

SPSS® 14.0 Developer's Guide

SPSS is a registered trademark and the other product names are the trademarks of SPSS Inc. for its proprietary computer software. No material describing such software may be produced or distributed without the written permission of the owners of the trademark and license rights in the software and the copyrights in the published materials.

The SOFTWARE and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at 52.227-7013. Contractor/manufacturer is SPSS Inc., 233 South Wacker Drive, 11th Floor, Chicago, IL 60606-6307.

General notice: Other product names mentioned herein are used for identification purposes only and may be trademarks of their respective companies.

TableLook is a trademark of SPSS Inc.

Windows is a registered trademark of Microsoft Corporation.

DataDirect, DataDirect Connect, INTERSOLV, and SequeLink are registered trademarks of MERANT Solutions Inc.

Portions of this product were created using LEADTOOLS © 1991-2000, LEAD Technologies, Inc. ALL RIGHTS RESERVED.

LEAD, LEADTOOLS, and LEADVIEW are registered trademarks of LEAD Technologies, Inc.

Portions of this product were based on the work of the FreeType Team (<http://www.freetype.org>).

SPSS® 14.0 Developer's Guide

Copyright © 2005 by SPSS Inc.

All rights reserved.

Printed in the United States of America.

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, recording, or otherwise, without the prior written permission of the publisher.

Overview

The *SPSS Developer's Guide* provides information about the developer's tools that are included with the SPSS for Windows product. The tools can be used with the Base system and all of its optional components.

Developer's Tools

The developer's tools include:

OLE Automation. A technology standard that makes it possible for you to write programs that incorporate, or extend, the functionality of SPSS. OLE Automation allows your application to access SPSS objects and manipulate them using methods and properties. Applications can be written in a variety of programming languages, such as Visual Basic and C++.

Third-party API. An application interface that enables you to configure SPSS so that your programs can be launched from SPSS menus. Use the interface to give users access to features not provided by SPSS. The working SPSS data file can be passed to the third-party application as an optional parameter. The third-party API works by recognizing specific Windows registry keys.

Input/output DLL. A dynamic link library that enables you to write applications that read and write data in an SPSS data file format. This tool is often used to share data between third-party applications and SPSS.

Production Facility. A Visual Basic application that uses SPSS OLE Automation to run SPSS unattended and automatically produces output from regularly repeated, time-consuming analyses.

Scripting facility. A built-in, VBA-compatible Sax Basic development environment that allows developers to write and run scripts to automate SPSS tasks. You can use it to produce output (like the Production Facility) and to do regularly repeated, time-consuming editing of output. Scripting works by using OLE Automation to manipulate SPSS objects.

MACRO and MATRIX procedures. Built-in SPSS commands that allow you to write customized statistical and data manipulation procedures for use within SPSS.

Each component helps applications developers build upon SPSS for Windows, although no single application will use all of them.

Documentation Map

You can work through this guide sequentially, or you can jump to topics that interest you. Install SPSS for Windows before continuing so that you can access the online Help and run the examples.

Programmer's introduction to SPSS. You must understand how SPSS works before you can be productive using the SPSS developer's tools. Chapter 2 tells you what you need to know about SPSS before you start developing applications with it.

OLE Automation quickstart. Learning how to write applications that manipulate SPSS objects begins with an understanding of the object hierarchy. Chapter 3 describes the object hierarchy and gives you enough information to start programming with the SPSS application.

Scripting quickstart. The foundation of scripting is the OLE Automation object model. Once you're familiar with that, Chapter 4 gives you enough information to start writing and running scripts within SPSS. It also introduces autoscripts, which can be used by anyone who understands how to use SPSS.

Example scripts and applications. Chapter 5 describes the sample code shipped with SPSS and tells you where to find the example scripts and applications.

Input/output DLL. Appendix A outlines the steps for developing an application using the I/O DLL procedures and contains a reference guide for DLL procedures.

Third-party API. Appendix B is a guide to editing the Windows Registry to use the third-party API.

Tutorial. SPSS is distributed with a tutorial, which includes topics on customizing SPSS and automating output production. To access the tutorials, from the SPSS menus choose:

Help
Tutorial

Click the book icons to expand topics and select Customizing SPSS and Automated Production.

OLE Automation Help. SPSS is distributed with a complete online reference to all OLE Automation objects, methods, and properties. Access the online Help by double-clicking *spssole.hlp* in your SPSS installation directory.

Online help is accessible from most development environments. To get help for the SPSS objects:

- ▶ Include the SPSS type libraries in your development project.
- ▶ Open your development environment's object browser.
- ▶ Select the object, method, or property of interest and press F1.

The OLE Automation Help file includes the object hierarchy diagram with links to high-level objects. Select Tree View of objects from the Contents tab, and then click the object of interest. You'll get a Help topic that includes a code example, lists all methods and properties for the object, and links to their Help topics.

Production Facility documentation and online Help. For an introduction to this topic, see "Working with the SPSS Production Facility" on p. 24. For more information, see the *SPSS Base User's Guide*. You can also launch SPSS, select Topics from the Help menu, click the Index tab, and search on the word "production."

Scripting facility documentation and online tips. For an introduction to this topic, see "Working with the SPSS Scripting Facility" on p. 22. For more information, see the *SPSS Base User's Guide*.

You can access the online scripting tips while you are running SPSS. Select Scripting Tips from the Help menu of a Script window. Select Script Language for Sax Basic help.

MACRO and MATRIX command language. For more information, see Macro Facility Command Syntax and Matrix Command Syntax in the SPSS online Help, and see also the *SPSS Syntax Reference Guide*. In the *SPSS Syntax Reference Guide*, macro syntax is listed under DEFINE—!ENDDDEFINE, and there are three macro examples in the *SPSS*

Syntax Reference Guide appendix “Using the Macro Facility.” Matrix syntax is listed under MATRIX—!ENDMATRIX.

Using the SPSS Developer's Tools

The developer's tools offer a variety of integration strategies, including application integration, customization, and automation.

Integration. Integrate your application into SPSS, or integrate SPSS into your application. If you want your application to be launched by users from the SPSS menus, use the third-party API tool. Once launched, your application will coexist with SPSS until you close your application or SPSS. If you want to control SPSS from your application, use OLE Automation.

Data transfer is possible. The third-party API will send a copy of the SPSS working data file to your application, you can use the input/output DLL to have your application create data files that SPSS can process, and you can use OLE Automation to get data values from SPSS.

Customization. You can customize the SPSS user interface and output using OLE Automation and scripting. You can add your statistical and data manipulation procedures to SPSS with the MACRO and MATRIX commands.

Automation. You can automate production of SPSS output using the Production Facility and perform batch editing of output using the scripting facility.

More Information

SPSS product information. Check the SPSS Web site (<http://www.spss.com>) often for product information, updates, patches, and news about planned software releases and new products.

SPSS Script eXchange. Download useful scripts and share your scripts at the Script eXchange Web site at <http://www.spss.com/software/spss/scriptexchange>. Check the site often for new scripts.

SPSS CD-ROM. Check the SPSS for Windows CD-ROM developer's folder (`(\spss\developer)`) for readme files and other late-breaking information.

Visual Basic information. Microsoft maintains Web sites for Visual Basic (<http://msdn.microsoft.com/vbasic/default.asp>) and Visual Basic for Applications (<http://msdn.microsoft.com/vba/default.asp>). These sites include news, code examples, and links to other useful sites.

Statement of Compatibility

SPSS makes no promises, expressed or implied, to keep future versions of the SPSS for Windows or the developer's tools compatible with applications built using the current tools. Although it is in our best interest to keep your applications upgradable, we cannot predict the future direction of our product line and the effect of that direction on the applicability of the interfaces described in this guide.

Technical Support

SPSS provides technical support for the developer's tools to customers who have purchased the full version of SPSS software. (Support is *not* provided for GradPack customers.) You, the developer, are responsible for the support of all products built using these tools.

Technical support for the developer's tools includes helping a customer make the software run as documented. This includes the installation process of our software, printing, and operational problems when things don't work as documented. Technical support includes helping a customer use what is in the documentation to get a specific task done by expanding on what is documented and giving an example of how to do it. SPSS does not provide technical support for non-SPSS products used in conjunction with the SPSS developer's tools, such as Visual Basic, Sax Basic, or other applications.

Distributing Your Finished Application

You have the right to distribute the application that you have developed using the SPSS developer's tools. Users of your application must license their own copies of the SPSS for Windows software by contacting the Sales Department at SPSS Inc. For any other types of licensing or distribution arrangements, please contact SPSS Inc. directly.

In order for your application to run properly, you must have your users install the SPSS for Windows software first and then install your application.

Programmer's Introduction to SPSS for Windows

For most of your applications, you will have more success using the SPSS developer's tools if you understand how SPSS works. If you intend only to integrate your application into SPSS and have users launch it from an SPSS menu, you can skip this chapter and proceed directly to the third-party API documentation in Appendix B.

When an end user works with SPSS, he or she can choose from two alternative interfaces: the graphical user interface of menus and dialog boxes or the SPSS command syntax language.

Functionally, the graphical interface and command language nearly overlap. You can use command syntax to produce virtually any SPSS output, but you cannot use commands to modify output. Output editing must be done with the graphical interface or through OLE Automation.

You can design your program to use the graphical interface, the command language, or both. You can use OLE Automation to invoke SPSS dialog boxes or to execute command syntax. When the output is produced, you can use OLE Automation to edit the output objects. In general:

- Use SPSS command syntax or invoke SPSS dialog boxes to produce output (for example, to run data transformations, statistical procedures, and charting procedures).
- Use OLE Automation directly on output objects to format and edit your output.

This chapter introduces you to the basic features of SPSS, including:

- Working with SPSS windows and output
- Basic steps for running an analysis
- Working with the SPSS scripting facility

- Working with the SPSS Production Facility
- Working in distributed mode

The emphasis in this chapter is on programming equivalents for performing various end-user tasks. If you're already familiar with SPSS and need an introduction to SPSS OLE Automation, skip to Chapter 3.

Working with Windows and Output

SPSS for Windows provides a powerful statistical analysis and data management system. It has specialized window types that allow users to request, display, and edit the output they want. It also has specialized output items to display results.

Window Types

SPSS provides specialized window types for end users. In the SPSS object model, these windows correspond to OLE Automation document objects that have methods and properties that support most of the functionality found in the user interface. The window types, their purpose for the end user, and the corresponding OLE Automation object can be summarized as follows:

Data Editor. The working data file is displayed in the Data Editor, which is a spreadsheet-like system for entering and editing data. The corresponding OLE Automation object is `ISpssDataDoc`.

Viewer. All statistical results, tables, charts, and other output are displayed in the Viewer. The Viewer makes it easy to browse and edit your results, selectively show and hide output, and move presentation-quality output items (for example, tables and charts) between SPSS and other applications. The corresponding OLE Automation object is `ISpssOutputDoc`.

Draft Viewer. Output is displayed as simple text (instead of presentation-quality output items) in the Draft Viewer. Editing is limited. The corresponding OLE Automation object is `ISpssDraftDoc`.

Syntax. Syntax is displayed and edited in the syntax window. You can type the syntax directly or create command syntax by pasting dialog box choices into a syntax window, where your selections appear in the form of command syntax. You can save these

commands in a file for use in subsequent SPSS sessions. The corresponding OLE Automation object is `ISpssSyntaxDoc`.

Script. The script window provides a programming environment for SPSS scripts. Scripts allow you to customize and automate many tasks in SPSS. The script window doesn't have a corresponding OLE Automation object—it is a programming environment.

Output Items

When an end user interacts with SPSS dialog boxes or runs command syntax, the user produces output in a Viewer or Draft Viewer window. SPSS output consists of a number of different types of items. These items correspond to OLE Automation objects and can be manipulated with their respective methods and properties much like they can be manipulated with the user interface. The SPSS output item types are:

Pivot tables. Many SPSS statistical procedures produce pivot table output that allows users to view results in many different ways. Users can switch (pivot) row and column variables, selectively show and hide categories, and change layers in multidimensional tables. Pivot tables are produced from the Analyze menu. The corresponding OLE Automation object is `PivotTable`.

Charts and interactive graphics. SPSS produces high-resolution charts, including pie charts, bar charts, histograms, scatterplots, and 3-D graphics. Charts and interactive graphs are produced by some statistical procedures on the Analyze menu and by the Graphs menu. The corresponding OLE Automation objects are `ISpssChart` (charts) and `ISpsslGraph` (interactive graphs).

Text. A few SPSS statistical procedures produce text output. Warnings, logs, and Viewer titles are text output. All tabular output to a Draft Viewer window is text. The corresponding OLE Automation object is `ISpssRtf`.

Map. Maps can be produced by SPSS if you have the Map option installed.

The object model description in Chapter 3 shows the window and output item objects in the SPSS object hierarchy (see Figure 3-1).

Overview of Running an Analysis

The basic steps to analyze data with SPSS are:

- **Launch SPSS.**
- **Get data into SPSS.** You can open a previously saved SPSS data file, read a spreadsheet, database, or text data file, or enter data directly in the Data Editor.
- **Select a procedure.** You can select a procedure from the menus or use SPSS command syntax to transform data, calculate statistics, and to create charts, interactive graphs, and maps.
- **Select the variables and run the procedure.** The variables in the data file are displayed in a dialog box for the procedure.
- **View and manipulate the results.** Results are displayed in the Viewer. You can browse, edit, and pivot the output and save it for use at a later time.

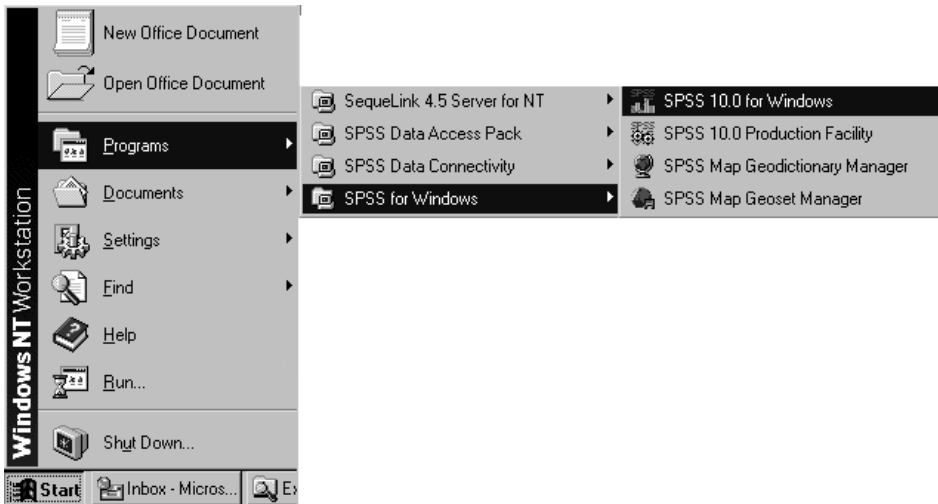
Each step and its corresponding OLE Automation commands are discussed in the following sections.

Launching SPSS

End users launch the SPSS application by selecting it from the Windows Start menu or by double-clicking the application executable, *spsswin.exe*.

Figure 2-1

Launch SPSS from Windows Start menu



Programmers launch SPSS by creating the Application object. The Application object is the container object where all other SPSS OLE Automation objects exist. Your program can run SPSS and access its properties directly. The specific techniques for

creating the application vary depending on which programming language you are using. In Visual Basic, you can use the CreateObject function:

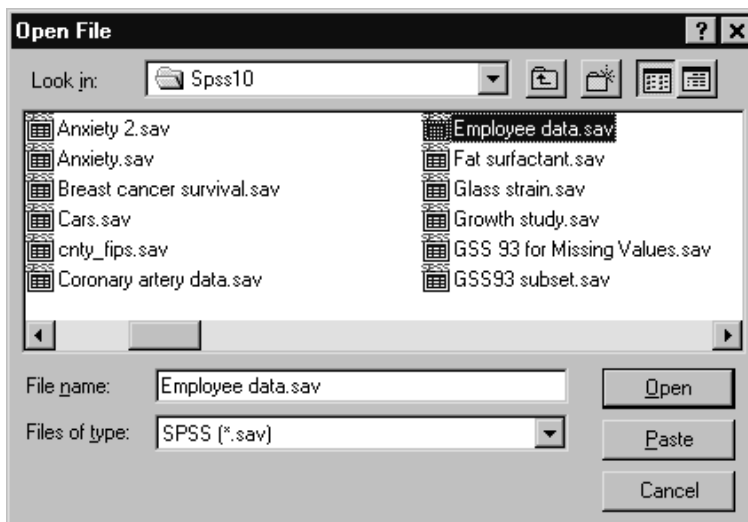
```
Dim objSpssApp As ISpssApp
Set objSpssApp = CreateObject("SPSS.Application")
```

This example accomplishes the equivalent of the end-user action shown in Figure 2-1. The first statement creates a variable named objSpssApp and assigns it to the Application object class. The second statement starts SPSS and stores a reference to it in the objSpssApp variable.

Getting Data into SPSS

SPSS can handle data in a number of spreadsheet, database, and text formats. It can also read data contained in databases using ODBC drivers. Users can open data files using the File menu or SPSS command syntax.

Figure 2-2
Open data file from menu



For programmers, the options for bringing data into SPSS are:

- For SPSS (*.sav) data files, you can use the OpenDataDoc method on the Application object. For example:

```
Set objDataDoc = objSpssApp.OpenDataDoc("d:\spss10\employee data.sav")
```

This example accomplishes the equivalent of the end-user action shown in Figure 2-2.

- For data files saved in any of the formats recognized by SPSS (including SPSS data files, all spreadsheet and database formats, and ODBC), use SPSS command syntax. (For an introduction to command syntax, see “Running Procedures with SPSS Command Syntax” on p. 14 and “Creating Command Syntax” on p. 16.)
- If necessary, you can paste data from the Windows clipboard by using the Paste method on the Data Document object. You may lose precision with this method because the values pasted are displayed values—often rounded to one or two decimal places—rather than the actual values stored in memory.

For instructions on how to use the I/O DLL to read and write SPSS data files directly, see Appendix A.

Selecting and Running a Procedure

End users can select and run procedures in SPSS either by choosing dialog boxes from the menus or by using command syntax. Both of these methods are available to programmers through OLE Automation.

Before writing an application, programmers, like end users, must decide what procedures they want to run. Use the graphical interface to accomplish the tasks that you want your application to perform. As you work with the interface, consider at each step which procedures, output items, and window types you are using. This knowledge, together with the information that follows in this chapter and in Chapter 3, will enable you to start programming with SPSS.

Figure 2-3
Selecting procedures from menus

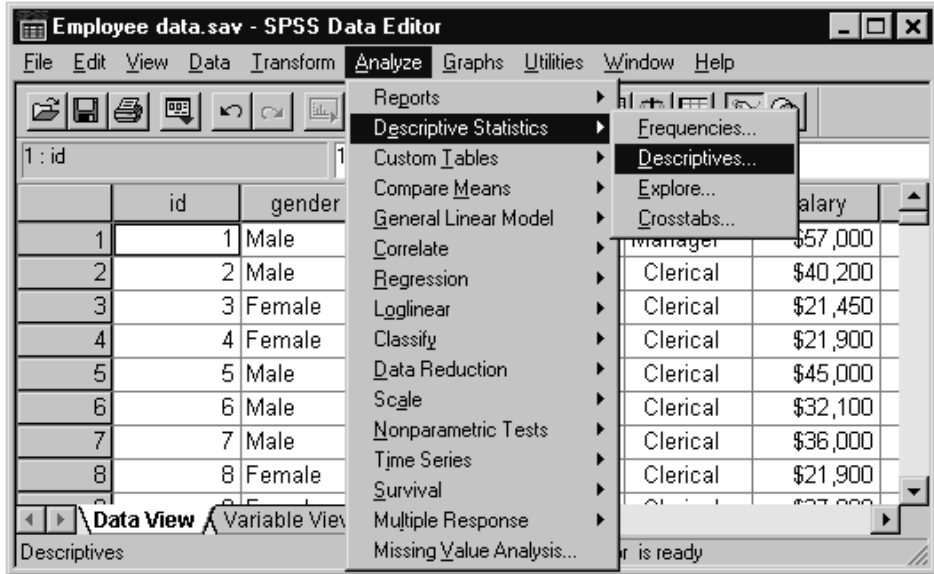
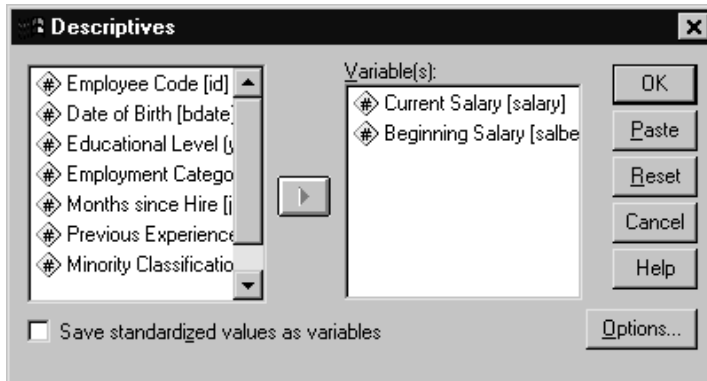


Figure 2-4
Selecting variables for analysis



Running Procedures with SPSS Dialog Boxes

You can run procedures by selecting dialog boxes from the menus. If you want your application to present dialog boxes to your end user (and allow end-user intervention), you can invoke many SPSS dialog boxes through OLE Automation by using the `InvokeDialogAndExecuteSyntax` method on the Data Document object. The `InvokeDialogAndExecuteSyntax` method is applied to the Data Document object because SPSS requires data before these dialog boxes can be invoked. This example opens the Descriptives dialog box, as shown in Figure 2-4:

```
Dim strPath as String
strPath = "analyze>descriptive statistics>descriptives"
objDataDoc.InvokeDialogAndExecuteSyntax (strPath, SpssWindowParent, True)
```

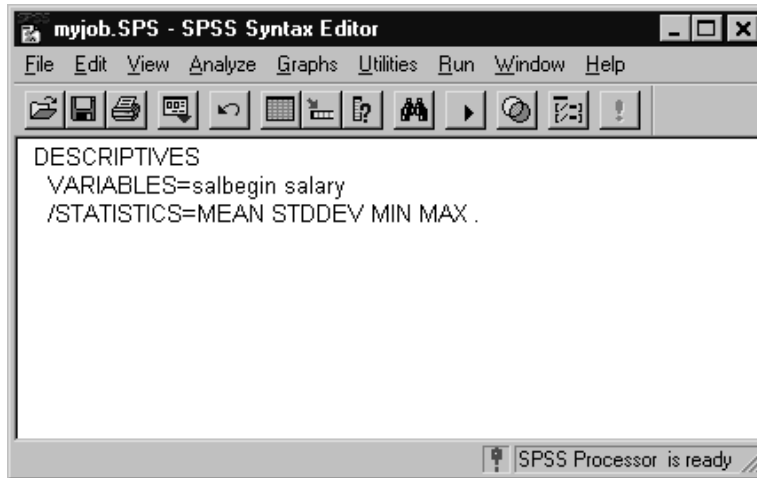
The `strPath` parameter contains the menu path to the dialog box that you want to invoke. For example, the dialog box for the Descriptives procedure can be found in the user interface on the Descriptive Statistics submenu of the Analyze menu. Once your application has opened the dialog box, the end user selects variables and options as if running SPSS in the normal manner. When the user clicks OK, SPSS executes the commands and then returns control to your program.

This method works on most dialog boxes invoked from the Analyze and Graphs menus and on some dialog boxes invoked from the Transform and Data menus. (In general, if a dialog box has a Paste button, you can use this method.)

Running Procedures with SPSS Command Syntax

The SPSS command language is an alternative to the SPSS dialog box interface. Command syntax provides complete access to all statistical and graphical procedures, data transformations, and most file operations. While most end users find the menus and dialog boxes easier to use, command syntax is a powerful tool. It provides access to additional procedures and options not available from the menus and makes it possible to save command streams in a syntax file so that they can be rerun.

Figure 2-5
Syntax window displaying command syntax



Command syntax is also a powerful tool for programmers. You can execute any valid SPSS command from within your application, allowing you full access to SPSS's capabilities. Always test your syntax by running it interactively from a syntax window before you incorporate it into your application. Use the `ExecuteCommands` method on the SPSS Application object. This example runs the syntax shown in Figure 2-5:

```
Dim strCommand as String
strCommand = strCommand + "DESCRIPTIVES"
strCommand = strCommand + " VARIABLES=salary salbegin"
strCommand = strCommand + " /STATISTICS=MEAN STDDEV MIN MAX ."
objSpssApp.ExecuteCommands strCommand, True
```

Rather than hard-coding the variables and statistics selections as shown, you can design your application to build the syntax in `strCommand` based on user input.

