model	Comments
<code>pwszName</code> <code>columnid</code>	<code>COLUMN_NAME</code>	variable name	
<code>pTypeInfo</code>	not available	not applicable	Reserved by Microsoft for future use
<code>iOrdinal</code>	<code>ORDINAL_POSITION</code>	variable number	
<code>dwFlags</code>	not applicable	not applicable	
<code>ulColumnSize</code>	<code>CHARACTER_MAXIMUM_LENGTH</code> and <code>CHARACTER_OCTET_LENGTH</code>	variable length	Defines the maximum length of this column/variable
<code>wType</code>	<code>DATA_TYPE</code>	variable type	When accessing Unicode data sets, character variables are mapped to <code>DBTYPE_WSTR</code> . When accessing data sets that are not Unicode, character variables are mapped to <code>DBTYPE_STR</code> . Numeric variables are mapped to <code>DBTYPE_R8</code> for data sets in all encodings.
<code>pPrecision</code>	<code>NUMERIC_PRECISION</code>	not applicable	

DBCOLUMNINFO Member	COLUMNS Schema Rowset Column	SAS Data Model	Comments
bScale	not applicable	not applicable	
not available	DESCRIPTION	variable label	
not available	IS_NULLABLE	not available	Indicates that a cell in this column could be set to "missing"

Members of the DBCOLUMNINFO structure and columns in the COLUMNS schema rowset that are not represented in this table either do not map to SAS constructs and are not supported, or they do not map directly to SAS column metadata.

Binding to Rowset Columns

Accessors are managed through the rowset interface IAccessor. The client application uses the IAccessor interface to specify a set of column bindings and to associate them with an accessor handle. The handle can be passed into other rowset methods that manipulate individual rows of data. When you specify column bindings and create accessors, consider these questions:

- How will the column be identified?
- What type can the client bind it to?
- How much space will be needed to store the data in the client's buffers?

If you have a DBCOLUMNINFO structure for a column, you can easily transfer that information into a DBBINDING structure, as shown in the following table:

Table A7.2 DBCOLUMNINFO Members Mapped to DBBINDING Members

DBCOLUMNINFO Members	DBBINDING Members
iOrdinal	iOrdinal
ulColumnSize	cbMaxLen
wType	wType
bPrecision	bPrecision
bScale	bScale

The value of iOrdinal must be the same in both the DBBINDING and DBCOLUMNINFO structures. However, the other members in a DBBINDING structure do not have to match the corresponding members that are returned in a DBCOLUMNINFO structure. For example, the wType value in the DBBINDING structure can have any meaningful DBTYPE because the providers support converting data between types as defined by the OLE DB specification. In addition, the values of the DBBINDING members cbMaxLen, bPrecision, and bScale are more closely related

to the `wType` member of the `DBBINDING` structure than are the corresponding members of the `DBCOLUMNINFO` structure.

If the column metadata is known at the time that you are creating the client application, then you do not need to query the data provider. Instead, you can build static `DBBINDING` structures, which can increase run-time performance. To build static `DBBINDING` structures, you must know how to map data set variable attributes to `DBBINDING` structure members. The following table explains the mapping:

Table A7.3 *DBBINDING Structure Members and Corresponding Variable Attributes*

DBBINDING Structure Member	Corresponding SAS Variable Attribute
<code>iOrdinal</code>	Set this member to the variable number that you want to bind to. The member should be a value from 1 to N, where N is the number of variables that are defined in the data set. (The value 0 is reserved for the self-bookmark column.)
<code>obValue</code> <code>obLength</code> <code>obStatus</code>	These three members determine the offset of the column value, its length, and its status of coercion in the buffer that you pass to the provider when you read and update. If the corresponding bit is not set in the <code>dwPart</code> member, these members are ignored.
<code>pTypeInfo</code>	Ignored. Reserved for future use.
<code>pObject</code>	Unsupported. OLE objects cannot be embedded in SAS data sets.
<code>pBindExt</code>	Used for formatting data.
<code>dwPart</code>	This bitmask defines which <code>obValue</code> , <code>obLength</code> , and <code>obStatus</code> members are meaningful. See the OLE DB specification for more information about this field.
<code>dwMemOwner</code>	This member does not map to any SAS variable attributes. You use as described by the OLE DB specification.
<code>eParamIO</code>	Ignored.
<code>cbMaxLen</code>	The number of bytes allocated in your buffer for the data value. This number is determined by the value of <code>wType</code> .
<code>dwFlags</code>	Set this member to 0x00. This value indicates to return regular text from the provider.
<code>wType</code>	A <code>DBTYPE</code> value that indicates the type that is used to represent the data value. This value can directly correspond to the SAS variable's type (<code>DBTYPE_WSTR</code> for character variables; <code>DBTYPE_R8</code> for numeric variables), or it can be any type that the provider can convert the data to. The OLE DB specification lists the valid conversions between <code>DBTYPE</code> s, and the SAS providers support all of these conversions. At run time you can discover which conversions are allowed by calling the <code>IConvertType::CanConvert()</code> rowset method.
<code>bPrecision</code>	The IOM provider looks only at this member when <code>wType</code> is set to <code>DBTYPE_NUMERIC</code> . In this case, this field defines the maximum precision to use when you get the data. It is ignored when you set and update data.
<code>bScale</code>	Ignored.