If you want to execute a long stream of commands or separate syntax from your code, you can save the commands in a syntax file (*.sps) and use the `ExecuteInclude` method on the SPSS Application object. `ExecuteInclude` works like the `ExecuteCommands` method except that it takes the name of a syntax file, rather than a string variable, as a parameter. This example runs the syntax file shown in Figure 2-5:

```
Dim strFilename as String
strFilename = "d:\spss10\myjob.sps"
objSpssApp.ExecuteInclude strFilename, False
```

When running syntax commands using `ExecuteCommands` methods, specify either `True` (for synchronous execution) or use a `While-Wend` loop to check if SPSS is busy. If you use the `Run` method on the `SyntaxDoc` object, you must use the `While-Wend` loop because synchronous mode is not available for this method.

Using command syntax allows your application greater flexibility and control than invoking dialog boxes from the graphical interface and makes it possible to run SPSS without the user's knowledge.

Running Procedures with the Production Facility

SPSS includes a separate application, the SPSS Production Facility, that can be used by end users and programmers to automate the running of syntax. The Production Facility is introduced in “Working with the SPSS Production Facility” on p. 24.

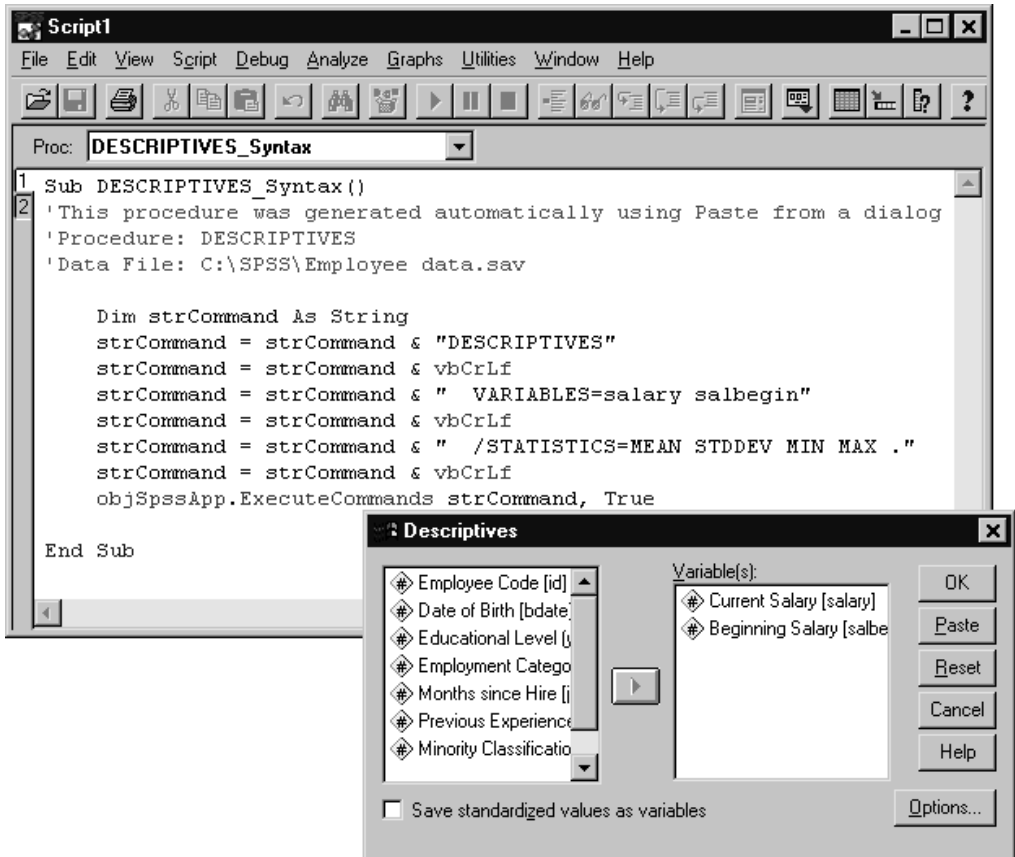
Creating Command Syntax

If you decide to use command syntax in your application, you can write it yourself by referring to the *SPSS Syntax Reference Guide*; however, writing syntax can get cumbersome. You have two other options: pasting the syntax from dialog boxes and copying syntax from SPSS log or journal files.

Pasting syntax from dialog boxes. SPSS generates command syntax from a dialog box when you click the Paste button. If you are writing code in an SPSS script window, SPSS will paste the required code and the syntax. To paste syntax and script code:

- ▶ Start SPSS and from the menus choose:
 - File
 - New
 - Script
- ▶ Or, open an existing script (*.sbs*) file.
- ▶ In the script window, choose the desired procedure from the File, Analyze, or Graphs menu.

Figure 2-6
Pasting command code into script window



- Make selections in the dialog box.
- Click Paste to paste the corresponding code statement commands into the script window.

The pasted code should run without modification in Sax Basic (the SPSS scripting language) and Visual Basic. In other languages, changes may be necessary.

Note: If you open the dialog box from any window other than the script window, the commands are pasted into a syntax window. Pasting to a script window works with all

of the dialog boxes on the Analyze and Graphs menus and many commands on the File menu. You can also paste commands from many of the dialog boxes on the Transform, and File menus, although they will be pasted into a syntax window rather than a script window. You will have to modify the code accordingly.

Copying syntax from the output log or journal file. SPSS keeps a log of your work while the program is running. Whether you run procedures by selecting them from the menus or by submitting commands, the corresponding commands are logged in the journal file. By default, the journal file is stored in your Windows temporary directory in *spss.jnl*. You can also display the commands in the Viewer.

To record commands in the journal:

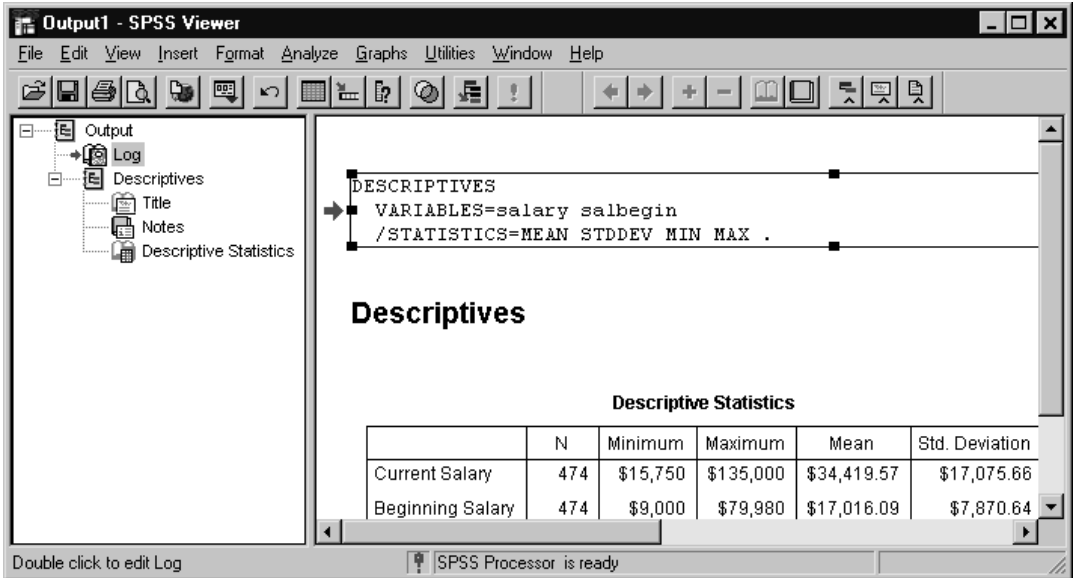
- ▶ From the menus choose:
 Edit
 Options...
- ▶ Click the General tab.
- ▶ Click Record syntax in journal.

To display commands in the Viewer:

- ▶ From the menus choose:
 Edit
 Options...
- ▶ Click the Viewer tab.
- ▶ Select Log and click Shown.

You can use the journal file and output log as sources for building syntax files or simply view them to reverse-engineer the commands needed for a particular task. For more information, see the *SPSS Base User's Guide* and the online Help.

Figure 2-7
Command syntax displayed in output log



Viewing and Manipulating Results

Whether procedures are run from the menus or with command syntax, the results are displayed in the Viewer, where end users can browse their results, selectively show and hide output, and modify their pivot tables, charts, and text output by direct manipulation. End users scroll the Viewer tree, select the item of interest, and double-click to activate it. Then they use the graphical interface to make changes to fonts, colors, and other attributes.

Figure 2-8
Viewing results in Viewer

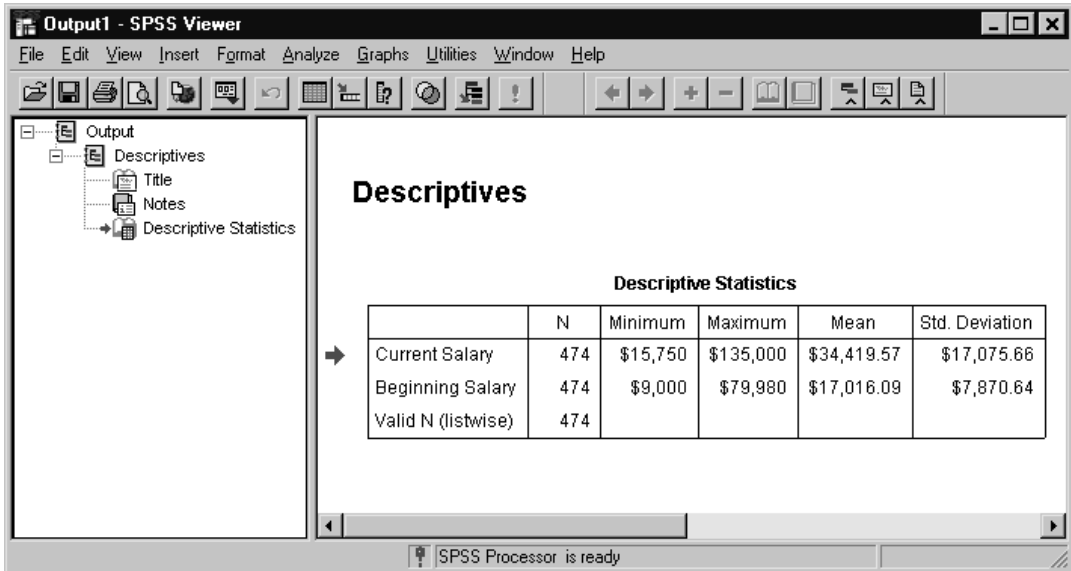
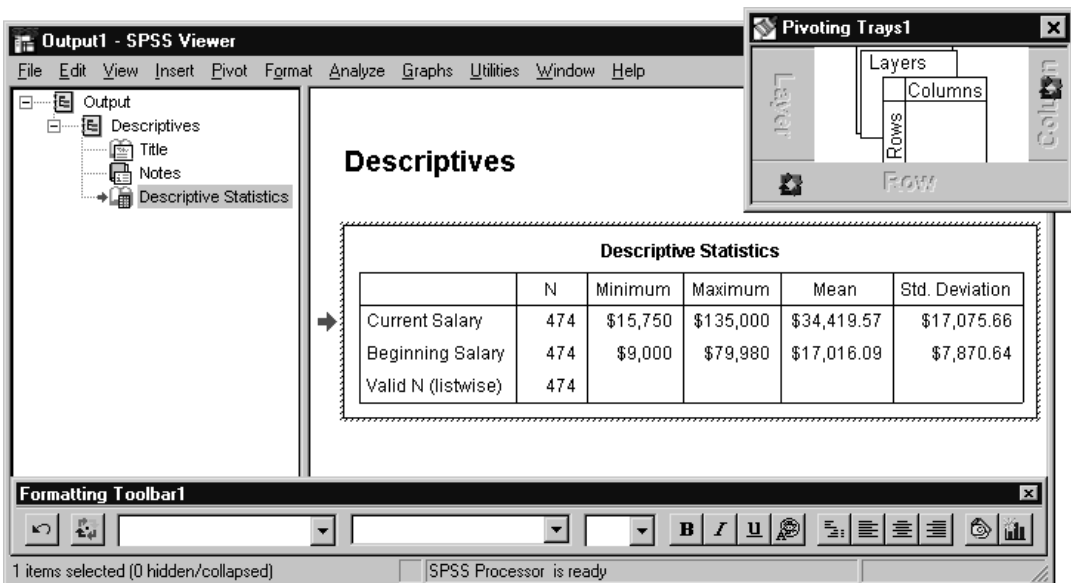


Figure 2-9
Activated pivot table object displayed in Viewer



Programmers use OLE Automation to view and manipulate output. Most of the Viewer functionality is available through OLE Automation, including the ability to select and activate an item of interest and full capabilities for editing pivot tables, interactive graphics, and text output. You modify output by getting an output document (the OLE Automation equivalent of the Viewer window), looping through the output items to find the item of interest, activating the item of interest, and then manipulating its methods and properties. For example, to rotate a pivot table's column labels:

```
' Declare variables.
Dim objSpssApp As ISpssApp
Dim objOutputDoc as ISpssOutputDoc
Dim objOutputItems as ISpssItems

' Start the SPSS application.
Set objSpssApp = CreateObject("SPSS.Application")

' Get an output document that contains a pivot table.
Set objOutputDoc = objSpssApp.GetDesignatedOutputDoc
Set objOutputItems = objOutputDoc.Items

' Get the output items collection and read the number of items.
Dim objOutputItem as ISpssItem
Dim objPivotTable as PivotTable
Dim intCount as Integer, I As Integer
intCount = objOutputItems.Count

' Loops through the items, testing each to see if it is a pivot table.
For I = 0 To intCount - 1
    Set objOutputItem = objOutputItems.GetItem (I)
    If objOutputItem.SpssType = SpssPivot Then
        ' Activate the pivot table.
        objPivotTable = objOutputItem.ActivateTable
        ' Rotate the column labels
        objPivotTable.RotateColumnLabels=True
    ' Insert additional editing here.
    Exit For
End If
Next
```

The SPSS scripting facility, which is introduced in the next section, provides another way to automate output editing.

Working with the SPSS Scripting Facility

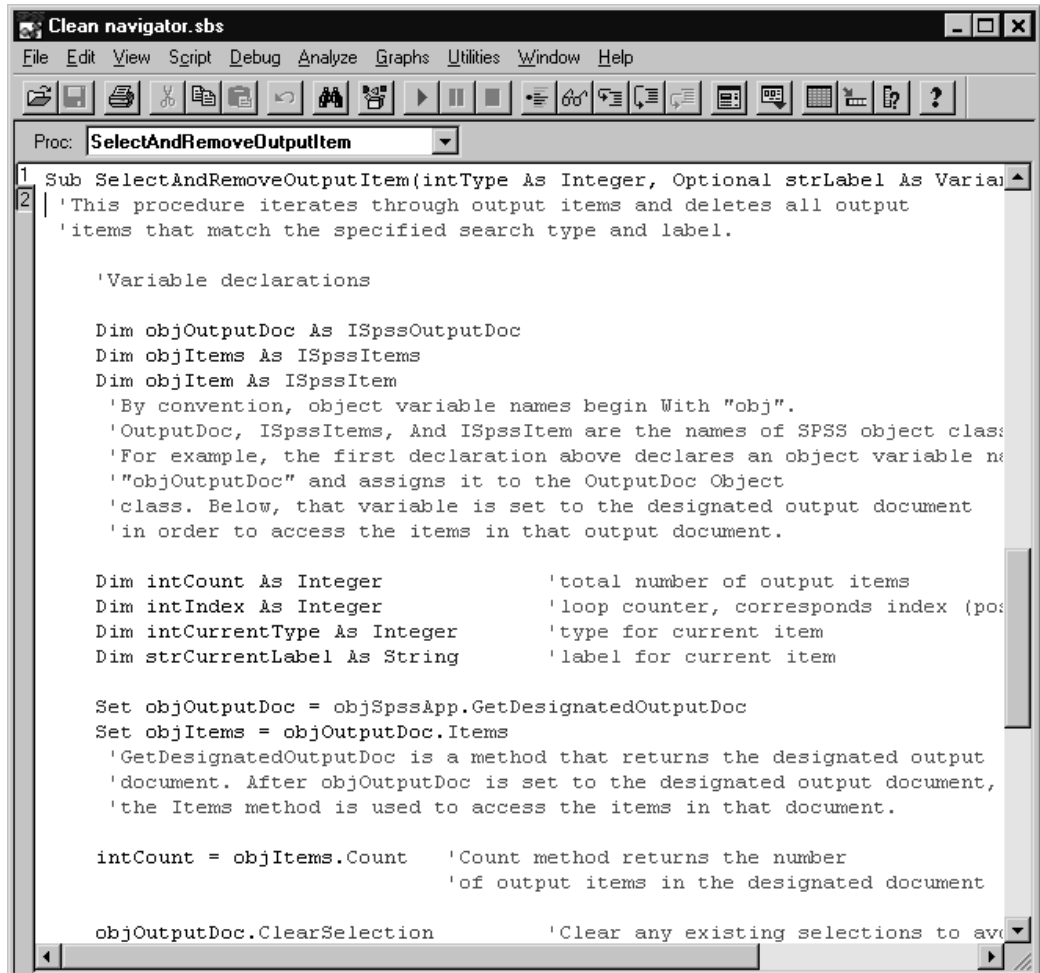
SPSS has its own internal scripting facility, shown in Figure 2-10, which uses OLE Automation to automate output production and editing from within SPSS. Scripts are written in Sax Basic, a language similar to Visual Basic. Scripting is for programmers and advanced end users who want to automate tasks from within SPSS, as opposed to programmers who want to build the SPSS functionality into their own applications. You work with scripts by opening a script window interactively in SPSS. You write, edit, and run the script from this window.

Despite some obvious differences between scripting and external programming (scripts run from within SPSS, so that the user doesn't have to worry about a compiler or starting or creating an interface to the application), both use similar techniques, often for the same goals. In principle, code developed for scripts should also run in Visual Basic, assuming you make the necessary adjustments.

SPSS includes a number of scripts that are ready to run, along with starter scripts that you can edit to create your own scripts or programs. The SPSS scripts are in the SPSS for Windows installation directory in *\Scripts* (see "Edit All Pivot Tables" on p. 88 for an example script).

For more information on the scripting facility, see Chapter 4 of this document, the "Scripting Facility" chapter in the *SPSS Base User's Guide*, and the SPSS online tutorials and Help system. To access the tutorials, select Tutorial from the Help menu. Online scripting help can be found by selecting Scripting Tips from the Help menu of a script window.

Figure 2-10
Script window



Script files and syntax files. Syntax files (*.sps) are not the same as script files (*.sbs). Syntax files have commands written in the SPSS command language that allow you to run SPSS statistical procedures and data transformations (see Figure 2-5). Scripts are written in Sax Basic and allow you to run syntax and manipulate SPSS program objects through OLE Automation.

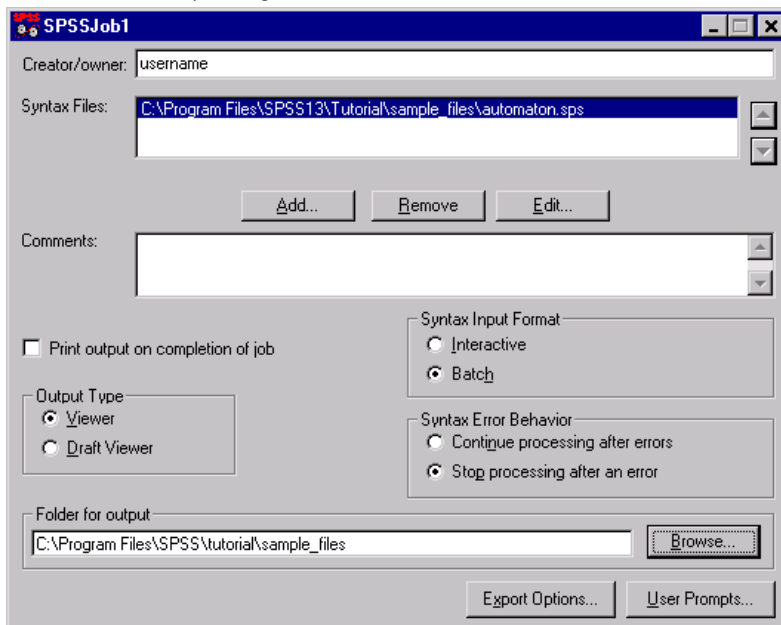
Working with the SPSS Production Facility

The SPSS Production Facility, shown in Figure 2-11, is launched from the Windows Start menu and uses syntax to automate production of output from SPSS. Programmers and advanced end users write, edit, and run the production jobs from this application. A production job is a collection of syntax files and specifications about how to run the job.

The SPSS Production Facility is a Visual Basic application that makes extensive use of SPSS OLE Automation. For more information about the Production Facility code, see “Production Facility Code” on p. 103. For more information on how the Production Facility works, see the “Production Facility” chapter in the *SPSS Base User’s Guide* and the online Help.

SPSS file types and production. A production job (*.spp) includes one or more syntax files (*.sps) and produces one output file (*.spo).

Figure 2-11
Production Facility dialog box



Working in Distributed Mode

Distributed analysis allows end users to run memory-intensive analyses on a server computer instead of a desktop computer. It requires a server version of SPSS. End users work in distributed mode by selecting Switch Server from the File menu and then logging onto a remote server.

Programmers use OLE Automation to create a client server Application object. The object is created in the same way that the SPSS Application object is created except that the Application object name is different. This example switches SPSS to distributed mode, as shown in Figure 2-12:

```
Dim objCSApp As CS_Application  
Set objCSApp = CreateObject("SPSS.CS_Application")
```

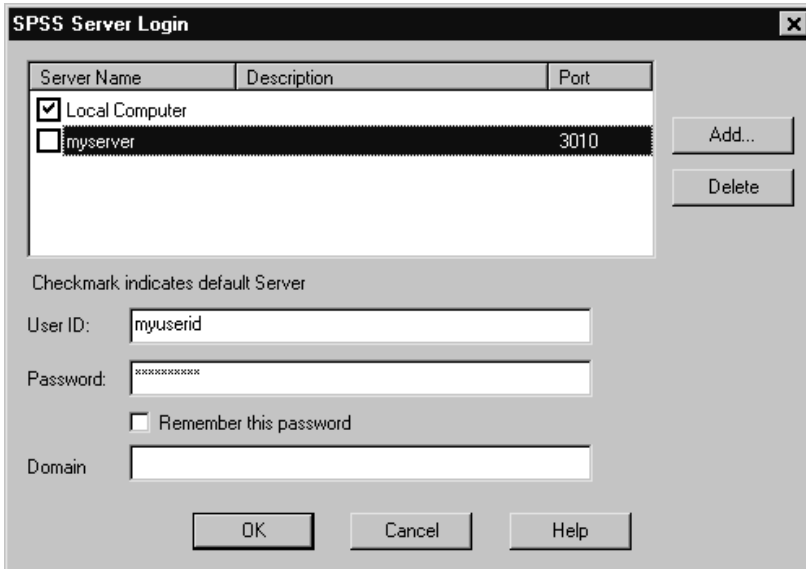
The next step is to log the client server Application object onto a server. This is done by adding your server name to a server's collection and then logging on. For example:

```
'Add a server.  
Dim objServers As ISpssServers  
objServers.Add "inet:myserver:3010"
```

```
'Get the server.  
Dim objServer As ISpssServer  
Set objServer = objServers.First
```

```
'Log the server in to the SPSS Server with an ID and password.  
objServer.Connect "myuserid", "mypasswd", 0
```

Figure 2-12
Server Login dialog box



The client server Application object has the same methods and properties as the SPSS application. Once you've created it, you work with it in the same way. The only difference is where the processing takes place. For a client server application, the processing takes place on the remote server.

Production mode also supports distributed analysis. You can log onto a remote server from the Options dialog box, as shown in Figure 2-13, and then run the production job as usual.

Figure 2-13
Production Facility options

The image shows the 'Options' dialog box for SPSS, specifically the 'Production Facility' options. The dialog has a title bar 'Options'. Inside, there is a section 'Editor for Syntax Files:' with a text box containing 'C:\WINNT\System32\notepad.exe' and a 'Browse...' button. Below this is a section 'SPSS Application Mode' with a checked checkbox 'Show SPSS when running' and two radio buttons: 'Leave SPSS open at completion of job' (selected) and 'Close SPSS at completion of job'. Further down is a section 'Remote Server' with a checked checkbox. Below the checkbox is a list box labeled 'Server Name' containing the entry 'qatest99 : 3010 : My test server'. To the right of the list box are three text boxes: 'User ID:' with 'myuserid', 'Password:' with '*****', and 'Domain Name:' which is empty. At the bottom are three buttons: 'OK', 'Cancel', and 'Help'.

Options

Editor for Syntax Files:

SPSS Application Mode

☒ Show SPSS when running

☒ Leave SPSS open at completion of job

☐ Close SPSS at completion of job

☒ Remote Server

Server Name

qatest99 : 3010 : My test server

User ID:

Password:

Domain Name:

OLE Automation Quickstart

This chapter introduces you to using OLE Automation with SPSS. It begins with an overview of OLE Automation and a code example that builds on the end-user tasks introduced in Chapter 2. It continues with descriptions of and programming examples for the SPSS OLE objects. It ends with an introduction to the SPSS object methods and properties and the SPSS type libraries. If you are already familiar with SPSS for Windows, this is a good place to start learning about SPSS OLE Automation. If you're unfamiliar with SPSS for Windows, read Chapter 2 first.

What Is OLE Automation?

If you have worked in Visual Basic or C++, you already know how to use objects that the program provides, such as command buttons, forms, and fields. OLE Automation is a technology standard that also allows you to use objects from other applications in your program. Because SPSS is fully enabled as an OLE Automation server, you can include SPSS objects as components of your program. Your program can run SPSS and take advantage of its extensive analytic capabilities.

OLE Automation is supported by a number of programming languages, including Visual Basic and C++. While the specific techniques for creating the application and accessing objects vary depending on the programming language, the techniques described in this chapter for manipulating SPSS objects are basically the same.

OLE Terminology

OLE Automation provides a standard set of interfaces for applications to provide objects to other applications, development tools, and macro languages. OLE takes advantage of, and is part of, the more general Component Object Model (COM). An **object**, in programming terminology, is a combination of code and data that can be treated as a unit; for example, a control, an item in a document, a document, or an application. An OLE object is also a component (or COM object).

All SPSS objects reside within the SPSS Application object, called the **OLE container**. Your program runs SPSS and then accesses the objects that it needs. The program that exposes the objects—in this case SPSS—is known as the **OLE Automation server**. The program that uses the objects—your program—is the **OLE Automation client**.

Figure 3-1 on p. 32 shows the types of objects, called **object classes**, that SPSS makes available, or exposes, to OLE Automation clients. Each object class has its own attributes and commands, called **properties** and **methods**, that define what you can do with that object.

Figure 3-1 also shows how the objects are related to each other hierarchically. That is important because many objects can exist only inside other objects. When you want to access objects lower in the hierarchy, you have to access the objects above them first, starting with the Application object.

SPSS object classes include the application itself, within which all other objects are contained; the file information object, which contains information about the SPSS data file; and objects for the different types of documents and output that SPSS produces.

Using Objects, Properties, and Methods

Like real-world objects, SPSS OLE Automation objects have features and uses. In programming terminology, the features are referred to as **properties**, and the uses are referred to as **methods**. Each object class has specific properties and methods that define what you can do with that object.

Object	Property	Method
pencil (real world)	Hardness Color	Write Erase
pivot table (SPSS)	TextFont DataCellWidths CaptionText	SelectTable ClearSelection HideFootnotes

Working with objects is a two-step process. First you get a reference to the object. Then you use its properties and methods to do something to the object.

How Do I Use OLE Automation with SPSS?

When you use SPSS OLE Automation you:

- ▶ Decide what you want your application to do with SPSS.
- ▶ Write the application code.

Deciding What You Want Your Application to Do

What Tasks Can Be Automated?

You can use OLE Automation to do most of the things you do when running SPSS interactively, including:

- Open and save SPSS data files and access data file information.
- Perform complex data manipulations and transformations using SPSS command syntax.
- Run SPSS statistical and graphical procedures to produce pivot tables, charts, and other statistical output.
- Automate repetitive tasks.
- Customize and manipulate output in the SPSS Viewer, including manipulations based on values in the output.
- Export output in HTML format for publication on the World Wide Web.
- Export charts as graphic files in a number of formats.
- Set options to customize the SPSS environment.

How Do I Figure Out Which Objects to Use?

The easiest way to figure out what you want to do with SPSS is to use it interactively with the dialog box user interface. Go through the sequence you want to run in your

application. At each step, think about the objects, methods, and properties that are in use. Table 3-1 shows the correspondence between high-level automation objects and the SPSS user interface.

Table 3-1

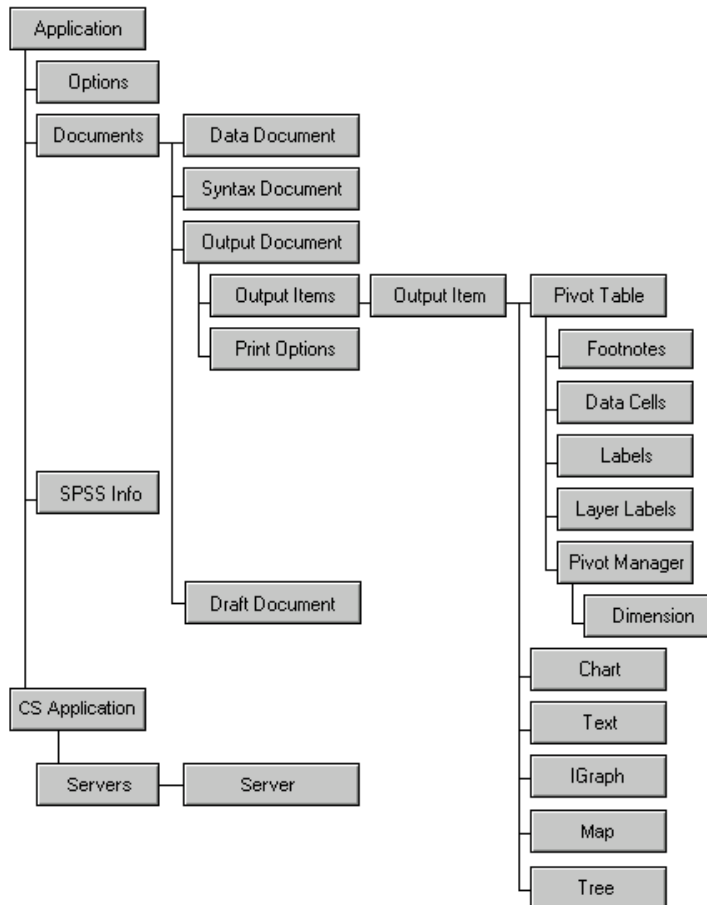
High-level OLE Automation objects and corresponding user interface

Object	User Interface
Application (ISpssApp)	SPSS for Windows application
Options (ISpssOptions)	Settings in the Options dialog box (Edit menu)
Spss Info (ISpssInfo)	None
Documents (ISpssDocuments)	All windows open in SPSS
Data Document (ISpssDataDoc)	Data Editor window
Syntax Document (ISpssSyntaxDoc)	Syntax window
Draft Document (ISpssDraftDoc)	Draft Viewer window
Output Document (ISpssOutputDoc)	Viewer window
Output Items (ISpssItems)	All items in the Viewer window
Pivot Table (PivotTable)	Produced by many items on the Analyze menu
IGraph (ISpssIGraph)	Produced by items on the Graphs > Interactive submenu
Text (ISpssRtf)	Produced by some items on the Analyze menu
Chart (ISpssChart)	Produced by items on the Graphs menu
Map	Produced by items on the Graphs > Map submenu
Tree Model (TreeModelControl)	Produced by Analyze>Classify>Tree

Note: The **Viewer** window is where the interactive user sees and manipulates output—it corresponds to the **Output Document** object, even though the names are different. This is because the window name in the user interface changed after the OLE Automation interface was defined.

Figure 3-1 shows the complete SPSS object model. “Object Browser and Online Help” on p. 71 explains how to browse and get Help on SPSS objects, methods, and properties while you are writing your application.

Figure 3-1
SPSS object model



Example of Interactive Use and Corresponding OLE Objects

The following steps show a simple sequence of actions that you can do with the SPSS user interface and the corresponding OLE Automation objects. Detailed code for this example is in “Writing Application Code” on p. 34.

► Launch SPSS.

User interface:

From the Windows Start menu choose:

Programs
SPSS for Windows
SPSS for Windows

OLE Automation:

```
Dim objSpssApp As ISpssApp
Set objSpssApp = CreateObject("SPSS.Application")
```

- **Open a data file.** For example, open the employee data file.

User interface:

From the SPSS menus choose:

File
Open
Data...

Select *employee data.sav*.

OLE Automation:

```
Dim objDataDoc As ISpssDataDoc
Set objDataDoc = objSpssApp.OpenDatadoc ("c:\spss\employee data.sav")
```

- **Open a dialog box.** For example, open the Frequencies dialog box.

User interface:

From the SPSS menus choose:

Analyze
Descriptive Statistics
Frequencies...

OLE Automation:

```
objDataDoc.InvokeDialogAndExecuteSyntax ("analyze>descriptive statistics>frequencies",  
SpssWindowParent, True)
```

- **Save output.**

User interface:

Click in the Viewer and from the menus choose:

File
Save

OLE Automation:

```
Dim objOutputDoc as ISpssOutputDoc
Set objOutputDoc = objSpssApp.GetDesignatedOutputDoc
objOutputDoc.SaveAs ("c:\spss\output1.spo")
```

Deciding the Extent of SPSS Integration into Your Application

Your programs can run simultaneously with SPSS on the user's desktop (and take advantage of the SPSS user interface), or SPSS can run in the background. The level of integration is up to you.

If you want to take advantage of the SPSS user interface, write applications that open dialog boxes using the `InvokeDialogAndExecuteSyntax` method on the Data Document object and use the `Visible` property on the document objects to make the SPSS windows visible.

If you want to run SPSS in the background, write applications that submit syntax. For example, use the `ExecuteCommands` method on the Application object.

Writing Application Code

When you've decided how your application will use SPSS and the extent of integration of SPSS into your application, it is time to start writing code. Writing code is typically done in a development environment that is tailored to the programming language that you are using. In this section, we will proceed through the same basic steps as in the example on p. 32, using Visual Basic as the programming language. The code starts SPSS, opens a data file, runs a procedure, edits and saves the procedure's output, and exits SPSS. You may find it useful to refer to the SPSS OLE object hierarchy in Figure 3-1 as you read through the example.

This example demonstrates a number of important OLE Automation techniques, including how to create the application and how to use properties or methods of higher-level objects to get at objects lower in the object hierarchy. The example also shows how to run a statistical procedure using SPSS command syntax. (See Chapter 2 for an introduction to SPSS command syntax.)

To write the code:

- **Include the SPSS type libraries in your project.** In Visual Basic, select References from the Project menu. The SPSS type libraries are listed and described in “SPSS Type Libraries” on p. 70.
- **Declare variables.** Although not always strictly required, it is a good idea to declare all variables before using them:

```
' Application object for SPSS.  
Dim objSpssApp As ISpssApp
```

```
' Data Document object for the data file.  
Dim objDataDoc as ISpssDataDoc
```

```
' Output Document object to store the output.  
Dim objOutputDoc As ISpssOutputDoc
```

```
' String to store the syntax for the procedure.  
Dim strCommand As String
```

By convention, the name of each variable indicates its type. Object variable names begin with obj, integer variables begin with int, and string variables begin with str. (These conventions are described more fully in Appendix C.)

For object variables, the name also indicates the object class to which the variable is assigned. For example, the first declaration above creates an object variable named `objSpssApp` and specifies its type as `ISpssApp` (belonging to the Application object class). The variable does not have a value until the application is actually created—all the statement does is declare that the variable exists.

Note: You can declare your object variables as an object class (such as `ISpssApp` or `ISpssOutputDoc`) only if your programming language supports a method called **variable binding**, which allows for early binding of variables at compilation time. Most programs support this method, but if the variable declarations produce an error, declare the variables as `Object`. For example,

```
Dim objSpssApp As Object  
Dim objOutputDoc As Object
```

- **Create the SPSS application.** Creating the application means to start SPSS and get a reference to the Application object so that you can access its properties and methods. The specific techniques for creating the application vary, depending on what

programming language you are using. In Visual Basic, you can use the `CreateObject` function:

```
Set objSpssApp = CreateObject("SPSS.Application")
```

This statement starts SPSS and it stores a reference to the Application object in the variable.

- **Open a data file.** You work with SPSS by getting its objects. To **get** an object means to create a reference to the object so that you can use properties and methods to do something. Each object reference that you get is stored in a variable. You have already seen how to create (or get) the Application object using Visual Basic's `CreateObject` function. However, most other SPSS objects cannot be created directly. Instead, you get them by using properties and methods on other, high-level objects. For example, once you have created the Application object, you can use the `OpenDataDoc` method to create a Data Document object and open a data file:

```
Set objDataDoc = objSpssApp.OpenDatadoc ("c:\spss\employee data.sav")
```

SPSS requires data before it will run procedures.

- **Run procedures.** Next, analyze the data. For this example, SPSS will run in the background and we'll submit syntax. The results of the analysis are placed in an output document, so we will create that object, too:

```
Set objOutputDoc = objSpssApp.NewOutputDoc
```

```
' Create the procedure command syntax
strCommand = strCommand + "DESCRIPTIVES"
strCommand = strCommand + " VARIABLES=salary salbegin"
strCommand = strCommand + " /STATISTICS=MEAN STDDEV MIN MAX ."
```

```
' Run the procedure.
objSpssApp.ExecuteCommands strCommand, True
```

- **Modify the output.** Now the analysis is complete and the results are in the Output Document object, ready for modification:

```
' SPSS can have more than one output document. This gets the one that is
' designated to receive output.
Set objOutputDoc=objSpssApp.GetDesignatedOutputDoc
```

```
' Clear the current selection.  
objOutputDoc.ClearSelection  
  
' Select all notes in the output document.  
objOutputDoc.SelectAllNotes  
  
' Delete the selection (all notes).  
objOutputDoc.Remove
```

In this step, we modified a collection of output items of the same type. You may want to edit an individual output item. In general, with SPSS OLE Automation you use properties or methods on higher-level objects to get at the objects beneath. The Output Document object is a good example of this. It has a property called `Items` that returns the Output Items Collection:

```
Set objOutputItems = objOutputDoc.Items()
```

And the Output Items Collection has a method called `GetItem` that returns an individual output item:

```
Set objOutputItem = objOutputItems.GetItem(2)
```

- **Save the output.** Finally, save the edited output:

```
objOutputDoc.SaveAs ("c:\myoutput\myoutput.spo")
```

- **Close SPSS.** When you have finished using the SPSS Application object, you can close it by:

```
objSpssApp.Quit
```

SPSS Objects, Methods, and Properties

In Figure 3-1, you saw the object classes that SPSS exposes to OLE Automation clients. If you use SPSS or have read Chapter 2, most of these object types should be familiar to you and you can probably guess the properties and methods associated with them.

The object model also shows how the objects are related to each other, which is important because most of the objects exist only inside other objects. You start by creating the Application object and navigate down the object hierarchy tree. Lower-level objects, such as data cells and labels, exist only as part of a higher-level object and cannot be created directly. For example, to change or format column labels in a

pivot table, you need to get all of the objects above and including the Column Labels object.

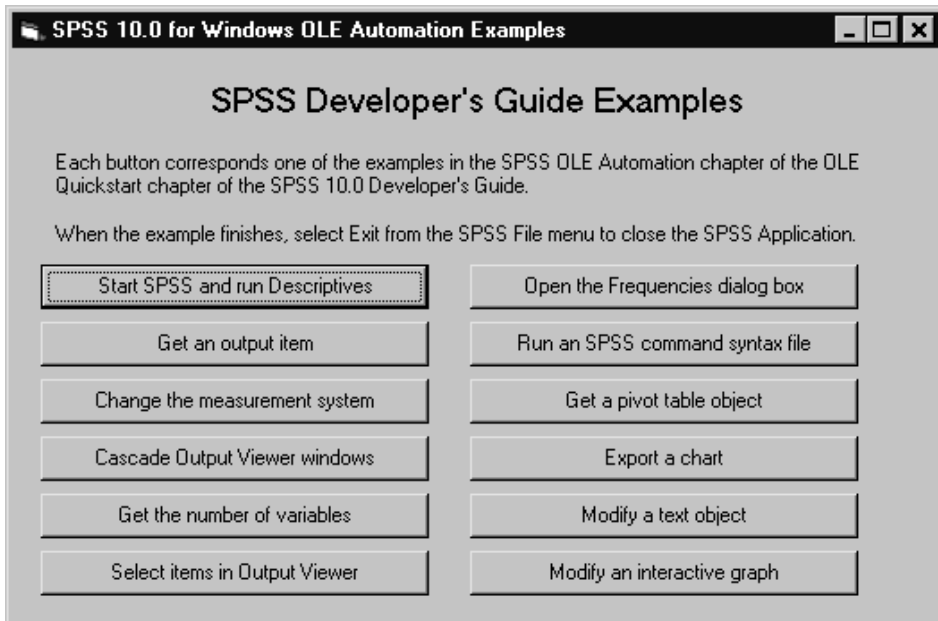
Note: The online Help for SPSS OLE Automation contains the same diagram, with links to Help topics for each object, including Help for each object's methods and properties. "Object Browser and Online Help" on p. 71 describes how to access the online Help.

Objects

In the following sections, each high-level object is described and its use is demonstrated by an example. The examples are written in Visual Basic. There is some overlap in the code for the examples—this ensures that each can be run independently. SPSS is made visible so that you can see what the example does. In a real-life application, you may choose to hide SPSS. The code for these examples is included on the SPSS for Windows CD-ROM in the Visual Basic project file `\SPSS\Developer\Programs\OLE Quickstart\spssole.vbp`. To get the most from the examples, open the project in Visual Basic and step through the code. Chapter 5 describes additional sample programs supplied with the developer's tools.

Figure 3-2

Example Visual Basic project user interface (spssole.vpb)



Application Object (ISpssApp)

The Application object is the container object inside of which all other SPSS objects exist. It is a user-creatable object, meaning that your program can run SPSS and access its properties directly. Other objects must exist inside higher-level objects. Access them by applying properties and methods on these higher-level objects. The Application object has properties to access the SPSS Options object, the File Information object, and the Documents object.

To get the Application object, declare an object variable as ISpssApp and create the object:

```
Dim objSpssApp as ISpssApp
Set objSpssApp = CreateObject("SPSS.Application")
```

By default, SPSS runs in the background when created through OLE Automation. You can use the Visible property to display the Data Editor, Viewer, or syntax windows:

```
objDataDoc.Visible = True
```

To avoid leaving SPSS running in the background, use the Quit method to exit SPSS before closing your program:

```
objSpssApp.Quit
```

The SPSS OLE Automation server does not alert you before overwriting files when it exits.

Figure 3-3 shows a simple example that starts SPSS, opens a data file, and produces a table of descriptive statistics. Figure 3-4 shows the result of running the example—the descriptive statistics are displayed in the Viewer.

Figure 3-3

Start SPSS Application object and run Descriptives procedure

'Example 1: Start SPSS and run Descriptives.

```
Private Sub cmdExample1_Click()
```

```
'Declare object variables.
```

```
Dim objSpssApp As ISpssApp
```

```
Dim strAppPath As String
```

```
Dim strFileName As String
```

```
Dim objOutputDoc As ISpssOutputDoc
```

```
Dim objDataDoc As ISpssDataDoc
```

```
Dim strCommand As String
```

```
'Create the application (start SPSS).
```

```
Set objSpssApp = CreateObject("SPSS.Application")
```

```
'Open a new Output Navigator.
```

```
Set objOutputDoc = objSpssApp.NewOutputDoc
```

```
'Get the SPSS installation directory.
```

```
' This example uses an example data file that was installed with SPSS.
```

```
strAppPath = objSpssApp.GetSPSSPath
```

```
'Build a path to the data file you want to open.
```

```
strFileName = strAppPath & "employee data.sav"
```

'Open the data file.

```
Set objDataDoc = objSpssApp.OpenDataDoc(strFileName)
```

'Display the Data Editor.

```
objDataDoc.Visible = True
```

'Run Descriptives procedure using command syntax.

```
strCommand = strCommand + "DESCRIPTIVES"
```

```
strCommand = strCommand + " VARIABLES=salary salbegin"
```

```
strCommand = strCommand + " /STATISTICS=MEAN STDDEV MIN MAX ."
```

```
objSpssApp.ExecuteCommands strCommand, True
```

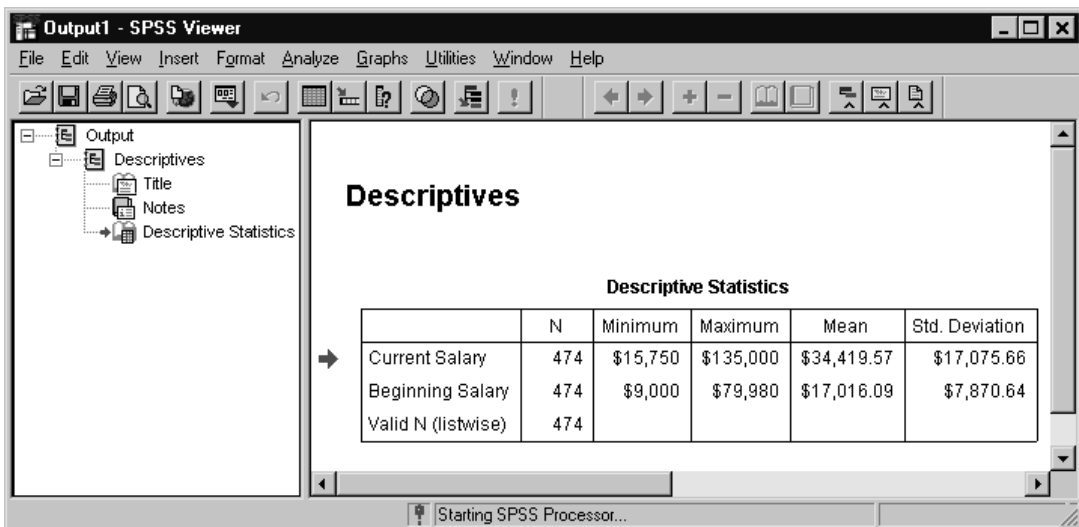
'Display Viewer window.

```
objOutputDoc.Visible = True
```

```
End Sub
```

Figure 3-4

Viewer displaying Descriptives results



Getting versus Creating the SPSS Application Object

Beginning with SPSS 10.0, multiple instances of the SPSS application can run on a computer, so your code needs to check to see if the SPSS application is already

running. If the application is running, use `GetObject`. If the application is not running use `CreateObject`. Here's an example:

```
Public Function GetSpss() As Application
    On Error Resume Next

    'Get a reference to existing SPSS.
    Set GetSpss = GetObject(Class:="Spss.Application")
    Debug.Print Err; Err.Description

    'There will be an error if no SPSS is running or if
    ' an SPSS version prior to 10.0 is running.
    ' If there is an error then we will create the SPSS application object.
    ' For SPSS versions prior to 10.0, CreateObject gets the running instance.
    If Err Then
        Err.Clear
        Set GetSpss = CreateObject("Spss.Application")
    End If
End Function
```

Options Object (ISpssOptions)

The Options object allows you to specify options for the Viewer, charts, pivot tables, and data and currency formats. In the user interface, these settings are specified in the Options dialog box, which is accessed from the Edit menu.

To get the Options object, declare an object variable as `ISpssOptions` and set it to the Options property of the Application object:

```
Dim objSPSSOptions as ISpssOptions
Set objSPSSOptions = objSpssApp.Options
```

Figure 3-5 shows an example that starts SPSS and changes the measurement system from inches to points. Figure 3-6 shows the Options dialog box with the changed measurement system.