In the following example, you open a rowset component named pRowset on a known data set named AUTOS, which contains these variables:

- MAKE, which is a 40-byte character variable
- MODEL, a numeric variable
- MPG, a numeric variable
- COST, a numeric variable

Bind MAKE as DBTYPE_BSTR and the remaining variables as DBTYPE_STR. This binding means that the first column is a fixed length and the other three columns are varying lengths. So, cbMaxLen and obLength are not considered when you bind MAKE, but they are considered when you bind to MODEL, MPG, and COST.

Put the values of MODEL, MPG, and COST into 15-byte buffers. Buffer your consumer output to place the status and value of MAKE first, followed by the lengths, status, and values of MODEL, MPG and COST. The following code fills in an array of DBBINDING structures to match the described binding and calls the appropriate method to create the OLE DB accessor handle:

```
IAccessor * pAccessor;
DBBINDING rgBindings[4];
DBBINDSTATUS rgStatus[4];
HACCESSOR hAccessor;
ULONG ulOffset;

// Lay out each column in memory.
struct COLUMNNDATA
{
    DWORD dwLength; // length of data returned
    DWORD dwStatus; // status of column
    BYTE bData[1]; // data here and beyond
};

// Rounding amount is always a power of two.
#define ROUND_UP( Size, Amount ) (((DWORD)(Size) + ((Amount) - 1)) & ~((Amount) - 1))

// Alignment for placement of each column within memory.
// Rule of thumb is "natural" boundary, such as a 4-byte member should be
// aligned on address that is multiple of 4.
// Worst case is double or __int64 (8 bytes).
#define COLUMN_ALIGNVAL 8

memset( rgBindings, 0, sizeof( rgBindings ) ); // defensive programming

// Start laying out data at the beginning of our buffer
ulOffset = 0;
rgBindings[0].iOrdinal = 1;
rgBindings[0].dwPart = DBPART_VALUE | DBPART_STATUS;
rgBindings[0].dwFlags = 0x00;
rgBindings[0].obValue = ulOffset + offsetof( COLUMNNDATA, bData );
rgBindings[0].obStatus = ulOffset + offsetof( COLUMNNDATA, dwStatus );
rgBindings[0].wType = DBTYPE_BSTR;

// Account for space taken by actual cell value
ulOffset += sizeof( BSTR ) + offsetof( COLUMNNDATA, bData );
ulOffset = ROUND_UP( ulOffset, COLUMN_ALIGNVAL ); // round up for alignment
rgBindings[1].iOrdinal = 2;
```

```

rgBindings[1].cbMaxLen = 15;
rgBindings[1].dwPart = DBPART_VALUE | DBPART_STATUS | DBPART_LENGTH;
rgBindings[1].dwFlags = 0x00;
rgBindings[1].obValue = ulOffset + offsetof( COLUMNNDATA, bData );
rgBindings[1].obLength = ulOffset + offsetof( COLUMNNDATA, dwLength );
rgBindings[1].obStatus = ulOffset + offsetof( COLUMNNDATA, dwStatus );
rgBindings[1].wType = DBTYPE_STR;

    // Account for space taken by actual cell value
ulOffset += rgBindings[1].cbMaxLen + offsetof( COLUMNNDATA, bData );
ulOffset = ROUND_UP( ulOffset, COLUMN_ALIGNVAL ); // round up for alignment
rgBindings[2].iOrdinal = 3;
rgBindings[2].cbMaxLen = 15;
rgBindings[2].dwPart = DBPART_VALUE | DBPART_STATUS | DBPART_LENGTH;
rgBindings[2].dwFlags = 0x00;
rgBindings[2].obValue = ulOffset + offsetof( COLUMNNDATA, bData );
rgBindings[2].obLength = ulOffset + offsetof( COLUMNNDATA, dwLength );
rgBindings[2].obStatus = ulOffset + offsetof( COLUMNNDATA, dwStatus );
rgBindings[2].wType = DBTYPE_STR;

    // Account for space taken by actual cell value
ulOffset += rgBindings[2].cbMaxLen + offsetof( COLUMNNDATA, bData );
ulOffset = ROUND_UP( ulOffset, COLUMN_ALIGNVAL ); // round up for alignment
rgBindings[3].iOrdinal = 4;
rgBindings[3].cbMaxLen = 15;
rgBindings[3].dwPart = DBPART_VALUE | DBPART_STATUS | DBPART_LENGTH;
rgBindings[3].dwFlags = 0x00;
rgBindings[3].obValue = ulOffset + offsetof( COLUMNNDATA, bData );
rgBindings[3].obLength = ulOffset + offsetof( COLUMNNDATA, dwLength );
rgBindings[3].obStatus = ulOffset + offsetof( COLUMNNDATA, dwStatus );
rgBindings[3].wType = DBTYPE_STR;

    // Account for space taken by actual cell value
ulOffset += rgBindings[3].cbMaxLen + offsetof( COLUMNNDATA, bData );
ulOffset = ROUND_UP( ulOffset, COLUMN_ALIGNVAL ); // round up for alignment
pRowset->QueryInterface( IID_IAccessor, &hAccessor );
pAccessor->CreateAccessor( DBACCESSOR_ROWDATA, // we are binding to columns in a rowset
    4, // binding 4 columns
    rgBindings,
    0, // ignored for rowset bindings
    &hAccessor,
    rgStatus );

    // If the above call fails you can look at rgStatus to identify which column's binding info
    // was invalid. When the call succeeds, hAccessor is a valid accessor handle which can be
    // used as input to IRowset::GetData(), IRowsetChange::SetData() and IRowsetChange::InsertRow()
    // When you are done with the accessor handle, release it.
pAccessor->ReleaseAccessor( hAccessor, 0 );
pAccessor->Release();

```

Appendix 8

Customized User Help for the Data Link Properties Dialog Box

Data Link Properties Dialog Box (Connection Tab)	269
Data Link Properties Dialog Box (Advanced Tab)	270

Data Link Properties Dialog Box (Connection Tab)

This version of the **Connection** tab is designed specifically for you to enter connection information when you are performing one of these actions:

- using the SAS IOM provider for OLE DB to access a SAS Workspace Server
- using the SAS OLAP provider for OLE DB to access a SAS OLAP Server

Note: If you need to specify a SAS Metadata Server that has not been configured by using the SAS Integration Technologies configuration utility (ITConfig), then you must enter connection information on the **Advanced** tab in the Data Link Properties dialog box. To display that tab, click **Advanced**.

The customized **Connection** tab has these items:

Select or enter a Data Source

You have two choices, depending on where the server is located.

- If you want a new, locally instantiated server connection, type `_LOCAL_` in this field, and then click **OK** to make the connection.
- If you want a remote server connection, use the Data Source URI format to specify the location of the SAS Workspace or SAS OLAP server that you want to access. For example, if you are connecting to a SAS Workspace Server configured as COM with host authentication, you might enter this code:

```
iom-com://workspace.example.com
```

If the SAS Workspace or SAS OLAP server is defined in an LDAP (Lightweight Directory Access Protocol) server or in a SAS Metadata Server, then the URI can use the logical name for the workspace or OLAP server. (The SAS Metadata Server must have been configured by using the ITConfig utility.) Here is a sample Data Source URI that references a logical name defined in a SAS Metadata Server:

```
iom-name://Logical Workspace Server
```

For more information about the Data Source URI, see [“Using the Data Source Property to Specify All Connection-Related Properties”](#) on page 31.

Note: If you do not complete the **Data Source** field, the SAS provider assumes that you want to establish a local connection.

Note: If your local connection does not succeed, then your system administrator must check your installation and COM permissions.

Enter information to log on to the server

For secure remote connections, enter your credentials in the **User name** and **Password** fields. (For the user name, type your user ID for logging on to the remote server.)