Figure 3-5

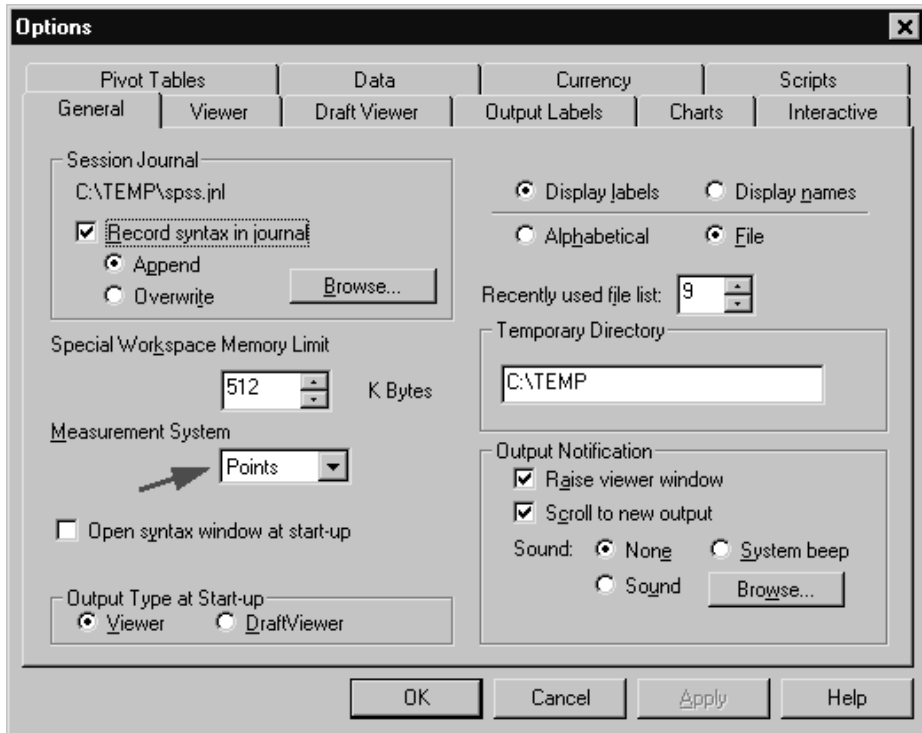
Change measurement system with Options object

'Example 3: Change the measurement system.
Private Sub cmdExample3_Click()

```
' Declare variables.
Dim objSpssApp As ISpssApp
Dim objDataDoc As ISpssDataDoc
Dim objSpssOptions As ISpssOptions
```

```
'Create the SPSS application.  
Set objSpssApp = CreateObject("SPSS.Application")  
  
'Open a new Data Editor.  
Set objDataDoc = objSpssApp.NewDataDoc  
  
'Display Data Editor.  
objDataDoc.Visible = True  
  
'Get the Options object.  
Set objSpssOptions = objSpssApp.Options  
  
'Set measurement system to points (it is inches by default).  
objSpssOptions.MeasurementSystem = 0  
  
'If SPSS is running hidden, changed options settings are  
' not saved beyond the current session.  
' In this example we ran SPSS visible, so the change is saved.  
'Be sure to set it back to inches if that is what you use.  
  
End Sub
```

Figure 3-6
SPSS Options dialog box



Documents Collection Object (ISpssDocuments)

The Documents Collection object provides access to the collection of SPSS documents, including data, output, and syntax documents. The object has properties that return the number of open documents of each type and methods to get documents of each type. This is one of several collection objects that exist primarily to allow you to get other objects.

To get the Documents Collection object, declare an object variable as `ISpssDocuments` and set it to the Documents property of the Application object:

```
Dim objDocuments as ISpssDocuments
Set objDocuments = objSpssApp.Documents
```

Figure 3-7 shows an example that starts SPSS and uses the Documents Collection to cascade windows. Figure 3-8 shows the cascaded windows.

Figure 3-7

Cascade Viewer windows with Documents Collection object

'Example 4: Cascade Viewer windows.

Private Sub cmdExample4_Click()

'Declare variables.

Dim objSpssApp As ISpssApp

Dim objDocuments As ISpssDocuments

Dim objOutputDoc As ISpssOutputDoc

'Create the SPSS application.

Set objSpssApp = CreateObject("SPSS.Application")

'Open three Viewer windows.

Set objOutputDoc = objSpssApp.NewOutputDoc

Set objOutputDoc = objSpssApp.NewOutputDoc

Set objOutputDoc = objSpssApp.NewOutputDoc

'Get the documents collection.

Set objDocuments = objSpssApp.Documents

'Loop through the documents collection, cascade the

' Viewer windows, and make them visible.

Dim intCount As Integer

Dim intWindowPos As Integer

intWindowPos = 40

intCount = objDocuments.OutputDocCount

For I = 0 To intCount - 1

 intWindowPos = intWindowPos + 60

 Set objOutputDoc = objDocuments.GetOutputDoc(I)

 objOutputDoc.Top = intWindowPos

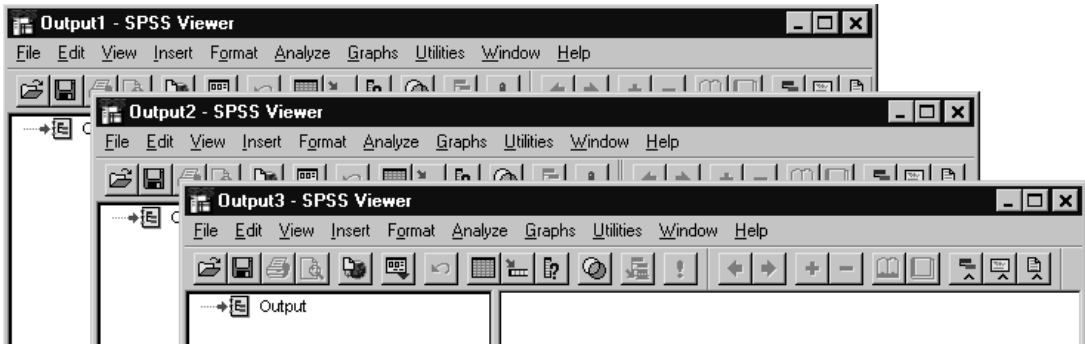
 objOutputDoc.Left = intWindowPos

 objOutputDoc.Visible = True

Next

End Sub

Figure 3-8
Cascaded Viewer windows



File Information Object (ISpssInfo)

The File Information object provides access to dictionary information on SPSS data files, including variable names, labels, sequential position of each variable in the file, print and write formats, missing values, and value labels.

To get the File Information object, declare an object variable as ISpssInfo and set it to the SpssInfo property of the Application object:

```
Dim objSpssInfo as ISpssInfo
Set objSpssInfo = objSpssApp.SpssInfo
```

You can also use the GetVariableInfo method on the Data Document to retrieve complete dictionary information with one call. Depending on your needs, GetVariableInfo can be more efficient than using the SPSS File Information object.

Figure 3-9 shows an example that starts SPSS and gets the number of variables in from an open data file. First it uses the File Information object and then it uses the Data Document object.

Figure 3-9
Get number of variables

'Example 5: Get the number of variables.

```
Private Sub cmdExample5_Click()
```

```
'Declare variables
```

```
Dim objSpssApp As ISpssApp
```

```
Dim objDataDoc As ISpssDataDoc
```

```
Dim objSpssInfo As ISpssInfo
Dim strAppPath As String
Dim strFileName As String
Dim intCountFileInfo As Integer

'Declare variants for GetVariableInfo method.
Dim numVarsDataDoc As Long
Dim vrtVarNames As Variant
Dim vrtVarLabels As Variant
Dim vrtVarTypes As Variant
Dim vrtVarMsmntLevels As Variant
Dim vrtLabelCounts As Variant

'Create the SPSS application
Set objSpssApp = CreateObject("SPSS.Application")

'Open a data file and make it visible so you can manually check the number of variables.
strAppPath = objSpssApp.GetSPSSPath
strFileName = strAppPath & "employee data.sav"
Set objDataDoc = objSpssApp.OpenDataDoc(strFileName)
objDataDoc.Visible = True

'There are two ways to get the number of variables:
' from the File Information object and from the
' GetVariableInfo method on the Data Document object.
' The Data Document object may perform faster.

'Get the File Information object and read the number of variables.
Set objSpssInfo = objSpssApp.SpssInfo
intCountFileInfo = objSpssInfo.NumVariables

'Display the number of variables from ISpssInfo in a message box.
Dim strMsgInfo As String
Dim intResponseInfo As Integer
Dim strTitleInfo As String
strMsgInfo = "The number of variables from ISpssInfo:" & intCountFileInfo
strTitleInfo = "SPSS OLE Automation"
intResponseInfo = MsgBox(strMsgInfo, vbOKOnly, strTitleInfo)

'Get the Data Document object and use GetVariableInfo to read the number of variables.
numVarsDataDoc = objDataDoc.GetVariableInfo(vrtVarNames, vrtVarLabels, vrtVarTypes,
vrtMsmntLevels, vrtLabelCounts)

'Display the number of variables from ISpssDataDoc in a message box.
Dim strMsgData As String
```

```

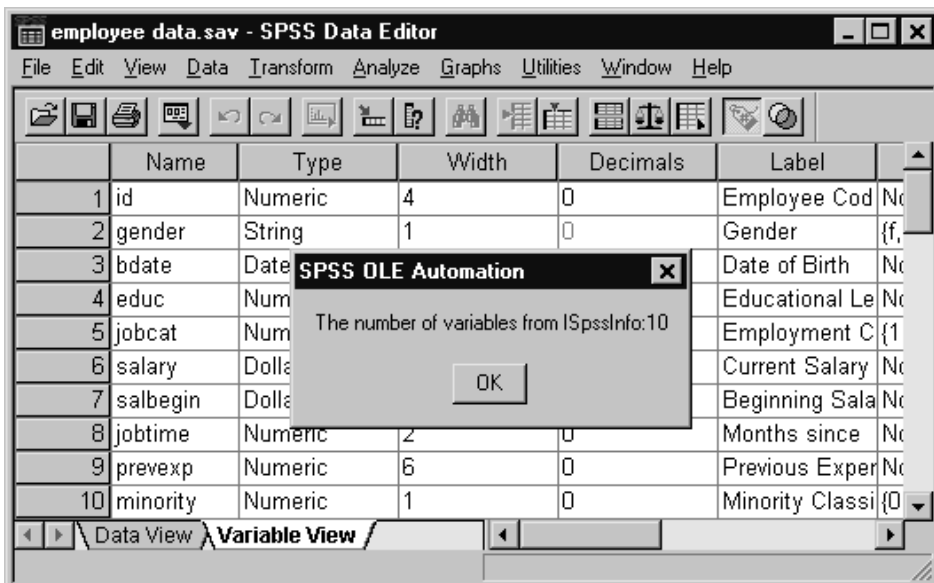
Dim intResponseData As Integer
Dim strTitleData As String
strMsgData = "The number of variables from ISpssDataDoc:" & numVarsDataDoc
strTitleData = "SPSS OLE Automation"
intResponseData = MsgBox(strMsgData, vbOKOnly, strTitleData)

End Sub

```

Figure 3-10

Example application displaying number of variables



Data Document Object (ISpssDataDoc)

The Data Document object is the SPSS Data Editor, with or without a working data file. You need a working data file before you can run any statistical analysis. Use the OpenDataDoc or NewDataDoc method on the SPSS Application object to create a Data Document:

```

Dim objDataDoc as ISpssDataDoc
Set objDataDoc = objSpssApp.OpenDatadoc ("c:\employee data.sav")

```

You can also copy, paste, save, and print data, and get attributes of the working data file, including the number of cases and variables, weighting and filter variables, window size and state, and whether or not toolbars and value labels are displayed.

You can use the Data Document object only to get attributes of data. If you want to set data attributes—for example, to specify a weighting variable rather than getting the current setting—use SPSS command syntax.

You do not need to close a Data Document. When you open a new data file or quit SPSS, the working data file is closed.

Figure 3-11 shows an example that starts SPSS, opens a working data file, and opens a dialog box for statistical analysis. Figure 3-12 shows the Data Editor and the dialog box.

Figure 3-11

Open Frequencies dialog box with Data Document object

'Example 6: Open the Frequencies dialog box.

```
Private Sub cmdExample6_Click()
```

```
'Declare variables.
```

```
Dim objSpssApp As ISpssApp
```

```
Dim objDataDoc As ISpssDataDoc
```

```
Dim strAppPath As String
```

```
Dim strFileName As String
```

```
'Create the SPSS application.
```

```
Set objSpssApp = CreateObject("SPSS.Application")
```

```
'Open a data file.
```

```
strAppPath = objSpssApp.GetSPSSPath
```

```
strFileName = strAppPath & "employee data.sav"
```

```
Set objDataDoc = objSpssApp.OpenDataDoc(strFileName)
```

```
'Open the Frequencies dialog box.
```

```
Dim strPath As String
```

```
strPath = "statistics>summarize>frequencies"
```

```
objDataDoc.InvokeDialogAndExecuteSyntax strPath, SpssWindowParent, True
```

```
'At this point the end user of your application would request an analysis.
```

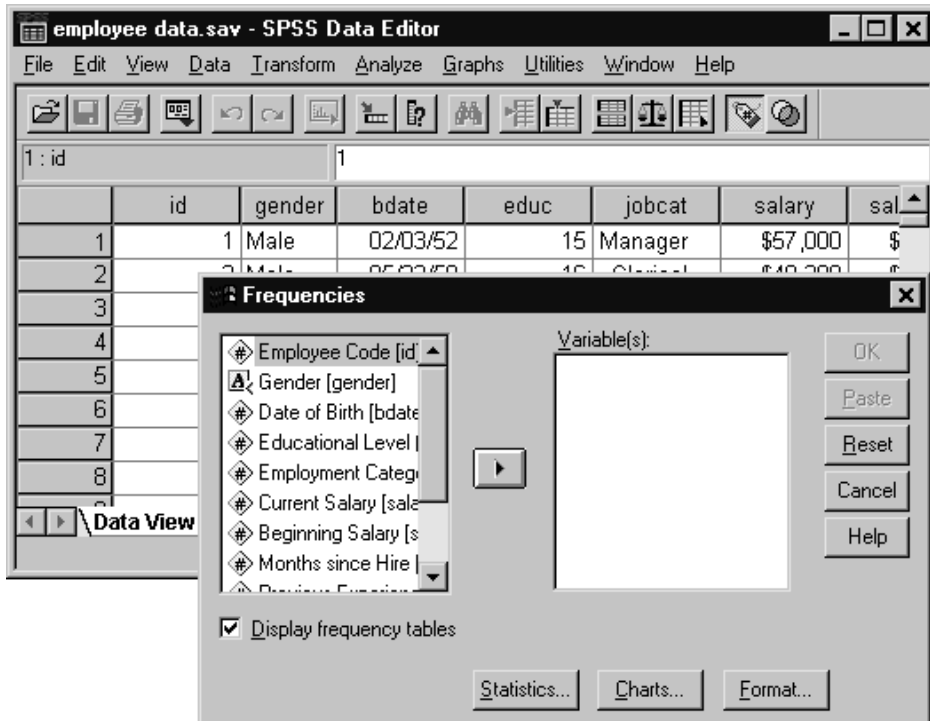
```
End Sub
```

Note: InvokeDialogAndExecuteSyntax can be used by other SPSS document objects as long as there is a working data file. The InvokeDialogAndExecuteSyntax topic in the

online Help for SPSS OLE Automation lists all of the valid menu paths. “Object Browser and Online Help” on p. 71 describes how to access the online Help.

Figure 3-12

SPSS Data Editor and Frequencies dialog box



Syntax Document Object (ISpssSyntaxDoc)

The Syntax Document object is an open syntax window for pasting, running, and saving SPSS command syntax files. For more information about SPSS command syntax, see Chapter 2.

Use the NewSyntaxDoc or OpenSyntaxDoc method on the SPSS Application object to open a syntax document:

```
Dim objSyntaxDoc as ISpssSyntaxDoc
Set objSyntaxDoc = objSpssApp.OpenSyntaxDoc ("c:\weekly.sps")
```

Use the `GetDesignatedSyntaxDoc` method on the Application object or the `GetSyntaxDoc` method on the Documents object to get a specific open syntax document. To close a syntax document, use the `Close` method on the Syntax Document object.

Figure 3-13 shows an example that starts SPSS and opens and runs a syntax file. Figure 3-14 shows the syntax window.

Figure 3-13

Run SPSS command syntax file

'Example 7: Run an SPSS command syntax file.

```
Private Sub cmdExample7_Click()
```

```
'Declare variables.
```

```
Dim objSpssApp As ISpssApp
```

```
Dim objSyntaxDoc As ISpssSyntaxDoc
```

```
Dim objOutputDoc As ISpssOutputDoc
```

```
Dim strAppPath As String
```

```
Dim strFileName As String
```

```
'Create the SPSS application.
```

```
Set objSpssApp = CreateObject("SPSS.Application")
```

```
'Open an SPSS syntax file and make it visible.
```

```
' This example uses an example syntax file that was installed with SPSS.
```

```
strAppPath = objSpssApp.GetSPSSPath
```

```
strFileName = strAppPath & "Descriptive Statistics.sps"
```

```
Set objSyntaxDoc = objSpssApp.OpenSyntaxDoc(strFileName)
```

```
objSyntaxDoc.Visible = True
```

```
'Run the syntax in the file.
```

```
objSyntaxDoc.Run
```

```
'Normally you would close the syntax file with the following command.
```

```
' We left it open so you can see it.
```

```
objSyntaxDoc.Close
```

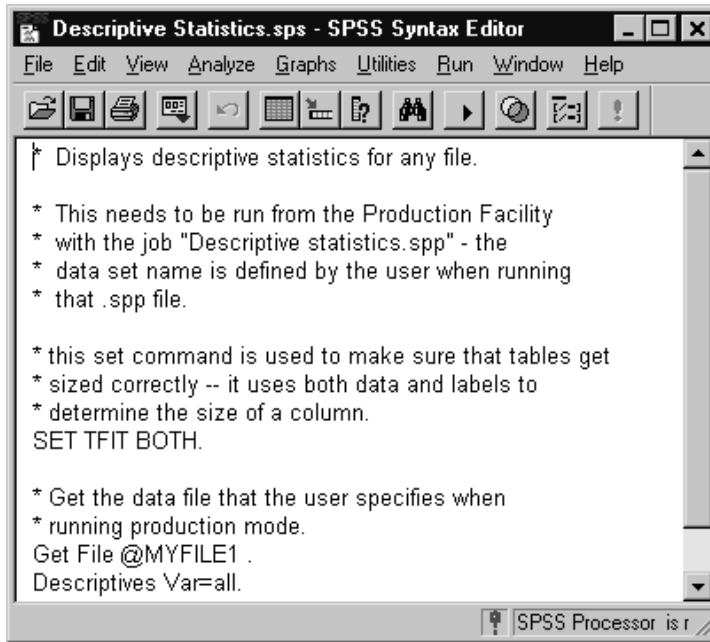
```
'Note: This syntax file was designed for the SPSS Production Facility, so
```

```
' it doesn't do anything in SPSS. It was used for this example
```

```
' because it is shipped with the SPSS product.
```

```
End Sub
```

Figure 3-14
Syntax window



Output Document Object (ISpssOutputDoc)

The Output Document object is an open Viewer document. This object contains the Output Items Collection and a Print Options object. Use the Output Items Collection object to manipulate output items at the outline level. You can cut, remove, promote, or demote selected output items. You can select all output items of a particular type, such as charts or notes tables, and export all or selected charts in any of a number of graphics formats (to access individual output items, you have to first access the Output Items Collection, described in the next section).

Use the NewOutputDoc or OpenOutputDoc method on the SPSS Application object to open an output document:

```
Dim objOutputDoc as ISpssOutputDoc
Set objOutputDoc = objSpssApp.OpenOutputDoc("c:\myoutput.spo")
```

Use the `GetDesignatedOutputDoc` method on the SPSS Application object or the `GetOutputDoc` method on the Documents object to get an output document that is already open:

```
Set objOutputDoc = objSpssApp.GetDesignatedOutputDoc
```

To close an output document, use the `Close` method on the Output Document object.

Figure 3-15 shows a Visual Basic example that starts SPSS, opens a data file, creates output items, and selects and removes all items that are notes.

Figure 3-15

Select and remove items from output

'Example 12: Select items in the Viewer window.

' This example selects and removes notes.

```
Private Sub cmdExample12_Click()
```

'Declare variables.

```
Dim objSpssApp As ISpssApp
```

```
Dim objDataDoc As ISpssDataDoc
```

```
Dim objOutputDoc As ISpssOutputDoc
```

'Create SPSS Application.

```
Set objSpssApp = CreateObject("SPSS.Application")
```

'Open a data file and create some output.

```
strAppPath = objSpssApp.GetSPSSPath
```

```
strFileName = strAppPath & "employee data.sav"
```

```
Set objDataDoc = objSpssApp.OpenDataDoc(strFileName)
```

```
strCommand = strCommand + "DESCRIPTIVES"
```

```
strCommand = strCommand + " VARIABLES=salary salbegin"
```

```
strCommand = strCommand + " /STATISTICS=MEAN STDDEV MIN MAX ."
```

```
objSpssApp.ExecuteCommands strCommand, True
```

'Get the Viewer window and make it visible.

```
Set objOutputDoc = objSpssApp.GetDesignatedOutputDoc
```

```
objOutputDoc.Visible = True
```

'Select all notes in the output document and remove them.

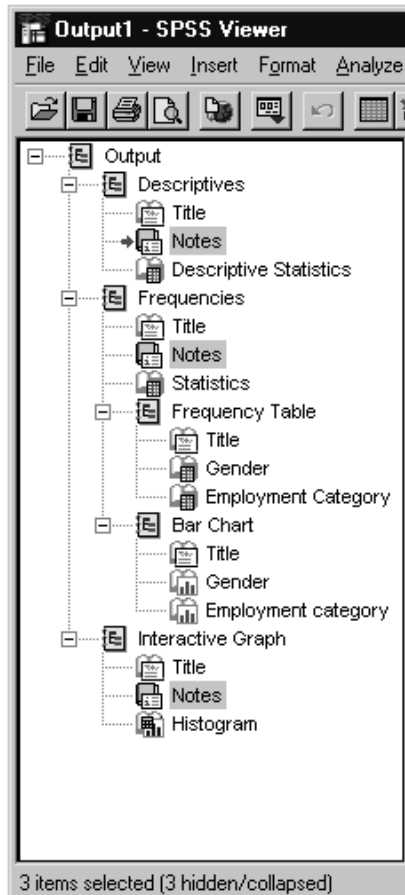
```
objOutputDoc.ClearSelection
```

```
objOutputDoc.SelectAllNotes
```

```
objOutputDoc.Remove
```

```
End Sub
```

Figure 3-16
Outline with items selected prior to removal



Output Items Collection (ISpssItems)

The Output Items Collection contains the items in an open output document. This object has a single property, Count, and single method, GetItem, that you use to get at the individual output items beneath it. For example, Figure 3-19 on p. 58 shows how to loop through the Items Collection and get the first pivot table of a given type.

To get the Output Items object, declare an object variable as `ISpssItems` and set it to the `Items` property of the Output Document object:

```
Dim objOutputItems As ISpssItems
Set objOutputItems = objOutputDoc.Items()
```

Note that the collection is zero-based index, so the first item is item zero, the second is item one, and so on. (Item zero is the root item labeled *SPSS Output* that appears even in an empty output document and cannot be deleted.)

Figure 3-17 shows an example that starts SPSS, opens a data file, creates output items, and gets an arbitrary item. Figure 3-18 shows the result of running the example—the arbitrary item is selected in the Viewer.

Figure 3-17

Get output item from Output Items Collection

'Example 2: Get an output item from the output items collection.

```
Private Sub cmdExample2_Click()
```

```
'Declare variables.
```

```
Dim objSpssApp As ISpssApp
Dim strAppPath, strFileName As String
Dim objDaaDoc As ISpssDataDoc
Dim objOutputDoc As ISpssOutputDoc
Dim objOutputItems As ISpssItems
Dim objOutputItem As ISpssItem
```

```
'Create SPSS Application.
```

```
Set objSpssApp = CreateObject("SPSS.Application")
```

```
'Open a data file and create some output.
```

```
strAppPath = objSpssApp.GetSPSSPath
```

```
strFileName = strAppPath & "employee data.sav"
```

```
Set objDataDoc = objSpssApp.OpenDataDoc(strFileName)
```

```
strCommand = strCommand + "DESCRIPTIVES"
```

```
strCommand = strCommand + " VARIABLES=salary salbegin"
```

```
strCommand = strCommand + " /STATISTICS=MEAN STDDEV MIN MAX ."
```

```
objSpssApp.ExecuteCommands strCommand, True
```

```
'Get the Viewer window and make it visible.
```

```
Set objOutputDoc = objSpssApp.GetDesignatedOutputDoc
```

```
objOutputDoc.Visible = True
```

```
'Get the Output Items collection.
```

```
Set objOutputItems = objOutputDoc.Items()
```

```

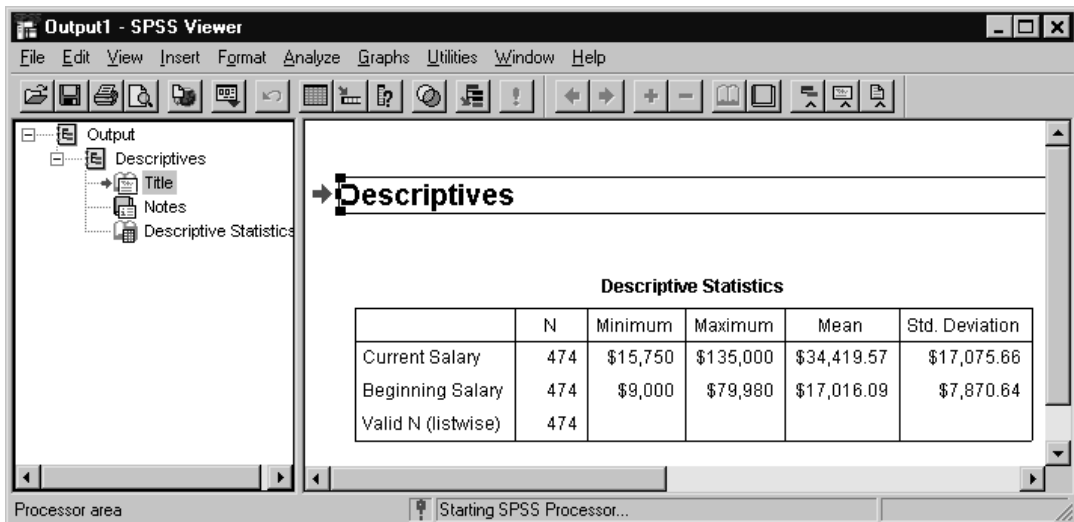
'Get the third output item.
'(Items are numbered starting at 0, thus item 2 is the third item)
Set objOutputItem = objOutputItems.GetItem(2)

'Select the item we just got.
objOutputItem.Selected = True

End Sub

```

Figure 3-18
Viewer displays selected output item



Output Item Object (ISpslItem)

The Output Item object is any item contained in an open Viewer window, including pivot tables, charts, and text output. Use this object to select, remove, activate, and modify output items.