Test Connection

Click this button in order to test the connection.

For more information about completing the **Connection** tab, contact your system administrator.

Data Link Properties Dialog Box (Advanced Tab)

This version of the **Advanced** tab is designed specifically for you to enter connection information when you are performing one of these actions:

- using the SAS IOM provider for OLE DB to access a SAS Workspace Server
- using the SAS OLAP provider for OLE DB to access a SAS OLAP Server

TIP As a best practice, use this tab only when you must specify a SAS Metadata Server that has not been configured by using the SAS Integration Technologies configuration utility (ITConfig). If the metadata server that you need to use has been configured by using ITConfig, then enter connection information on the **Connection** tab.

The customized **Advanced** tab has these items:

Enter Metadata Server information

Use default

You can use this option only if you are connecting to a SAS Workspace or SAS OLAP server that is defined in a SAS Metadata Server that has been configured by using ITConfig. If this configuration exists, then you should use the **Connection** tab to enter the connection information. To display that tab, click **Connection**.

Metadata Server Location

Use the Data Source URI format to identify the SAS Metadata Server. For example, you might enter this code:

```
iom-bridge://metadata.example.com:8561
```

User name

Type your user ID for logging on to the SAS Metadata Server.

Password

Type the password for the user ID that you entered.

Select or enter SAS Server information

Data Source

Use the Data Source URI format to specify the location of the SAS Workspace or SAS OLAP server that you want to access. The URI should reference the logical name of the workspace or OLAP server that is defined in the SAS Metadata Server that you are using. For example, you might enter this URI.:

iom-name://Logical Workspace Server

For more information about the Data Source URI, see [“Using the Data Source Property to Specify All Connection-Related Properties”](#) on page 31.

Test Connection

Click this button in order to test the connection.

For more information about completing the **Advanced** tab, contact your system administrator.

Glossary

ActiveX Data Objects

a simplified programming interface to OLE DB. Both ADO and OLE DB have been developed by Microsoft. Short form: ADO.

ActiveX Data Objects (Multidimensional)

an extension to the ADO programming interface that enables users to read multidimensional schema, to query cubes, and to retrieve the results. ADO MD accesses data through a multidimensional data provider such as the IOM Data Provider. ADO MD has been developed by Microsoft.

ActiveX Data Objects Extension for Data Definition Language and Security

an extension to the ADO programming interface that enables users to create, modify, and delete database objects such as tables. This extension, which is commonly referred to as ADOX, can also be used to manage user permissions and group permissions on database objects. Short form: ADOX.

ADO

See ActiveX Data Objects.

ADO MD

See ActiveX Data Objects (Multidimensional).

ADOX

See ActiveX Data Objects Extension for Data Definition Language and Security.

API

See application programming interface.

application programming interface

a set of software functions that facilitate communication between applications and other kinds of programs or services. Short form: API.

authentication domain

a SAS internal category that pairs logins with the servers for which they are valid. For example, an Oracle server and the SAS copies of Oracle credentials might all be classified as belonging to an OracleAuth authentication domain.

bit mask

a string of bits that has a specific pattern of binary 0s and 1s that you use to compare with other values.

cube

See OLAP cube.

data provider

software that makes data available through the OLE DB interfaces to a consumer such as an ADO application.

data set

See SAS data set.

database management system

See

DBMS

See

DNS name

a name that is meaningful to people and that corresponds to the numeric TCP/IP address of a computer on the Internet. For example, www.alphaliteairways.com might be the DNS name for an Alpalite Airways Web server whose TCP/IP address is 192.168.145.6.

fatal error

an error that causes a program to end abnormally or that prevents the program from starting.

informat

See SAS informat.

Integrated Object Model server

See IOM server.

Internet Protocol Version 4

See IPv4.

Internet Protocol Version 6

See IPv6.

IOM server

a SAS object server that is launched in order to fulfill client requests for IOM services. Short form: IOM server.

IPv4

a protocol that specifies the format for network addresses for all computers that are connected to the Internet. This protocol, which is the predecessor of Internet Protocol Version 6, uses dot-decimal notation to represent 32-bit address spaces. An example of an Internet Protocol Version 4 address is 10.23.2.3. Short form: IPv4.