To get an arbitrary output item, first get the Output Items Collection and then use the GetItem method. For example, as shown in Figure 3-17, to get the third item in the (zero-based) collection:

```

Dim objOutputItem As ISpslItem
Set objOutputItem = objOutputItems.GetItem(2)

```

More often you'll want to loop through the Items Collection to get items that meet specified criteria. For example, use the `SpssType` property on the Output Item object to get the item type and then test if the item is of the desired type:

```
Dim objOutputItem As ISpssItem
Dim intItemType As Integer
intItemType = objOutputItem.SPSSType
If intItemType = SPSSINote Then
```

The examples shown in Figure 3-19, Figure 3-21, Figure 3-23, and Figure 3-25 all use this technique to access different types of items.

Pivot Table Object (PivotTable)

The Pivot Table object is an activated pivot table. You can use automation to do most of the things you can do in the Pivot Table Editor. There are two ways you can use this object:

- Select groups of cells (results or labels) or other elements (such as footnotes) and apply properties and methods that modify the entire selection. For example, you can use the `ForegroundColor` property to change the foreground color for selected cells.
- Get an individual element and modify it using properties and methods that apply to the sub-objects contained in the pivot table. For example, with the Data Cells object, you can use the `ForegroundColorAt` property to set the foreground color for the current data cell. A number of sub-objects are contained within the Pivot Table object, including Footnotes, Data Cells, Row and Column Labels, Layer Labels, and the Pivot Manager.

To get a Pivot Table object, loop through the Items Collection as shown in Figure 3-19. The SPSS Base system also includes a number of sample scripts that demonstrate techniques for manipulating pivot tables. For a brief introduction to scripting, see “Working with the SPSS Scripting Facility” on p. 22 in Chapter 2. For more information about scripting, see Chapter 4 in this document, the “Scripting Facility” chapter in the *SPSS Base User's Guide*, and the online Help.

Figure 3-19 shows an example that starts SPSS, opens a data file, creates output items, and gets a pivot table. Figure 3-20 shows the result of running the example—the pivot table is activated in the Viewer.

Figure 3-19

Get and activate pivot table object

'Example 8: Get a pivot table object.

Private Sub cmdExample8_Click()

'Declare variables.

Dim objOutputDoc As ISpssOutputDoc

Dim objOutputItems As ISpssItems

Dim objOutputItem As ISpssItem

Dim objPivotTable As PivotTable

Dim strAppPath As String

Dim strFileName As String

Dim strCommand As String

'Start SPSS.

Set objSpssApp = CreateObject("SPSS.Application")

'Open a data file and create some output so we can get a pivot table.

strAppPath = objSpssApp.GetSPSSPath

strFileName = strAppPath & "employee data.sav"

Set objDataDoc = objSpssApp.OpenDataDoc(strFileName)

strCommand = strCommand + "REGRESSION"

strCommand = strCommand + " /MISSING LISTWISE"

strCommand = strCommand + " /STATISTICS COEFF OUTS R ANOVA"

strCommand = strCommand + " /CRITERIA=PIN(.05) POUT(.10)"

strCommand = strCommand + " /NOORIGIN"

strCommand = strCommand + " /DEPENDENT salary"

strCommand = strCommand + " /METHOD=ENTER salbegin ."

objSpssApp.ExecuteCommands strCommand, True

'Get the Viewer window and make it visible.

Set objOutputDoc = objSpssApp.GetDesignatedOutputDoc

objOutputDoc.Visible = True

'Get Output Items collection

Set objOutputItems = objOutputDoc.Items()

Dim intItemCount As Integer 'Number of output items.

Dim intItemType As Integer 'Output item type.

Dim strLabel As String'Output item label.

'Loop through the output items, checking type and label.

```

' We'll look for ANOVA pivot tables.
' If type and label match, activate the item.
intItemCount = objOutputItems.Count() 'Get the number of items.
For Index = 0 To intItemCount - 1
    Set objOutputItem = objOutputItems.GetItem(Index)
    intItemType = objOutputItem.SPSSType() 'Get the item type.
    strLabel = objOutputItem.Label 'Get the item label.
    If intItemType = SPSSPivot And strLabel = "ANOVA" Then
        Set objPivotTable = objOutputItem.Activate()
    Exit For
End If
Next Index

End Sub

```

Figure 3-20

Viewer displays activated pivot table

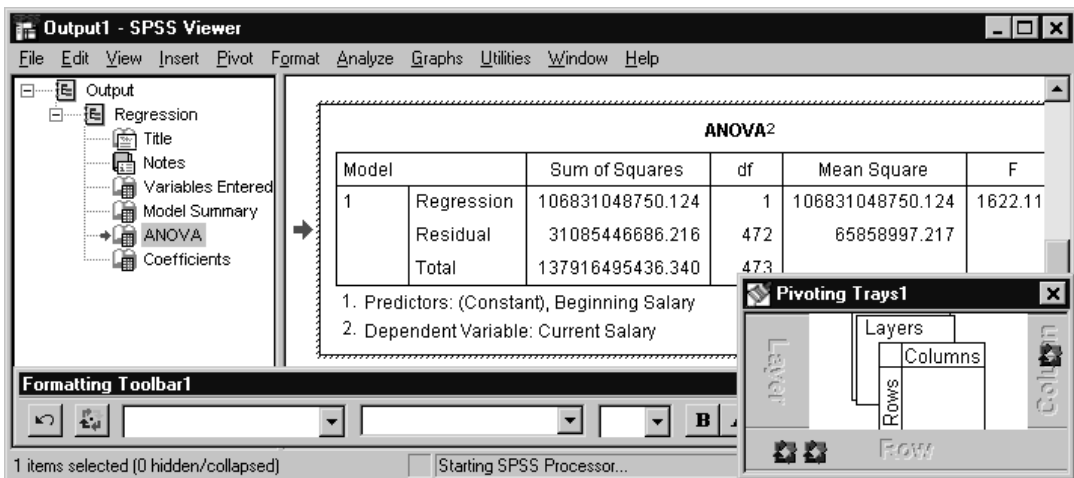


Chart Object (ISpssChart)

The Chart object is a chart contained in the Output Document object. Use this object to export a single chart. To export a number of charts in the same format, use the ExportChart or ExportDocument method of the Output Document object.

To get a Chart object, declare an object variable as ISpssChart and set it to the ActivateChart method of the Output Item object. You need to deactivate the item when you have finished manipulating the Chart object.

Figure 3-21 shows an example that starts SPSS, opens a data file, and creates and exports a chart.

Figure 3-21
Export chart

```
'Example 9: Export a chart
Private Sub cmdExample9_Click()

'Declare variables.
Dim objOutputDoc As ISpssOutputDoc
Dim objOutputItems As ISpssItems
Dim objOutputItem As ISpssItem
Dim objSPSSChart As ISpssChart
Dim strCommand As String

'Start SPSS.
Set objSpssApp = CreateObject("SPSS.Application")

'Open a data file and create some output so we can export it.
strAppPath = objSpssApp.GetSPSSPath
strFileName = strAppPath & "employee data.sav"
Set objDataDoc = objSpssApp.OpenDataDoc(strFileName)
strCommand = strCommand + "GRAPH"
strCommand = strCommand + " /BAR(SIMPLE)=COUNT BY gender"
strCommand = strCommand + " /MISSING=REPORT."
objSpssApp.ExecuteCommands strCommand, True

'Get the Viewer window and make it visible.
Set objOutputDoc = objSpssApp.GetDesignatedOutputDoc
objOutputDoc.Visible = True

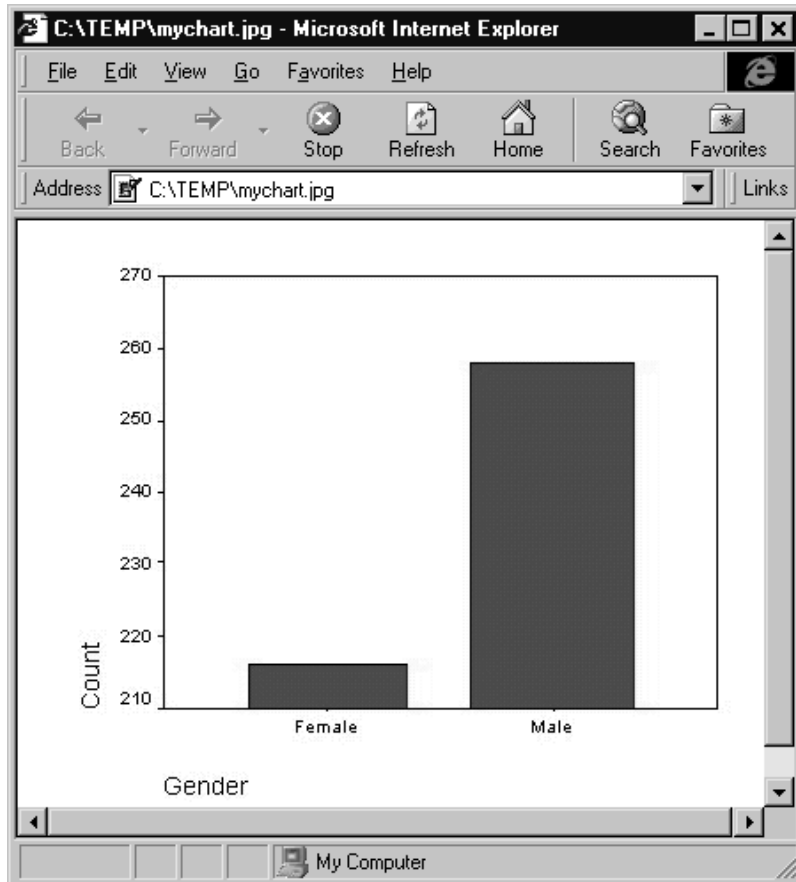
'Get Output Items collection
Set objOutputItems = objOutputDoc.Items()
Dim intItemCount As Integer           'Number of output items.
Dim intItemType As Integer           'Output item type.
```

```
'Loop through the output items, checking type.  
' We'll look for charts.  
' If type matches, activate the item and export it as a JPEG file.  
intItemCount = objOutputItems.Count()  
  For Index = 0 To intItemCount - 1  
    Set objOutputItem = objOutputItems.GetItem(Index)  
    intItemType = objOutputItem.SPSSType()  
    If intItemType = SPSSChart Then  
      Set objSPSSChart = objOutputItem.ActivateChart  
      objSPSSChart.ExportChart "c:\temp\mychart.jpg", "JPEG File"  
    Exit For  
  End If  
Next Index  
  
'Tip: Check c:\temp for the file "mychart.jpg" to confirm the example worked.  
' If you want to look at the file and have an application that is associated with jpg files,  
' double-click it (e.g., Internet Explorer).  
  
End Sub
```

Note: It is not possible to modify charts using OLE Automation. To control the appearance of charts produced by automation, specify a chart template when creating the chart. For more information about chart templates, see the *SPSS Base User's Guide* and the online Help.

Figure 3-22

Microsoft Internet Explorer displays exported chart



Graph Object (ISpssIGraph)

The Graph object (ISpssIGraph) is an interactive graph contained in the Output Item object (ISpssItem). Use this object to retrieve and modify other objects associated with the graph and to export an interactive graph.

To get a Graph object, declare an object variable as ISpssIGraph and set it to the GetIGraphOleObject method of the Output Item object.

Figure 3-23 shows an example that starts SPSS, opens a data file, creates an interactive graph, activates it, and turns on the value label display. Figure 3-24 shows the result of running the example—the interactive graph is activated and displays value labels.

Figure 3-23

Edit interactive graph

'Example 11: Modify an interactive graph

Private Sub cmdExample11_Click()

'Declare variables.

Dim objSpssApp As ISpssApp

Dim objDataDoc As ISpssDataDoc

Dim objOutputDoc As ISpssOutputDoc

Dim objOutputItems As ISpssItems

Dim objOutputItem As ISpssItem

Dim objSpssIGraph As ISpssIGraph

Dim objBarElement As ISpssIGraphBarElement

Dim strCommand As String

'Start SPSS.

Set objSpssApp = CreateObject("SPSS.Application")

'Open a data file and create some output so we can get an interactive graph object.

strAppPath = objSpssApp.GetSPSSPath

strFileName = strAppPath & "employee data.sav"

Set objDataDoc = objSpssApp.OpenDataDoc(strFileName)

strCommand = strCommand + "IGRAPH"

strCommand = strCommand + " /VIEWNAME='Bar Chart'"

strCommand = strCommand + " /X1 = VAR(jobcat) TYPE = CATEGORICAL /Y = \$count"

strCommand = strCommand + " /COORDINATE = VERTICAL"

strCommand = strCommand + " /X1LENGTH=3.0 /YLENGTH=3.0 /X2LENGTH=3.0

/CHARTLOOK='NONE'"

strCommand = strCommand + " /CATORDER VAR(jobcat) (ASCENDING VALUES OMITEMPTY)"

strCommand = strCommand + " /BAR KEY=ON SHAPE = RECTANGLE BASELINE = AUTO."

objSpssApp.ExecuteCommands strCommand, True

'Get the Viewer window and make it visible so we can see the graph after we change it.

Set objOutputDoc = objSpssApp.GetDesignatedOutputDoc

objOutputDoc.Visible = True

'Get Output Items collection

```

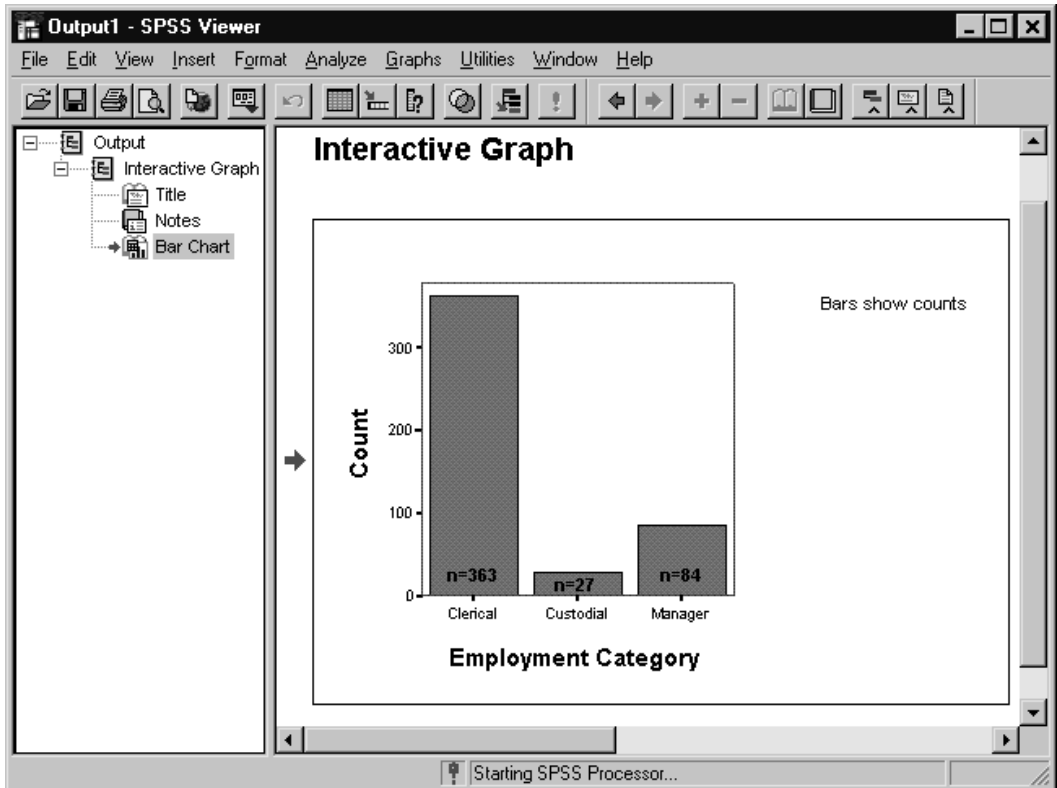
Set objOutputItems = objOutputDoc.Items()
Dim intItemCount As Integer           'Number of output items.
Dim intItemType As Integer           'Output item type.
'Dim strTitle As String               'The title text we want to find and change.

'Loop through the output items, checking type.
' We'll look for interactive graphs.
' If type matches, activate the item, turn on count labels and deactivate it.
intItemCount = objOutputItems.Count
For Index = 0 To intItemCount - 1
    Set objOutputItem = objOutputItems.GetItem(Index)
    intItemType = objOutputItem.SPSSType
    If intItemType = SPSSIGraph Then
        Set objSpssIGraph = objOutputItem.Activate()
        Set objBarElement = objSpssIGraph.GetElement(SpssIGraphBar)
        objBarElement.GetCountLabel.Show = True
        objSpssIGraph.Redraw           'Always redraw to see your change.
        objOutputItem.Deactivate
    Exit For
End If
Next

End Sub

```

Figure 3-24
Viewer displays edited interactive graph



Text Object (ISpssRTF)

The Text object is an RTF text editor contained in the Output Document object. You can access and manipulate SPSS text output, including warnings, logs, and titles, using this object.

To get a Text object, declare an object variable as `ISpssRtf` and set it to the return value of the `ActivateText` method of the Output Item object. You need to deactivate the item when you are finished manipulating the text object.

Figure 3-25 shows an example that starts SPSS, opens a data file, creates output items, gets a text object, and changes the text. Figure 3-26 shows the modified text in the Viewer.

Figure 3-25
Edit Text object

'Example 10: Modify a text object.
Private Sub cmdExample10_Click()

'Declare variables.
Dim objSpssApp As ISpssApp
Dim objDataDoc As ISpssDataDoc
Dim objOutputDoc As ISpssOutputDoc
Dim objOutputItems As ISpssItems
Dim objOutputItem As ISpssItem
Dim objSpssText As ISpssrtf
Dim strCommand As String

'Start SPSS.
Set objSpssApp = CreateObject("SPSS.Application")

'Open a data file and create some output so we can get a text object.
strAppPath = objSpssApp.GetSPSSPath
strFileName = strAppPath & "employee data.sav"
Set objDataDoc = objSpssApp.OpenDataDoc(strFileName)
strCommand = strCommand + "CROSSTABS"
strCommand = strCommand + " /TABLES=gender BY jobcat"
strCommand = strCommand + " /FORMAT= AVALUE TABLES"
strCommand = strCommand + " /CELLS= COUNT ."
objSpssApp.ExecuteCommands strCommand, True

'Get the Viewer window and make it visible so we can see the title after we change it.
Set objOutputDoc = objSpssApp.GetDesignatedOutputDoc
objOutputDoc.Visible = True

'Get Output Items collection
Set objOutputItems = objOutputDoc.Items()

Dim intItemCount As Integer 'Number of output items.
Dim intItemType As Integer 'Output item type.
Dim strItemTitle As String 'The title text we want to find and change.

'Loop through the output items, checking type and text.
' We'll look for titles with the text "Crosstabs" (titles are text objects).
' If type and text match, activate the item, change the text and deactivate it.
intItemCount = objOutputItems.Count
For Index = 0 To intItemCount - 1
 Set objOutputItem = objOutputItems.GetItem(Index)

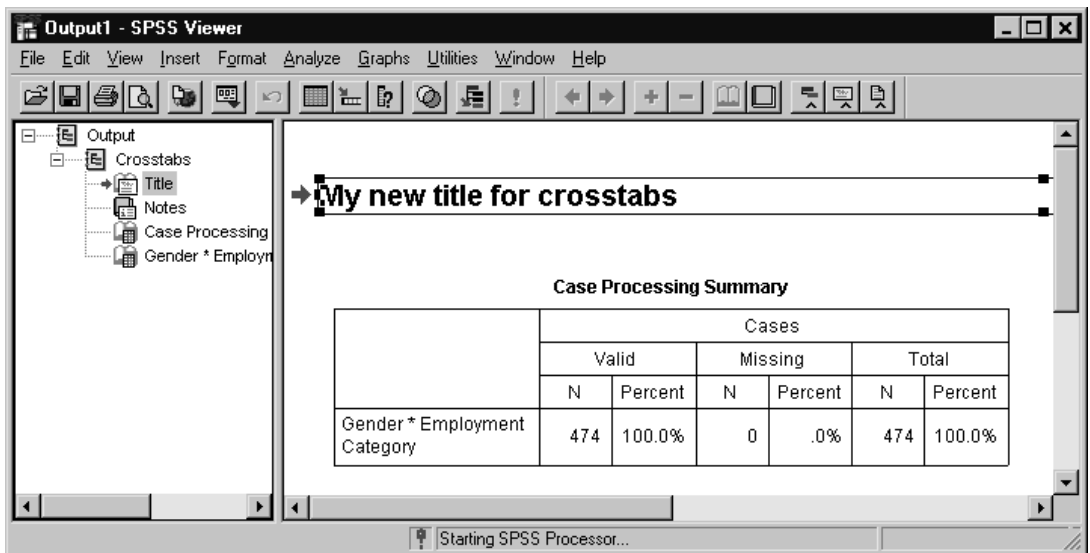
```

intltemType = objOutputItem.SPSSType
If intltemType = SPSSTitle Then
    Set objSpssText = objOutputItem.ActivateText
    strltemTitle = objSpssText.Text
    If strltemTitle = "Crosstabs" Then
        objSpssText.Text = "My new title for crosstabs"
        objOutputItem.Deactivate
    Exit For
End If
End If
Next
End Sub

```

Figure 3-26

Viewer displays the edited title text



Properties and Methods

Most SPSS objects have properties that you can use to query the attributes of an object and methods to manipulate the object. Table 3-2 on p. 69 provides an overview of available SPSS methods and properties for high-level SPSS objects.

Properties. Properties set or return attributes of objects. Some properties return another object, as discussed above; other properties are attributes, such as color or width. For example, objects of the Pivot Table class have a property called `CaptionText`. To set the caption at the bottom of a pivot table (`objPivotTable`) to *My Results*, type the following statement:

```
objPivotTable.CaptionText = "My Results"
```

When a property appears on the left side of an equals sign (as in the above example), you are **setting** its value. When a property appears on the right side, you are **getting**, or reading, its value. For example, to get the caption of the pivot table and save it in a variable:

```
strFontName = objPivotTable.CaptionText
```

Methods. Methods perform actions on objects, such as selecting all of the elements in a table:

```
objPivotTable.SelectTable
```

or removing a selection:

```
objPivotTable.ClearSelection
```

Like properties, some methods return another object. For example, the `GetDesignatedOutputDoc` method returns the designated output document:

```
Set objOutputDoc = objSpssApp.GetDesignatedOutputDoc
```

Table 3-2

Example properties and methods for high-level OLE Automation objects

Object	Properties	Methods
ISpssApp	Documents Options SpssInfo	ExecuteCommands GetDesignatedOutputDoc NewDataDoc OpenDataDoc Quit
ISpssOptions	DisplayCommands OutputBeep WarningsVisible	none
ISpssInfo	NumVariables VarType	GetSelectedVariables
ISpssDocuments	DataDocCount	GetDataDoc
ISpssDataDoc	Modified PromptToSave Visible	Copy GetNumberOfCases GetVariables SelectCells SaveAs
ISpssSyntaxDoc	Designated PromptToSave Text	Close PrintDoc Run SaveAs
ISpssOutputDoc	PrintOptions SplitterPosition Visible	ClearSelection ExportCharts InsertTitle SelectAllMaps
ISpssDraftDoc	Height Width WindowState	Close GetDocumentPath PrintRange
ISpssItems	Count	GetItem
PivotTable	BackgroundColor TableLook TextStyle	CreateChart HideFootnotes SelectCaption ShowAll
ISpssIGraph	CoordinateSystem Elements Title	DeleteTitle GetElement Redraw

Table 3-2

Example properties and methods for high-level OLE Automation objects

Object	Properties	Methods
ISpssRtf	none	RtfText
ISpssChart	none	ExportChart SetSize
Map	none	none
TreeModelControl	none	ExportImage SetSize

Note: Table 3-2 doesn't list all of the available properties and methods. The online Help for SPSS OLE Automation documents all properties and methods of SPSS objects. "Object Browser and Online Help" below describes how to access the online Help.

SPSS Type Libraries

The complete set of object classes (or object types) and the properties and methods associated with each are described in the SPSS type libraries. A **type library** is a file that contains OLE Automation standard descriptions of exposed objects, along with the properties and methods associated with each.

SPSS provides four type libraries:

SPSS type library (*spsswin.tlb*). Includes the Application object, Options object, File Information object, the complete Documents Collection, the Items Collection, the Chart object, and Maps.

PivotTable type library (*spsspvt.tlb*). Includes the Pivot Table object and all of the objects that reside within it.

Graphics Editor OLE control (*spssgctl.tlb*). Includes the Interactive Graphs object and all of the objects that reside within it.

RTF type library (*spssrtf.tlb*). Includes the RTF text object.

SPSS type libraries are automatically registered in the Windows registration database the first time you run SPSS after you have properly installed it.

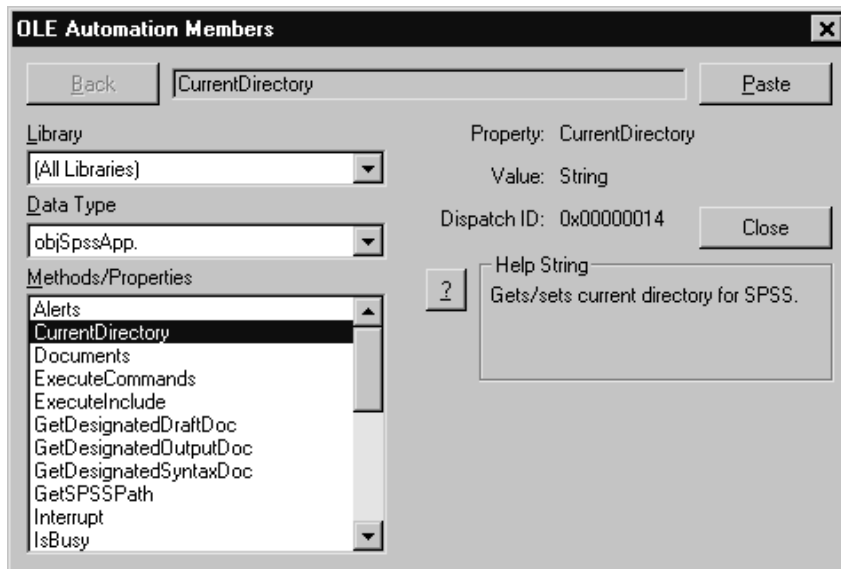
Some programming environments, such as Visual Basic, require you to explicitly add the type libraries to the development environment before you can access them. If

you must do so, make sure that you add all four type libraries. See your programming language's documentation for specific instructions.

Object Browser and Online Help

Most development environments, including Visual Basic, C++, and the SPSS Script Editor, provide an object browser facility that allows you to view and use the type libraries. You can browse all SPSS objects, their properties and methods, and the predefined constants. You can also paste the syntax of selected properties and methods directly into your code, and you can access context-sensitive online Help and code examples.

Figure 3-27
SPSS Script Editor object browser



To view objects and get Help in most object browsers:

- ▶ Select the type library that contains the objects of interest.
- ▶ Select an object class to display the methods and properties for that class.

- ▶ Select individual properties and methods to paste them into your code, or press F1 to access context-sensitive Help.

To access OLE Automation object tree Help from the SPSS product:

- ▶ Launch SPSS. From the Windows Start menu choose:
 - Programs
 - SPSS for Windows
 - SPSS for Windows
- ▶ Access a script window. For example, from the menus choose:
 - File
 - New
 - Script
- ▶ Open the Help. From the script window menus choose:
 - Help
 - Objects
- ▶ Click the object of interest on the tree to access Help, example code, and a complete list of that object's properties and methods.

Scripting Quickstart

This chapter introduces you to using the SPSS scripting facility. The chapter begins with an overview of scripting features and concludes with step-by-step examples. If you are already familiar with SPSS for Windows and SPSS OLE Automation, this is a good place to start learning about writing a script. If you're unfamiliar with SPSS for Windows, read Chapter 2 first. If you're unfamiliar with SPSS OLE Automation, read Chapter 3 first.

What Is the SPSS Scripting Facility?

Scripts work by manipulating SPSS OLE Automation objects by using their properties and methods. Scripts are created and edited in the SPSS script window. The SPSS scripting facility is introduced on p. 22 in Chapter 2.

Scripting versus OLE Automation Applications

Scripting uses the same SPSS OLE Automation object, properties, and methods that are described in Chapter 3. To write scripts, you first need to familiarize yourself with the object model hierarchy shown in Figure 3-1.

The main distinction between writing a script and writing an OLE Automation application is that the script runs within the SPSS application—it isn't a separate application. The code that you write for a script can be essentially the same as the code you write for an OLE Automation application except that when you write a script, you do not need to declare the SPSS Application object (because SPSS is already running).

Write scripts when you want to control SPSS from within an SPSS session—for example, when you want to:

- Customize SPSS output.
- Add a feature to SPSS.

Write OLE Automation applications when you want to control SPSS from another application—for example, when you want to:

- Add SPSS functionality to another application.
- Write an application with a completely alternate user interface to SPSS.

Script Window Features

The script window is a fully featured programming environment that uses the Sax Basic language and includes a dialog box editor, object browser, debugging features, and context-sensitive Help. (Figure 2-10 in Chapter 2 shows the script window.)

Pasting syntax. Many SPSS analysis and data management dialog boxes include a Paste button that generates command syntax for the current procedure. If you open a dialog box from a script window, SPSS will paste the command syntax and the code required to run it. See “Creating Command Syntax” on p. 16 in Chapter 2 for step-by-step instructions.

Command syntax. The SCRIPT syntax command can be used to pass a parameter from a syntax file to a script. For example, you can pass a filename. See the online Help topic ScriptParameter Method for details and an example (choose Help on the script window menu, then choose Object, and then look at the Index). “Writing an Original Script” on p. 82 includes an example use of the SCRIPT command.

Customized descriptions. You can add a description to your script, which is displayed in the Run Script and Use Starter Script dialog boxes. Add a comment on the first line of the script that starts with ‘Begin Description, followed by your comments (one or more lines), followed by ‘End Description.

Procedure display. As you move the cursor, the name of the current procedure is displayed at the top of the window.

Color cues. Terms colored blue are reserved words in Sax Basic (for example Sub, End Sub, and Dim). Objects, properties, and methods are displayed in magenta. Comments are displayed in green.

Dialog boxes. The SPSS scripting facility supports custom dialog boxes. Use these when you want to solicit input from a user about how the script should run or when you want to customize SPSS behavior and hide that fact from the user. The script window has a UserDialog Editor that provides a way to define the dialog box. Access the dialog editor from the Script menu.

Debugging. The Debug menu allows you to step through your code, executing one line or subroutine at a time and viewing the result. You can also insert a break point in the script to pause the execution at the line that contains the break point.

Object browser and Help. Press F2 to display the object browser, which displays SPSS objects, properties, and methods and affords access to the online Help. The object browser also allows you to paste the correct code for selected properties and methods directly into your script.

Types of Scripts

SPSS includes many sample scripts that are installed with the product in the `\Scripts` directory. In addition to ordinary sample scripts, which you can run to get the results you want, scripts for special purposes include the following:

Starter. Starter scripts supply code for one or more common sequences of tasks. They include comments with hints on how to customize the script to your particular needs. Starter scripts are installed in the SPSS installation directory in `\Scripts\Starter`. SPSS automatically prompts you to open a starter script when you create a new script window (see “Modifying a Starter Script” on p. 77 for an example). Of course, you can use any script as a starter script, although it probably won’t be as easy to customize. Simply open the script, customize it, and save it with a different filename.

Global. A global procedures script is a library of procedures that can be called by other scripts. When you open a script window, the global file is loaded automatically and its procedures are available to your script. To view the global script, click the tab labeled 2. The tab is located on the left side of the script window. You can add your own frequently used procedures to the default global file (`\Scripts\global.sbs`), or you can specify a different global file in the Options dialog box (on the Edit menu).

Autoscript. An autoscript runs automatically when it is triggered by the creation of a specific type of output from a specific procedure. For example, there is an autoscript that runs whenever a Correlations table is produced by the Bivariate Correlations procedure. The script automatically removes the upper diagonal of the table as soon as

it appears in the Viewer. You can add your own autoscripts to the default autoscript file (`\Scripts\Autoscript.sbs`), or you can specify a different autoscript file in the Options dialog box (on the Edit menu). For an example, see “Adding an Autoscript” on p. 79.

How Do I Use Scripting?

When using the SPSS Scripting Facility, you:

- ▶ Decide what you want your script to do.
- ▶ Write the script code.
- ▶ Run the script.

Deciding What You Want Your Script to Do

You can use scripting to do most of the things you can do with OLE Automation, and that includes most of the things you do when running SPSS interactively. For examples of tasks, see the list on p. 30 in Chapter 3.

Because scripts run within SPSS, one of your main decisions is what to do in the graphical user interface versus what tasks to code into the script. Working in the user interface allows maximum flexibility and user control—it is best for analytic tasks. Scripting allows a sequence of actions to be repeated exactly—it is best for repetitive, predictable tasks.

Look at the example scripts distributed with SPSS for Windows to get ideas about what types of tasks can be scripted. Example scripts (*.sbs) are in the scripts folder in the SPSS installation directory and in *\SPSS Products and Services\SPSS Script eXchange* on the SPSS for Windows CD-ROM.

Writing Script Code

Start writing script code by modifying starter scripts (for an example, see “Modifying a Starter Script” on p. 77).

Before writing your own scripts, use the graphical user interface to perform the tasks you want to script. At each step, think about the OLE Automation objects,

methods, and properties that correspond to what you are doing. For more information and an example, see p. 30 in Chapter 3.

Running Scripts

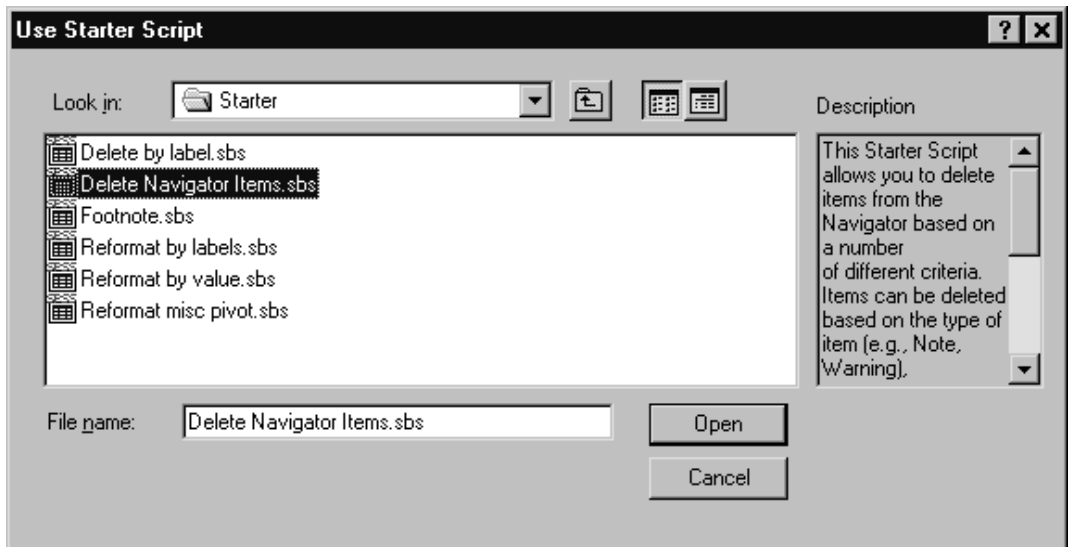
Autoscripts run automatically. Other scripts are run from the Utilities menu (see Figure 4-3).

Examples

Modifying a Starter Script

The following steps show an example of how to work with a starter script. For this example, we want to delete all Notes tables from an SPSS output file. We could manually select and delete each unwanted table, or we could use a script.

Figure 4-1
Choosing a starter script



- **Launch SPSS.** From the Windows Start menu choose:

Programs
SPSS for Windows
SPSS for Windows

- **Create a script window.** From the SPSS menus choose:

File
New
Script

- **Choose a starter script.** For example, select *Delete Navigator Items.sbs*, which has a promising description, since we want to delete all Notes tables from the output (see Figure 4-1).
- **Review the script.** Read the comments and look at the code. Decide if the script supports what you want to do and, if it does, how you want to modify it to suit your needs.
- **Modify the script.** For example, make the necessary changes to delete all Notes tables. For each line listed below, remove the comment character (') from the beginning of the line. Removing the comment character causes the line of code to be executed when the script is run.

Line of Code	Effect
intTypeToDelete = SPSSNote	Causes Note table items to be deleted when the DeleteByAll procedure is called.
Call DeleteAllByType(intTypeToDelete)	Calls a procedure to delete all Notes tables and passes it the type of item to delete.
Call DeleteSelectedItems	Calls a procedure to delete selected items.

A copy of the modified script is on the SPSS for Windows CD-ROM in
`\SPSS\Developer\Programs\SPSS Script\Modify starter script.sbs`.

- **Save the script.** From the menus choose:

File
Save As...

Type a name and browse to a location. For example, save the script as
`C:\SPSS\Scripts\Delete all notes.sbs`.

- **Open an output file.** This script will work on any valid SPSS output file (*.spo*) that contains Notes tables. From the SPSS menus choose:

File
 Open
 Output...

Navigate to the location of the output file and select it. A sample output file with two Notes tables is included on the SPSS for Windows CD-ROM in
`\SPSS\Developer\Programs\SPSS Script\Modify starter script.spo`.

- **Run the script.** From the SPSS menus choose:

Utilities
 Run Script...

Navigate to the location where you saved the script and select it. For example, select `C:\SPSS\Scripts\Delete all notes.sbs`. The script runs and removes all Notes tables. (If you ran the script on *Modify starter script.spo*, there were seven items before you ran the script and five items afterwards—the two Notes tables were deleted).

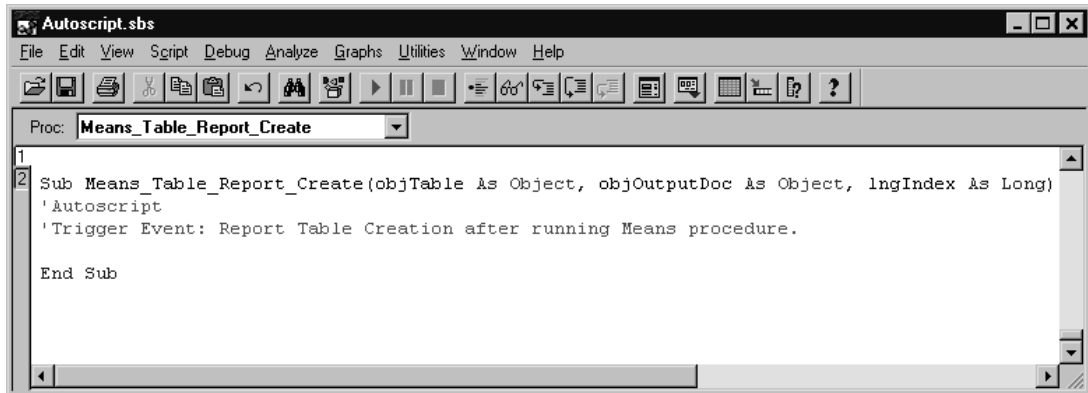
Since this script can be run on any valid SPSS output file that contains notes, you now have a quick and easy way to accomplish what would otherwise be a boring and repetitive task.

Adding an Autoscript

The following steps show an example of how to add a new autoscript procedure to the default autoscript file. For this example, we want to automatically make the font of the row totals bold italic whenever SPSS produces a Means table.

Figure 4-2

Autoscript file with a newly created autoscript ready for your code



- **Launch SPSS.** From the Windows Start menu choose:
 - Programs
 - SPSS for Windows
 - SPSS for Windows
- **Open a data file.** For example, open the employee data file. From the SPSS menus choose:
 - File
 - Open
 - Data...
 Select *employee data.sav*.
- **Run the procedure that creates the type of output item you want to customize.** For example, run the Means procedure. From the menus choose:
 - Analyze
 - Compare Means
 - Means...
 Move *Current Salary* to the Dependent list and *Employment Category* to the Independent list.
- **Select the output item you want to customize.** In the Viewer, scroll to the table titled *Report* and right-click to open the shortcut menu.

- **Create a new Autoscript.** Choose Create/Edit Autoscript from the shortcut menu. This automatically opens *autoscript.sbs* in a script window and inserts several lines of code:

Comment. At the beginning of the file:

```
' enabled Means_Table_Report_Create
```

Autoscript procedure for Means table creation. At the end of the file (see Figure 4-2):

```
Sub Means_Table_Report_Create(objTable As Object, objOutputDoc As Object,
    lngIndex As Long)
```

```
'Autoscript
```

```
'Trigger Event: Report Table Creation after running Means procedure.
```

```
End Sub
```

- **Add the necessary code to accomplish the customizing you want.** Insert your code between the 'Trigger Event comment and End Sub. In this case, we make the row totals bold italic. We'll use a procedure that is already in the autoscript file, *SelectRowLabelsAndData*, to find and select the cells we want to change. The completed code is:

```
Sub Means_Table_Report_Create(objTable As Object, objOutputDoc As Object,
    lngIndex As Long)
```

```
'Autoscript
```

```
'Trigger Event: Report Table Creation after running Means procedure.
```

```
'Your inserted code begins here.
```

```
'Declare a variable to keep track of what cells are selected.
```

```
Dim bolSelection As Boolean
```

```
'Call a procedure, SelectRowLabelsAndData, that selects the row TOTAL.
```

```
'The objTable parameter is the Means table that has been created.
```

```
' It is passed to the procedure as objPivotTable.
```

```
'The cTOTAL paramter, defined above as the string 'Total'
```

```
' is passed to the procedure as strSearchString.
```

```
Call SelectRowLabelsAndData(objTable, cTOTAL, bolSelection)
```

```
'When the procedure returns a cell selection, turn it bold and italic.
```

```
If bolSelection = True Then
```

```
    objTable.TextStyle = SpssTSBoldItalic
```

```
End If
```

```
'Your insterted code ends here
```

```
End Sub
```

The procedure `SelectRowLabelsAndData` was already coded for us in the autoscript file. What we did in this example was to apply that procedure to the type of table that we wanted to customize—in this case, the Means table.

We found the available text styles for `objTable` by using the script window's object browser. Follow the steps on p. 71 in Chapter 3 to open the object browser. Browse the PivotTable data type and the `TextStyle` property. Click ? for a list of available style settings.

- **Save your changes.** From the menus choose:

- File
- Save

- **Run the Autoscript.** The script will run automatically each time you produce output with the Means procedure. Open a data file and from the menus choose:

- Analyze
- Compare Means
- Means...

- **To deactivate the new Autoscript.** From the menus choose:

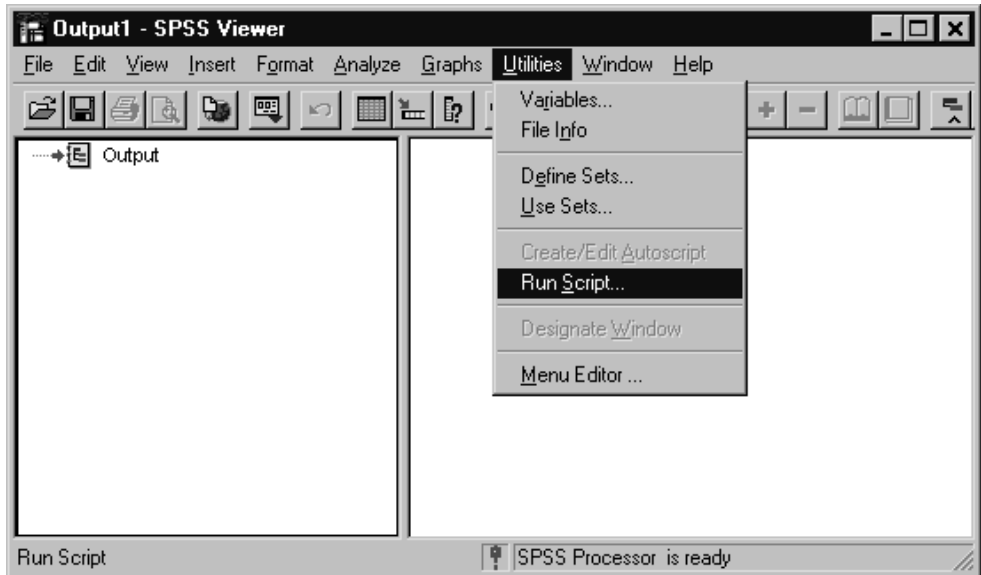
- Edit
- Options...

Click the Scripts tab, and then click `Means_Table_Report_Create` in the Autoscript subroutine status list to deselect it.

Writing an Original Script

The following steps show an example of how to write an original script. For this example, we want to open an output file (*.spo*), export the visible items as HTML (charts as JPEG), and close the file.

Figure 4-3
Running a script



- **Launch SPSS.** From the Windows Start menu choose:
 Programs
 SPSS for Windows
 SPSS for Windows
- **Perform the script scenario with the user interface.** Think about the objects, methods, and properties you are using. For this example, we'll open an output file, export it, and close the output file.

Open the output file. From the menus choose:

File
 Open
 Output...

Navigate to the location of the output file and select it. A sample output file is included on the SPSS for Windows CD-ROM in `SPSS\Developer\Programs\SPSS Script\Modify starter script.spo`.

The OLE automation equivalent is to use the `OpenOutputDoc` method on `objSpssApp` to open `ISpssOutputDoc`. `ISpssOutputDoc.Designated=True` was set when you opened the file—documents are automatically designated when they are opened.

Export. In the Viewer, from the menus choose:

```
File
  Export...
```

We want to export HTML for all visible objects, so select Output document from the Export list, the All Visible Objects radio button, and HTML file (*.htm) for the file type. Click Options and select file type JPEG *.JPG to export charts as JPEG.

The OLE Automation equivalent is to use the Export method on ISpssOutputDoc.

Close the output file. In the Viewer, from the menus choose:

```
File
  Close
```

The OLE automation equivalent is to use the Close method on ISpssOutputDoc.

- **Create a script window.** From the SPSS menus choose:

```
File
  New
    Script
```

The window is created with the code:

```
Sub Main
```

```
End Sub
```

You will be inserting your code between those two lines.

- **Write the code.** From the user interface scenario in the step above, you already know the basic steps, objects, methods, and properties. As you write the code, press F2 for the object browser and for online Help on SPSS objects. See Appendix C for code-writing conventions.

Declare variables and other housekeeping.

```
Sub Main
```

```
    'Begin description.
    'This example gets the specified output document,
    ' designates it, and exports all visible items to HTML.
    ' Charts are exported as JPEG files.
    'End description.
```

```
    'Declare variables.
    Dim objOutputDoc As ISpssOutputDoc
    Dim objSpssOptions As ISpssOptions
```

```
Dim strCurrentDir As String
Dim strOutputFileName As String
Dim strExportFileName As String

'Make sure charts are exported as JPEG.
Set objSPSSOptions = objSpssApp.Options
objSpssOptions.DefaultChartExportFormat = "JPEG File"

'Get the current directory to use as a default later.
strCurrentDir = objSpssApp.CurrentDirectory
```

Open the output file.

```
'Get the name of the output file to open. That's the file that you will export.
'You can get that from syntax via ScriptParameter.
strOutputFileName = objSpssApp.ScriptParameter(0)

'If the name wasn't passed with syntax, prompt the user.
If strOutputFileName = "" Then
strOutputFileName = GetFilePath$("spo", strCurrentDir, "Select Output File to
Export", 0)
End If

'Now that we have the name of the file to Export, open and designate it.
Set objOutputDoc = objSpssApp.OpenOutputDoc (strOutputFileName)
objOutputDoc.Designated = True
```

Export.

```
'Prompt the user for the name of the of the file to export to.
strExportFileName = GetFilePath$ ("Export.htm", "htm", , "Export File Name for "
+strOutputFileName, 3)
```

```
'Export it.
objOutputDoc.ExportDocument (SpssVisible, strExportFileName,
SpssFormatHtml, True)
```

Close the output file.

```
'Close it.
objOutputDoc.Close
End Sub
```

- **Save the script.** In the script window, from the menus choose:

```
File
  Save
```

And type a name for the script—for example, *Export output.sbs*.