IPv6

a protocol that specifies the format for network addresses for all computers that are connected to the Internet. This protocol, which is the successor of Internet Protocol Version 4, uses hexadecimal notation to represent 128-bit address spaces. The format can consist of up to eight groups of four hexadecimal characters, delimited by colons, as in FE80:0000:0000:0000:0202:B3FF:FE1E:8329. As an alternative, a group of consecutive zeros could be replaced with two colons, as in FE80::0202:B3FF:FE1E:8329. Short form: IPv6

library reference

See libref.

libref

a SAS name that is associated with the location of a SAS library. For example, in the name MYLIB.MYFILE, MYLIB is the libref, and MYFILE is a file in the SAS library.

member-level access

a type of access to a SAS library that permits only one user to use a member (such as a SAS data set) at a time.

member-level locking

a method of restricting access to a library member by providing exclusive access to the user who owns the lock.

metadata

descriptive data about data that is stored and managed in a database, in order to facilitate access to captured and archived data for further use.

Object Linking and Embedding

See OLE.

ODBC

an interface standard that provides a common application programming interface (API) for accessing data. Many software products that run in the Windows operating environment adhere to this standard so that you can access data that was created using other software products. Short form: ODBC.

OLAP cube

a logical set of data that is organized and structured in a hierarchical, multidimensional arrangement to enable quick analysis of data. A cube includes measures, and it can have numerous dimensions and levels of data.

OLE

a method of interprocess communication supported by Windows that involves a client/server architecture. OLE enables an object that was created by one application to be embedded in or linked to another application. Short form: OLE.

OLE DB

an open specification that has been developed by Microsoft for accessing both relational and nonrelational data. OLE DB interfaces can provide much of the same functionality that is provided by database management systems. OLE DB evolved from the Open Database Connectivity (ODBC) application programming interface.

Open Database Connectivity

See ODBC.

padding a value with blanks

in SAS software, a process in which the software adds blanks to the end of a character value that is shorter than the length of the variable.

persisted information

information such as formatting that remains associated with a data source element such as a column even after the program that created or accessed the data has been terminated. Persisted information can be retrieved programmatically at any time.

record-level access

a type of access to a SAS data set or other file that permits more than one user to access the SAS data set or file at a time. Only one user can use a single observation or record of the file at a time, but other users can access other observations or records in the same file.

recordset

an ADO object that contains tabular (rows and columns) data. A recordset can be returned as a result of a query or an executed command.

result set

the set of rows or records that a server or other application returns in response to a query.

SAS catalog

a SAS file that stores many different kinds of information in smaller units called catalog entries. A single SAS catalog can contain different types of catalog entries.

SAS data set

a file whose contents are in one of the native SAS file formats. There are two types of SAS data sets: SAS data files and SAS data views. SAS data files contain data values in addition to descriptor information that is associated with the data. SAS data views contain only the descriptor information plus other information that is required for retrieving data values from other SAS data sets or from files whose contents are in other software vendors' file formats.

SAS informat

a type of SAS language element that applies a pattern to or executes instructions for a data value to be read as input. Types of informats correspond to the data's type: numeric, character, date, time, or timestamp. The ability to create user-defined informats is also supported. Examples of SAS informats are BINARY and DATE. Short form: informat.

SAS IOM workspace

in the IOM object hierarchy for a SAS Workspace Server, an object that represents a single session in SAS.

SAS Metadata Server

a multi-user server that enables users to read metadata from or write metadata to one or more SAS Metadata Repositories.

SAS Open Metadata Architecture

a general-purpose metadata management facility that provides metadata services to SAS applications. The SAS Open Metadata Architecture enables applications to exchange metadata, which makes it easier for these applications to work together.

SAS session

the activity between invoking and exiting a specific SAS software product.

SAS/ACCESS software

a group of software interfaces, each of which makes data from a particular external database management system (DBMS) directly available to SAS, as well as making SAS data directly available to the DBMS.

SAS/SHARE server

the result of an execution of the SERVER procedure, which is part of SAS/SHARE software. A server runs in a separate SAS session that services users' SAS sessions by controlling and executing input and output requests to one or more SAS libraries.

Sashelp library

a SAS library supplied by SAS software that stores the text for Help windows, default function-key definitions and window definitions, and menus.

sasroot

a representation of the name for the directory or folder in which SAS is installed at a site or a computer.

SASROOT

a term that represents the name of the directory or folder in which SAS is installed at your site or on your computer.