- **Run the script.** You can run this script from the user interface or from syntax.

To run the script interactively and prompt the user for the output file to export, from the menus choose (see Figure 4-3):

```
Utilities
  Run Script...
```

Navigate to the location of your script file and select it. For example, select *Export output.sbs*. You will be prompted for an output file to export and for the export filename.

To run the script from syntax, use the `SCRIPT` command syntax and include the name of the output file to export as a script parameter. For example, the syntax:

```
SCRIPT 'c:\myscripts\Export output.sbs' ("myoutput.spo").
```

will open and export *myoutput.spo*.

A copy of this script is included on the SPSS for Windows CD-ROM in
`\SPSS\Developer\Programs\SPSS Script\Export output.sbs`.

Additional Examples

SPSS for Windows includes samples of code that illustrate various ways to use the SPSS developer's tools. This chapter provides a description of each sample program. You may find the examples useful when you design applications, and you can take the sample code and modify it to suit your needs. The code for all of the examples is located on your SPSS for Windows CD-ROM in `\SPSS\Developer`.

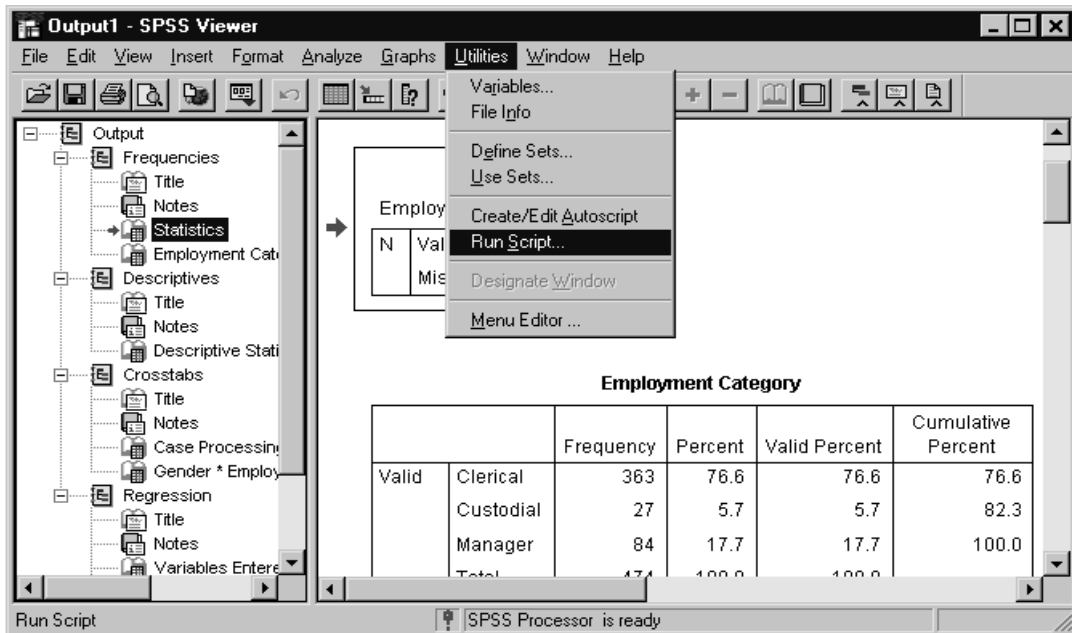
Notes:

- The examples in this chapter come from a variety of sources and are written with a variety of coding styles. They are intended only to illustrate the concepts involved in writing applications with the SPSS developer's tools; they do not contain all of the error checking and exception handling typical of finished applications.
- All Visual Basic examples were developed in version 4.0 and were resaved as version 6.0 projects.
- All Visual Basic examples assume that SPSS is not currently running. If you want to write an application that checks to see if SPSS is running, use the sample code on p. 41 in Chapter 3 as your starting point.

For more examples. Additional examples that use the SPSS scripting facility are available on the CD-ROM in `\Spss Products and Services\SPSS Script eXchange` and on the SPSS Web site at <http://www.spss.com/software/spss/scriptexchange>. Sample scripts are installed with the SPSS system in the `\Scripts` folder. The sample code for the Visual Basic application described in Chapter 3 is found in `\SPSS\Developer\Programs\OLE Quickstart\spssole.vbp`.

Edit All Pivot Tables

Figure 5-1
Run Script on Viewer



Description. This script finds each pivot table in an output document, activates it, and modifies it. The distributed example applies the AutoFit method to all tables to recalculate the cell size. The user can replace AutoFit with whatever pivot table editing method(s) he or she chooses.

Development tools. The SPSS scripting facility and OLE Automation.

Features. The program shows how scripting can be used to automate routine editing tasks.

Location. The program is located on the SPSS for Windows CD-ROM in
 \SPSS\Developer\Programs\SPSS Script\Edit all pivot tables.sbs.

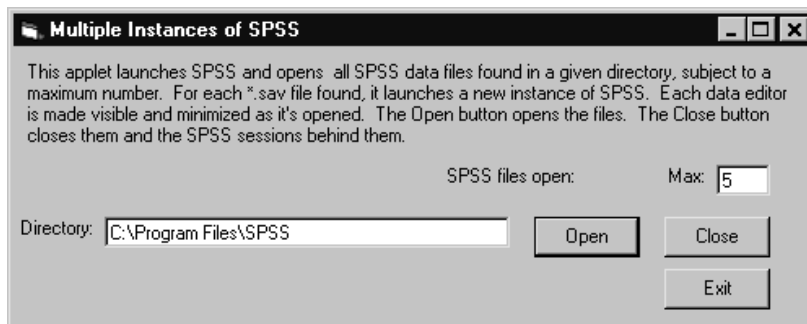
Requirements. SPSS must be running, and an output file with at least one pivot table must be open.

Running the application. To run the script from the Viewer:

- ▶ Click the output window you want to edit. It must contain at least one pivot table.
- ▶ From the menus choose:
 - Utilities
 - Run Script...
- ▶ Browse to the location of *Edit all pivot tables.sbs*, and select it.
- ▶ Click Run.

Manage Multiple Instances of SPSS

Figure 5-2
Multiple instances of SPSS example



Description. This example opens a dialog box that prompts the user to a data file location and the maximum number of files to open. After making these specifications, the user clicks Open. For each file in the location up to the maximum number, an instance of SPSS starts and opens the file. The Close button closes the files and exits all instances of SPSS.

Development tools. Visual Basic and SPSS OLE Automation.

Features. The program shows a basic example of how to launch and handle multiple instances of SPSS with OLE Automation. It exercises the Application (ISpssApp) and ISpssDataDoc objects.

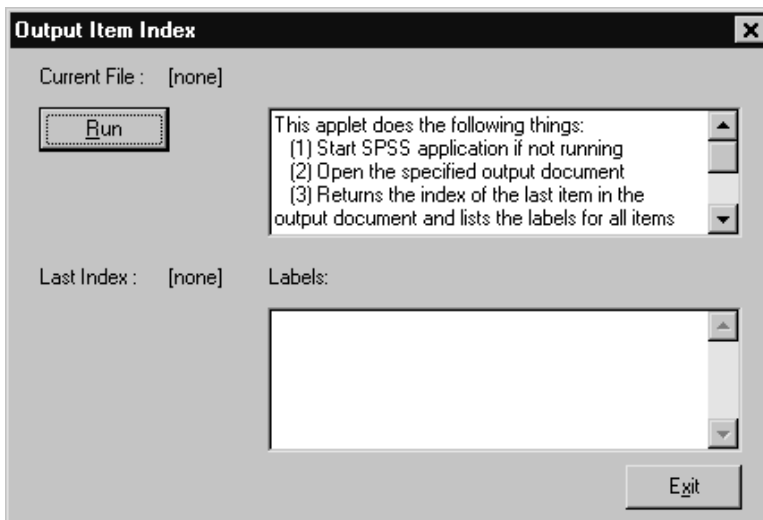
Location. The program is located on the SPSS for Windows CD-ROM in `\SPSS\Developer\Programs\Visual Basic\Load SPSS data files\LoadSPSS.vbp`.

Requirements. This example requires SPSS for Windows. It will open the SPSS data files (`.sav`) from any location.

Running the application. Run the `.vbp` file from within Visual Basic or execute `\SPSS\Developer\Programs\Visual Basic\Load SPSS data files\LoadSPSS.exe` from the SPSS for Windows CD-ROM.

Output Item Index

Figure 5-3
Output item index example



Description. This simple program starts SPSS, opens the output file that the user chooses, lists the number of items in the file, and prints the index number and label for each item. The program demonstrates the way that output items are indexed in the Viewer.

Development tools. Visual Basic and SPSS OLE Automation.

Features. The program shows how output items are indexed. It also shows how to raise a Windows common dialog box to open a specific type of SPSS file. It introduces the Output Items Collection (`ISpssItems`) and `ISpssItem`.

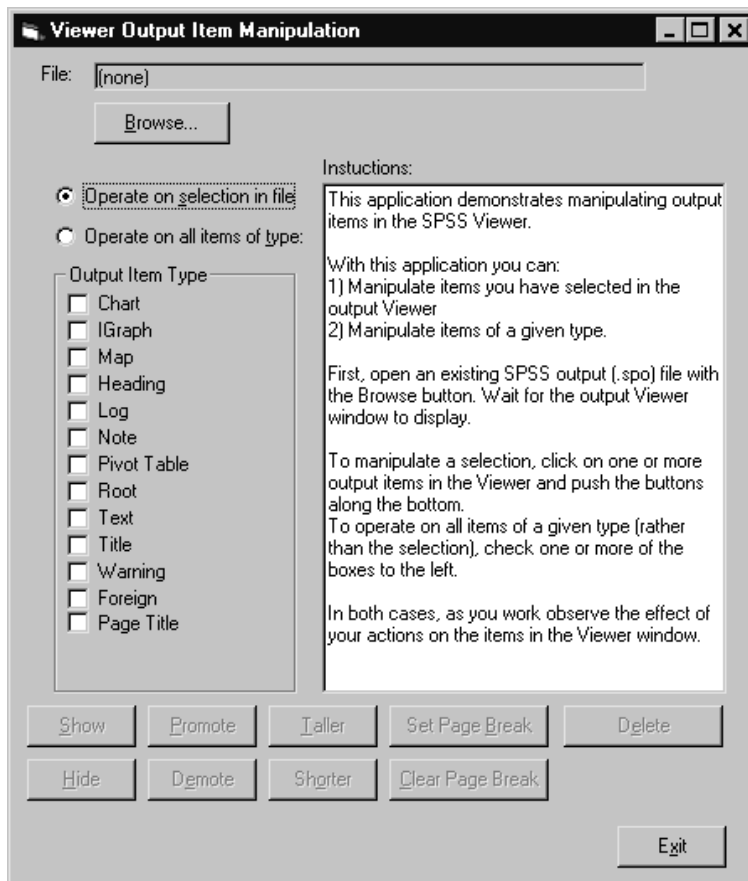
Location. The program is located on the SPSS for Windows CD-ROM in `\SPSS\Developer\Programs\Visual Basic\List index\lstindex.vbp`.

Requirements. This example requires SPSS for Windows. It works on any SPSS output file (.spo). A sample output file is included in the same directory as the example.

Running the application. You can run the .vbp file from within Visual Basic or execute `\SPSS\Developer\Programs\Visual Basic\List index\lstindex.exe` from the SPSS for Windows CD-ROM.

Manipulate Output Items

Figure 5-4
Viewer output item manipulation example



Description. This program manipulates items in Viewer (for example, it shows and hides items). The program acts either on the user's selection of items or on all items of a given type (for example, charts, pivot tables, and notes).

Development tools. Visual Basic and SPSS OLE Automation.

Features. The program illustrates how to use OLE Automation to navigate through the outline tree in the Viewer, locate a specific type of output item, and use methods on output items. It exercises the `ISpssOutputDoc`, `ISpssItems`, `ISpssItem` objects, and the `SPSSType` property; it also exercises the `Delete`, `Visible`, `Selected`, `Promote`, `Demote`, `PageBreak`, and `Height` methods.

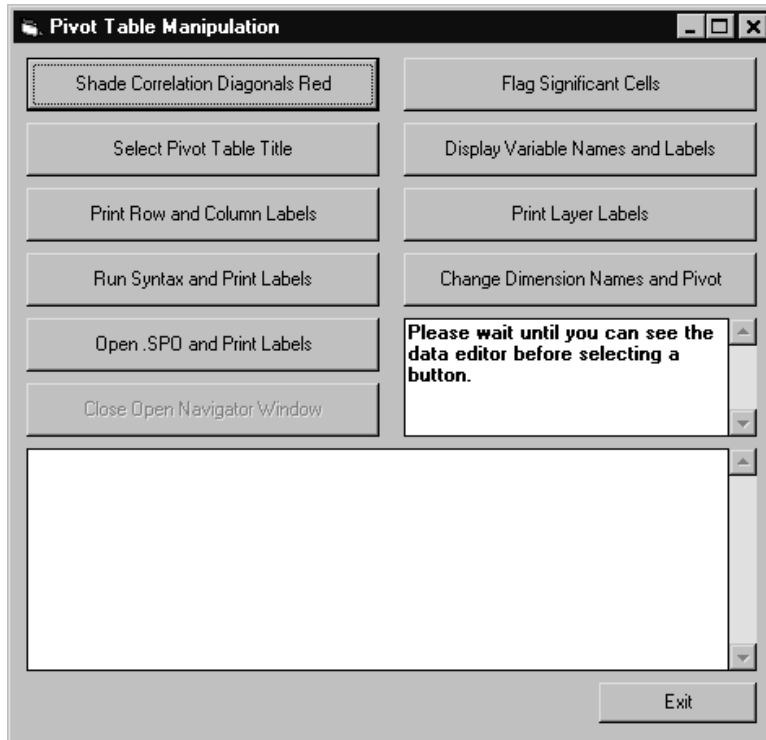
Location. The program is located on the SPSS for Windows CD-ROM in `\SPSS\Developer\Programs\Visual Basic\Manipulate nodes in the Viewer\navmanip.vbp`.

Requirements. This example requires SPSS for Windows and an SPSS output file. A sample output (`.spo`) file is included in the same directory as the example.

Running the application. You can run the `.vbp` file from within Visual Basic or execute `\SPSS\Developer\Programs\Visual Basic\Manipulate nodes in the Viewer\navmanip.exe` from the SPSS for Windows CD-ROM.

Pivot Table Manipulation

Figure 5-5
Pivot table manipulation example



Description. This program starts SPSS, opens the *Employee data.sav* file, and waits for the user to choose one of the pivot table manipulation buttons. The program demonstrates a number of techniques for manipulating pivot table output through automation. Available manipulations include applying color to table cells that meet specific criteria, printing table labels and cell values, and pivoting a table.

Development tools. Visual Basic and SPSS OLE Automation.

Features. The program shows a variety of ways to customize a pivot table to meet your needs, including modifications based on the values of table cells. It also shows how to open a data file and run a syntax file to produce output and how to open an existing output file. It introduces objects in the pivot table type library, including

ISpssPivotTable, ISpssDataCells, ISpssLabels, ISpssLayerLabels, ISpssPivotMgr, and ISpssDimension. In addition, it uses ISpssDataDoc, ISpssSyntaxDoc, ISpssItems, ISpssOutputDoc, ISpssItems, and ISpssItem.

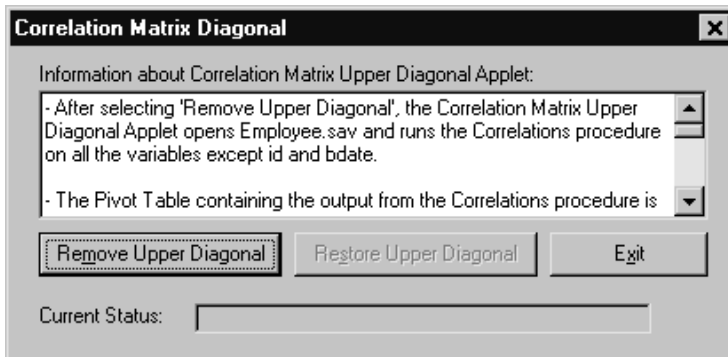
Location. The program is located on the SPSS for Windows CD-ROM in `\SPSS\Developer\Programs\Visual Basic\Pivot table exerciser\pvtxrsiz.vbp`.

Requirements. This example requires SPSS for Windows, the *Employee data.sav* data file that is distributed with the SPSS Base system, and several SPSS syntax and output files that are included in the same directory as the example.

Running the application. You can run the `.vbp` file from within Visual Basic or execute `\SPSS\Developer\Programs\Visual Basic\Pivot table exerciser\pvtxrsiz.exe` from the SPSS for Windows CD-ROM.

Correlation Matrix Diagonal

Figure 5-6
Correlation matrix diagonal example



Description. The correlation diagonal example opens a dialog box that prompts the user to remove the upper diagonal of a correlation matrix. When the user clicks Remove Upper Diagonal, the application starts SPSS, opens a data file, runs the SPSS Correlations procedure, and removes the upper diagonal from the resulting correlations table. A separate button restores the upper diagonal. The Exit button closes SPSS and the example application.

Development tools. Visual Basic and SPSS OLE Automation.

Features. The program shows a basic example of how SPSS output (in this case, a pivot table object from a specific SPSS procedure) can be edited with OLE Automation. In this example, specific cells in pivot table output are hidden. It exercises the `ISpssItems`, `ISpssItem`, `ISpssPivotTable`, and `ISpssDataCells` objects. This kind of output manipulation can also be done with the SPSS scripting facility (see subroutine `RemoveUpperDiag` and function `GetVarGroupSize` in *autoscript.sbs*, which is installed with SPSS in the `\Scripts` folder).

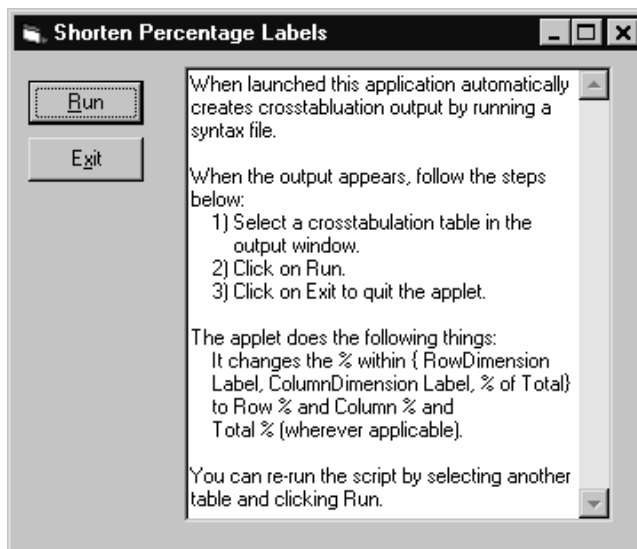
Location. The program is located on the SPSS for Windows CD-ROM in `\SPSS\Developer\Programs\Visual Basic\Correlations diagonal\corrdiag.vbp`.

Requirements. This example requires SPSS for Windows and the *Employee data.sav* data file that is distributed with the SPSS Base system.

Running the application. You can run the *.vbp* file from within Visual Basic or execute `\SPSS\Developer\Programs\Visual Basic\Correlations diagonal\corrdiag.exe` from the SPSS for Windows CD-ROM.

Shorten Percentage Labels in Crosstabulation

Figure 5-7
Shorten percentage labels example



Description. This program starts SPSS, opens the *Employee data.sav* file, and runs a syntax file, *Percent.sps*, that produces crosstabulation pivot tables. The user selects the table from the resulting output and clicks Run to shorten its labels to *row %*, *column %*, and *total %* where appropriate.

Development tools. Visual Basic and SPSS OLE Automation.

Features. The program shows another basic example of how the SPSS output from a specific SPSS procedure can be edited with OLE Automation. In this example, specific label text in the pivot table output is located and replaced. It also shows how to open a data file and run a syntax file to produce output. It exercises the *ISpssItems*, *ISpssItem*, *ISpssPivotTable*, and *ISpssLabels* objects and uses the *RowLabelArray* and *ColumnLabelArray* methods, which returns a labels object. This kind of output manipulation can also be done with the SPSS scripting facility (see subroutines *ChangeToPercent* and *SearchAndReplaceLabel* in *autoscript.sbs*, which is installed with SPSS in the *\Scripts* folder).

Location. The program is located on the SPSS for Windows CD-ROM in *\SPSS\Developer\Programs\Visual Basic\Shorten percentage labels in crosstabulation\percent.vbp*.

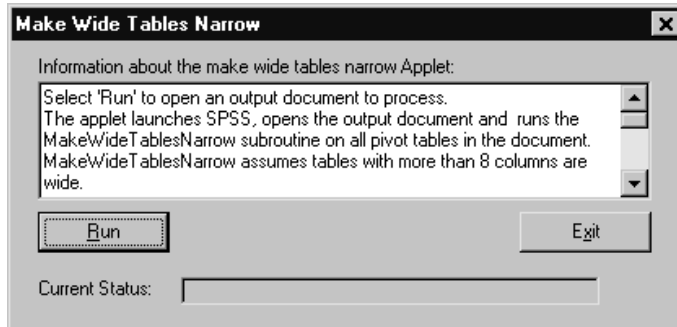
Requirements. This example requires SPSS for Windows, the *Employee data.sav* data file that is distributed with the SPSS Base system, and an SPSS syntax file, *Percent.sps*, which is included in the same directory as the example.

Running the application. You can run the *.vbp* file from within Visual Basic or execute *\SPSS\Developer\Programs\Visual Basic\Shorten percentage labels in crosstabulation\percent.exe* from the SPSS for Windows CD-ROM.

Make Wide Pivot Tables Narrow

Figure 5-8

Make wide pivot tables narrow example



Description. This example starts SPSS, opens the output file that the user chooses, and applies a number of algorithms to make wide pivot tables narrower. Only tables with more than eight columns are processed. When the user clicks Exit, the application prompts the user to save his or her changes.

Development tools. Visual Basic and SPSS OLE Automation.

Features. The program shows ways to automatically format tables using SPSS OLE Automation. It introduces the Options object (ISpssOptions) and exercises the ISpssOutputDoc, ISpssItems, ISpssItem, PivotTable, ISpssFootnotes, ISpssDataCells, and ISpssLabels objects.

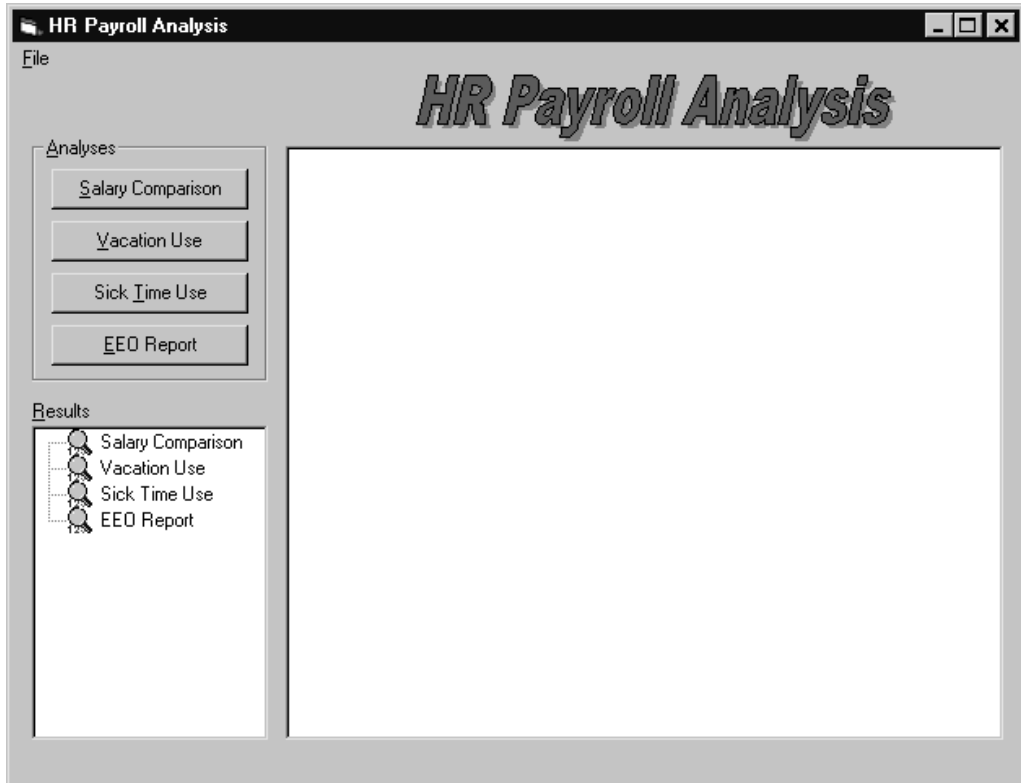
Location. The program is located on the SPSS for Windows CD-ROM in `\SPSS\Developer\Programs\Visual Basic\Make pivot tables narrow\narrow.vbp`.