TCP/IP

an abbreviation for a pair of networking protocols. Transmission Control Protocol (TCP) is a standard protocol for transferring information on local area networks such as Ethernets. TCP ensures that process-to-process information is delivered in the appropriate order. Internet Protocol (IP) is a protocol for managing connections between operating environments. IP routes information through the network to a particular operating environment and fragments and reassembles information in transfers.

WHERE expression

defines the criteria for selecting observations.

Work library

a temporary SAS library that is automatically defined by SAS at the beginning of each SAS session or SAS job. Unless you have specified a User library, any newly created SAS file that has a one-level name will be placed in the Work library by default and will be deleted at the end of the current SAS session or job.

workspace

See SAS IOM workspace.

Index

A

- ADO connection properties
 - See* [connection properties](#)
- ADO objects
 - supported methods and properties [143](#)

B

- Base SAS provider
 - connection properties [27](#)
 - recipes [21](#)

C

- CacheSize property [120](#)
- character data
 - padding with blanks [97](#)
- connection methods
 - overview [30](#)
- connection properties [26](#), [123](#)
 - Base SAS provider [27](#)
 - IOM provider [28](#)
 - OLAP provider [29](#)
 - SAS/SHARE provider [28](#)
- connections
 - customizing [49](#)
 - sample code [30](#)
- Connections
 - recipes [37](#)
- cursor combinations
 - client-side [141](#)
 - server-side [139](#)

D

- data access
 - recipes [67](#)
- data access permissions
 - controlling [57](#)
- Data Link Properties dialog box [50](#), [269](#)
- data sets

- creating and deleting [80](#)
- displaying metadata [73](#)
- identifying [68](#)
- missing values [101](#)
- password-protected [71](#)
- recipes [79](#)
- data sets, subsetting
 - read only, sequential access [85](#), [89](#)
- Data Source URI format [31](#)
 - constructing a dataSourceString [32](#)
 - protocols [32](#)
 - sample code [32](#)
- displaying data
 - recipes [91](#)

E

- error objects
 - using ADO to handle [129](#)
 - using OLE DB to handle [131](#)

F

- file formats
 - managing [60](#)
- file types
 - data sources [5](#)

I

- installation [12](#)
- IOM provider
 - connection properties [28](#)
 - recipes [18](#)
 - troubleshooting [134](#)
- IOM Workspace
 - reusing [63](#)
- IPv4 [33](#)
- IPv6 [33](#)

L

libref
 assigning to use with IOM provider 69
 local data
 connecting to 38
 local data (single-user server)
 connecting to 39
 local provider
 recipes 17
 troubleshooting 135
 locking records 113

M

mapping and binding 263
 Maximum Open Rows property 120
 Microsoft Data Link (.udl) file 56
 missing values
 reading from a data set 102
 recipes 101
 special numeric missing values 105
 writing to a data set 108

O

OLAP provider
 connection properties 29
 recipes 22
 OLE DB interfaces 215
 OLE DB properties 149

P

password
 password-protected data sets 71
 performance
 ADO and OLE DB properties 119
 CacheSize property 120
 Maximum Open Rows property 120
 SAS Page Size property 121
 permissions 57
 providers
 troubleshooting 133

R

reading data 92
 recipes
 Base SAS provider 21
 connection 37
 data access 67
 data sets 79
 displaying data 91
 IOM provider 18
 local provider 17
 missing values 101

OLAP provider 22
 SAS/SHARE provider 20
 updating and locking 111
 recordsets
 updating 111
 remote SAS OLAP Server
 connecting to 46
 remote SAS Workspace Server
 connecting to 42
 using objects to connect to 45
 remote SAS/SHARE server
 connecting to 41

S

SAS formats
 processing with OLE DB 253
 reading data 92
 user-defined 96
 SAS informats
 processing with OLE DB 253
 user-defined 96
 writing data 94
 SAS OLAP cubes
 reading 76
 SAS Page Size property 121
 SAS providers
 Base SAS provider 5
 common features 7
 connection properties 26
 identifying 26
 installation 12
 IOM provider 5
 local provider 5
 OLAP provider 5
 overview 4
 SAS/SHARE provider 5
 supported data sources 5
 supported file types 5
 SAS servers
 associated data providers 11
 SAS/ACCESS engines 72
 SAS/SHARE provider
 connection properties 28
 recipes 20
 SAS/SHARE server
 requesting a specific version 64
 schema rowsets 230
 system requirements 11

T

third-party data
 accessing 72
 trailing blanks 97
 troubleshooting 133

W

writing data [94](#)