Requirements. This example requires SPSS for Windows. It works on any SPSS output file (.spo). A sample output file with a wide table is included in the same directory as the example.

Running the application. You can run the .vbp file from within Visual Basic or execute `\SPSS\Developer\Programs\Visual Basic\Make pivot tables narrow\narrow.exe` from the SPSS for Windows CD-ROM.

Display, Print, and Export Reports

Figure 5-9
Payroll example



Description. This application populates a tree control with SPSS analyses based on employee data exported from a Ceridian human resources database. The user can generate tables and charts showing average salaries, vacation time, and sick time. The reports can be broken down by job title, department, marital status, gender, and ethnicity. User requests are translated into SPSS syntax and submitted to SPSS. The SPSS results are displayed in the application. Tables and charts generated by this application can be viewed on-screen, printed, and exported. Export formats include HTML and JPEG, which are suitable for posting on a company intranet.

Development tools. Visual Basic and SPSS OLE Automation.

Features. The program demonstrates how to use a Visual Basic user interface to collect user requests for analyses and to display the SPSS output. The application generates the appropriate SPSS syntax from the user interface, sends it to SPSS for processing, and receives the results from SPSS. It exercises the `ISpssApp`, `ISpssOptions`, `ISpssDataDoc`, `ISpssOutputDoc`, `ISpssItems`, and `ISpssItem`. It also accesses pivot tables with `PivotTable`, `ISpssLabels`, `ISpssPivotMgr`, and `ISpssDimension` and accesses charts with the `ExportChart` method on `ISpssChart`.

Location. The program is located on the SPSS for Windows CD-ROM in `\SPSS\Developer\Programs\Payroll\payroll.vbp`.

Requirements. This example requires SPSS for Windows, the data file *Ceridian.sav*, and several SPSS files that are included in the same directory as the example. Before you run the program, you must enable copying of objects as ActiveX controls from SPSS by running the file *objs-on.bat* in your SPSS directory.

Running the application. You can run the *.vbp* file from within Visual Basic or execute `\SPSS\Developer\Programs\Payroll\payroll.exe` from the SPSS for Windows CD-ROM.

Display a Report in Microsoft Word

Figure 5-10
Marketing quarterly expense report example

Quarterly Expense Report

Quarter of the Year: 1 Market: All

Remarks:

Provide your initial comments here.

☐ View Running Application

Start Load List Process View

This example uses OLE Automation with SPSS and Microsoft Word to automatically prepare a quarterly marketing expense report. In an actual application, the first two command buttons above this box would be incorporated with the form load event and the last two into the OK button, which is initially disabled for this demonstration.

OK Exit

Description. This application is written for the international office of a fictitious company to perform a quarterly analysis of marketing expenditures. It combines the analytic capability of SPSS with the presentation capability of Microsoft Word.

The program starts SPSS, opens the *mexpense.sav* file, populates the dialog box with values from the SPSS file, and starts Microsoft Word. The user selects the quarter and international region of interest and clicks View to see the SPSS tables and charts in a Microsoft Word document using a template (*mexpense.dot*).

Development tools. Visual Basic, Microsoft Word macro recorder (Visual Basic for Applications), and SPSS OLE Automation.

Features. The program demonstrates communication between software applications. It uses SPSS OLE Automation and a Microsoft Word macro to pass the results of an SPSS analysis to a predefined Microsoft Word form.

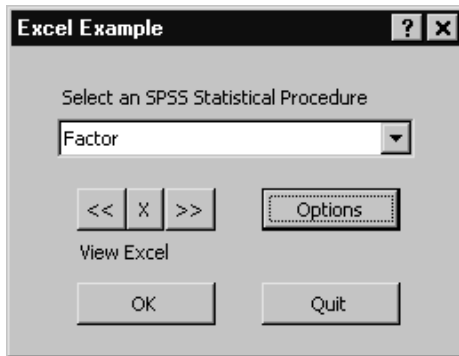
Location. The program is located on the SPSS for Windows CD-ROM in `\SPSS\Developer\Programs\Microsoft Word VBA\wdsample.vbp` and *wdsample.dot*.

Requirements. This example requires SPSS for Windows, Microsoft Word 97 or later, the *mexpense.sav* SPSS data file, and the *mexpense.dot* Microsoft Word template that are included in the same directory as the example.

Running the application. You can run the .vbp file from within Visual Basic or execute `\SPSS\Developer\Programs\Microsoft Word VBA\wdsample.exe` from the SPSS for Windows CD-ROM.

Analyze Excel Data and Display Reports in Excel

Figure 5-11
Microsoft Excel example



Description. This example uses SPSS to analyze data in a Microsoft Excel worksheet. The program exports the columns in the current worksheet as variables for analyses in SPSS and opens a dialog box that prompts the user to select a statistical procedure. The SPSS dialog box for the requested procedure is displayed, allowing the user to select variables for analysis. The results of the analysis are passed back to Excel and are displayed on one or more worksheets. Optionally, the user can display the SPSS application, transfer SPSS-created data (for example, regression residuals) to Excel, control which SPSS output objects to display, and specify the format for exported SPSS pivot tables.

Development tools. Microsoft Excel macro (Visual Basic for Applications) and SPSS OLE Automation.

Features. The program demonstrates communication and data transfer between software applications. It uses Excel and SPSS OLE Automation to pass data from an

Excel spreadsheet to SPSS for analysis. The SPSS output is passed back to Excel for display and further analysis. The program shows how to open SPSS dialog boxes from another application. It exercises the `ISpssDocuments`, `ISpssDataDoc`, `ISpssOutputDoc`, `ISpssOutputItems`, and `ISpssOutputItem` objects. In addition, it uses the `InvokeDialogAndReturnSyntax` with menu paths, `ExecuteCommands`, `Copy`, and `ExportChart` methods.

Location. The program is located on the SPSS for Windows CD-ROM in `\SPSS\Developer\Programs\Microsoft Excel VBA\dkexcel.xls`.

Requirements. This example requires SPSS for Windows and Excel 97 or later. It works on any Excel worksheet with appropriate numeric data. A sample worksheet, *Employee data.xls*, is included in the same directory as the example.

Running the application. You can run the Excel macro (*dkexcel.xls*) from an Excel worksheet. An easy way to run the macro is to associate it with a toolbar icon. You can also run it from the Visual Basic editor.

To run from a toolbar:

- ▶ Open the macro *dkexcel.xls* in Excel.
- ▶ From the menus choose:
 - Tools
 - Customize...
- ▶ If the toolbar where you want to add the icon is not visible, click the Toolbars tab and then click the check box next to the toolbar name.
- ▶ Click the Commands tab.
- ▶ Select Macros from the Categories list.
- ▶ Drag a custom button icon from the Customized dialog box onto the toolbar (leave the Customize dialog box open).
- ▶ Right-click the toolbar button, and select Assign Macro from the shortcut menu.
- ▶ Select Main from the list of macros.
- ▶ Close the Customize dialog box.
- ▶ Open a worksheet that contains that data you want to analyze in SPSS (any Excel worksheet with appropriate numeric data will work).

To run in the Visual Basic editor:

- ▶ Open the macro *dkexcel.xls* in Excel.
- ▶ From the menus choose:
 - Tools
 - Macro
 - Macros...
- ▶ Select Main from the list of macros.
- ▶ Click Edit.

Production Facility Code

Figure 5-12
SPSS Production Facility



Description. The SPSS Production Facility is distributed with SPSS, so that SPSS runs in an automated fashion. SPSS runs unattended and terminates after executing the last command, so that you can perform other tasks while it runs. Production mode is useful if you often run the same set of time-consuming analyses, such as weekly reports.

The SPSS Production Facility uses command syntax files to instruct SPSS what to do. Each production run creates an output file with the same name as the production job and the extension *.spo*. For example, a production job file named *prodjob.spp* creates an output document named *prodjob.spo*.

Development tools. Visual Basic and SPSS OLE Automation.

Features. The program shows how to use SPSS OLE Automation to create an application that handles routine, time-consuming tasks. This example also has a more complex user interface and error-handling capability compared to the other examples. It also demonstrates how to offer users the choice of running SPSS in a distributed analysis mode, introducing *ISpssCSApp*, *ISpssServers*, and *ISpssServer*.

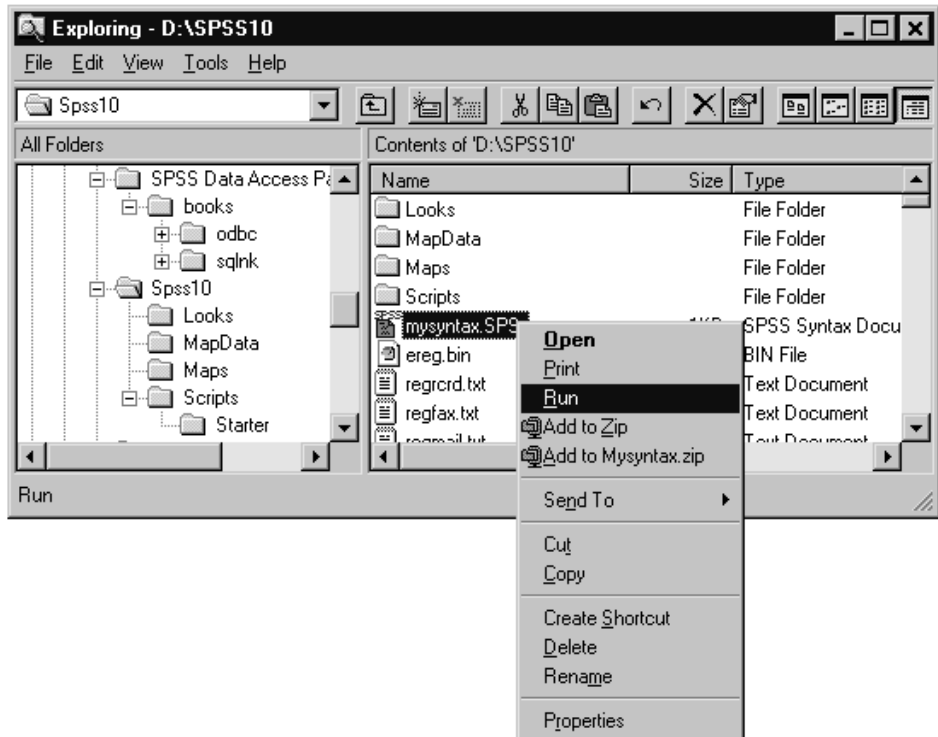
Location. The source code is distributed on the SPSS for Windows CD-ROM in *\SPSS\Developer\Programs\Visual Basic\Source code for SPSS Production Facility*. The executable, *spssprod.exe*, is located in your SPSS for Windows installation directory (*C:\Program Files\SPSS* by default).

Requirements. This example requires SPSS for Windows. It can be used with any valid SPSS command syntax file.

Running the application. You can run the *spssprod.vbp* file from within Visual Basic or execute *prodmode.exe* from your SPSS for Windows installation directory.

Run Syntax Code

Figure 5-13
Running syntax from Windows Explorer



Description. The Run Syntax utility is distributed with SPSS to run an SPSS command syntax file from the Windows Explorer. The utility launches SPSS, opens the syntax file, runs it, and displays the output in a Viewer window. When SPSS for Windows is installed, it automatically registers a RUN command for the syntax (*.sps) document. The RUN command executes *runsyntax.exe* on the currently selected syntax file.

Development tools. Visual Basic and SPSS OLE Automation.

Features. The program shows how you can use SPSS OLE Automation and the Windows Registry to add commands to the Windows Explorer File and shortcut menus. It introduces the Documents Collection (ISpssDocuments) and exercises the ISpssSyntaxDoc, ISpssDataDoc, and ISpssOutputDoc objects. You can see how *runsyntx.exe* is registered by looking at the HKEY_LOCAL_MACHINE\SOFTWARE\Classes\SPSS.SyntaxDoc\shell\Run\command key in the Windows Registry editor.

Location. The source code is distributed on the SPSS for Windows CD-ROM located in \SPSS\Developer\Programs\Visual Basic\Source code for Runsyntx.exe. The executable is in your SPSS for Windows installation directory (C:\Program Files\SPSS by default).

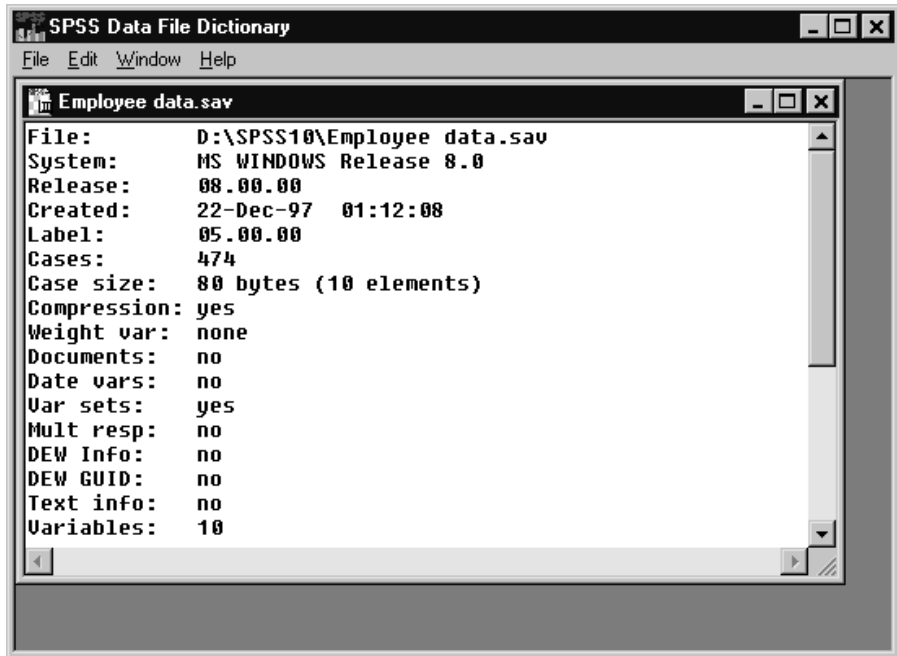
Requirements. This example requires SPSS for Windows. It can be used with any valid SPSS command syntax file.

Running the application. You can run the *runsyntx.vbp* file from within Visual Basic or execute *runsyntx.exe*. This can be done as follows:

- ▶ Select a syntax (*.sps) file in the Windows Explorer.
- ▶ Right-click to get the shortcut menu.
- ▶ From the shortcut menu click Run.

Display Dictionary Information

Figure 5-14
Input/output DLL example



Description. This example demonstrates how to use the SPSS input/output DLL to retrieve data dictionary information (for example, variable names) from an SPSS data file. The user selects a file and views the dictionary information in a child window. There are two versions of the program: one written in Visual C++ and one written in Visual Basic.

Development tools. Visual Basic or Visual C++ and the SPSS input/output DLL. The I/O DLL is documented in Appendix A.

Location. The program is located on the SPSS for Windows CD-ROM in
\\SPSS\\Developer\\IO_DLL\\Smpl_vb\\dictlist.vbp and
\\SPSS\\Developer\\IO_DLL\\Smpl_cpp\\dictlist.cpp.

Requirements. This example requires SPSS for Windows and the input/output DLL *spssio32.dll* that are included on the SPSS for Windows CD-ROM in

`\SPSS\Developer\IO_DLL\`. You'll need Visual Basic or Visual C++ to compile the example.

Running the application. You can run the `.vbp` from within Visual Basic or run the `.cpp` file from within Visual C++. Instructions for compiling executables are located in `\SPSS\Developer\IO_DLL\Smpl_vb\smpl_vb.doc` and `\SPSS\Developer\IO_DLL\Smpl_cpp\smpl_cpp.doc` on the SPSS for Windows CD-ROM.

SPSS Input/Output DLL

An SPSS data file is a binary file that contains the case data on which SPSS operates and a dictionary describing the contents of the file. Many developers have successfully created applications that directly read and write SPSS data files. Some of these developers have asked for a dynamically linked library (DLL) to help them manipulate the rather complex format of SPSS data files. The I/O DLL documented in this appendix is designed to satisfy this need.

You can use the I/O DLL to:

- Read and write SPSS data files
- Set general file attributes, create variables
- Set values for variables
- Read cases
- Copy a dictionary
- Append cases to an SPSS data file
- Directly access data

Developers can call SPSS I/O DLL procedures in client programs written in C, Visual Basic, and other programming languages. It is necessary to include the header file *spssdio.h*. The specific calling conventions are `__pascal` for 16-bit programs and `__stdcall` for 32-bit programs. The `__stdcall` conventions are compatible with FORTRAN, although calling I/O DLL procedures is not specifically supported for FORTRAN programs.

This appendix outlines the steps for developing an application using the I/O DLL procedures. It also contains a description of each procedure.

The I/O DLL files are on the SPSS for Windows CD-ROM in `\spss\developer\io_dll`.

New I/O DLL for SPSS 14.0

The I/O DLL was completely rewritten for SPSS 14.

- The new architecture should facilitate further development. However, much of the code is not used in the SPSS product itself and has not received as much testing as that in the predecessor DLL.
- An unintended but necessary limitation of the new DLL is that the `spssOpenAppend` function will not work correctly on compressed data files created by SPSS systems prior to release 14.
- To assist in the handling of non-western character sets, we are now using IBM's International Components for Unicode or ICU. As a result, the DLL depends on three ICU DLL's which are installed with the SPSS product - *icudt32.dll*, *icuin32.dll*, and *icuuc32.dll*. These have an aggregate size of 11MB.
- The I/O DLL now uses the DLL resident C run-time *msvcr71.dll* and/or the C++ run-time *msvcp71.dll*. If the client application shares this run-time, it will also share the locale. As a result, any call to `spssSetLocale` will affect both the I/O DLL and the client. Such a call is unnecessary if the client has already set the locale. When the DLL is loaded, it sets the locale to the system default.
- Prior to SPSS 14.0.1, the name of the multiple response set specified for `spssAddMultRespDefC` or `spssAddMultRespDefN` was limited to 63 bytes, and the DLL automatically prepended a dollar sign. In the interest of consistency, the name is now limited to 64 bytes and must begin with a dollar sign. Also, the length of the set label was previously limited to 60 bytes. It may now be as long as a variable label, 255 bytes.

Using the I/O DLL

The following sections list the sequence of procedures calls required to complete specific tasks with the I/O DLL. See “DLL Procedure Reference” on p. 121 for detailed information about each procedure.

Writing an SPSS Data File

The sequence of procedure calls to create an SPSS data file is as follows:

1. To open a physical file for output and to initialize internal data structures, call `spssOpenWrite`.
2. To set general file attributes, such as file label and compression, call `spssSetIdString` and `spssSetCompression`. These attributes may also be set anytime before the dictionary is committed (see step 7).
3. To create one or more variables, call `spssSetVarName`.
4. To set attributes of variables, such as output formats, variable and value labels, missing values, etc., call appropriate procedures, such as `spssSetVarPrintFormat`, `spssSetVarLabel`, `spssSetVarNValueLabel`, etc. Variable creation and attribute setting may be interleaved as long as no reference is made to a variable that has not yet been created.
5. (Optional) If you want to set variable sets, Trends date variables, or multiple response set information, call `spssSetVariableSets`, `spssSetDateVariables`, or `spssSetMultRespDefs`.
6. To set the case weight variable, call `spssSetCaseWeightVar`.
7. To commit the dictionary, call `spssCommitHeader`. Dictionary information may no longer be modified.
8. To prepare to set case data values, call `spssGetVarHandle` once for each variable and save the returned variable handles. A variable handle contains an index that allows data to be updated efficiently during case processing. While setting data values, variables must be referenced via their handles and not their names.
9. To set values of all variables for a case, call `spssSetValueChar` for string variables and `spssSetValueNumeric` for numeric ones. To write out the case, call `spssCommitCaseRecord`. Repeat from the beginning of this step until all cases are written.
10. To terminate file processing, call `spssCloseWrite`.

Utility procedures such as `spssSysmisVal` and any of the `spssConvert` procedures may be called at any time. They are useful primarily while setting case data values.

It is possible to construct complete cases in the form the cases would be written to an uncompressed data file and then present them to the DLL for output (which will take care of compression if necessary). Note that it is very easy to write out garbage this way. To use this approach, replace step 8 and step 9 with the following steps:

11. To obtain the size of an uncompressed case record in bytes, call `spssGetCaseSize`. Make sure that the size is what you think it should be. Allocate a buffer of that size.
12. Fill up the buffer with the correctly encoded numeric and string values, taking care of blank padding and doubleword alignment. To write the case, call `spssWholeCaseOut`. Repeat from the beginning of this step until all cases are written.

Copying a Dictionary

Developers can open a new file for output and initialize its dictionary from that of an existing file. The function, `spssOpenWriteCopy`, that implements this feature is a slight extension of `spssOpenWrite`. It is useful when the dictionary or data of an existing file is to be modified or all of its data is to be replaced. The typical sequence of operations is:

1. Call `spssOpenWriteCopy (newFileName, oldFileName, ...)` to open a new file initialized with a copy of the old file's dictionary.
2. Call `spssOpenRead (oldFileName, ...)` to access the old file's data.

Appending Cases to a Existing SPSS Data File

To append cases, the existing data file must be compatible with the host system; that is, the system that originally created the file must use the same bit ordering and the same representation for the system-missing value as the host system. For example, a file created on a computer that uses high-order-first bit ordering (for example, Motorola) cannot be extended on an computer that uses low-order-first bit ordering (for example, Intel).

When appending cases, no changes are made to the dictionary other than the number of cases. The originating system and the creation date are not modified.

The sequence of procedure calls to append cases to an existing SPSS data file is as follows:

1. To open a physical file and to initialize internal data structures, call `spssOpenAppend`.
2. To get general file attributes, such as file label, compression, and case weight, call `spssGetIdString`, `spssGetCompression`, and `spssGetCaseWeightVar`. To get the list of variable names and types, call `spssGetVarNames`, or call `spssGetNumberOfVariables` and `spssGetVarInfo` if you are using Visual Basic. To get attributes of variables,

such as output formats, variable and value labels, missing values, etc., call `spssGetVarPrintFormat`, `spssGetVarLabel`, `spssGetVarNValueLabel(s)`, etc.

3. To set values of all variables for a case, call `spssSetValueChar` for string variables and `spssSetValueNumeric` for numeric variables. To append the case, call `spssCommitCaseRecord`. Repeat from the beginning of this step until all cases are written.
4. To terminate file processing, call `spssCloseAppend`.

Utility procedures such as `spssSysmisVal` and any of the `spssConvert` procedures may be called at any time. They are useful primarily while setting case data values.

For step 3, it is also possible to call `spssWholeCaseOut` to construct complete cases in the form in which the cases would be written to an uncompressed data file and then present them to the DLL for output (which will take care of compression if necessary). The same precaution should be taken as you write whole cases to an SPSS data file.

Reading an SPSS Data File

The sequence of procedure calls to read an SPSS data file is much less restricted than the sequence of calls to write an SPSS data file. Cases, of course, must be read in sequence. However, calls that report file or variable attributes may be made anytime after the file is opened. A typical sequence of steps is:

1. To open a physical file for input and to initialize internal data structures, call `spssOpenRead`.
2. To get general file attributes, such as file label, compression, and case weight, call `spssGetIdString`, `spssGetCompression`, and `spssGetCaseWeightVar`. To get the list of variable names and types, call `spssGetVarNames`, or call `spssGetNumberOfVariables` and `spssGetVarInfo` if you are using Visual Basic. To get attributes of variables, such as output formats, variable and value labels, missing values, etc., call `spssGetVarPrintFormat`, `spssGetVarLabel`, `spssGetVarNValueLabel(s)`, etc.
3. (Optional) If you want to set variable sets, Trends date variables, or multiple response set information, call `spssSetVariableSets`, `spssSetDateVariables`, or `spssSetMultRespDefs`.
4. To find out the number of cases in the file, call `spssGetNumberOfCases`.
5. To prepare to read case values, call `spssGetVarHandle` once for each variable whose values are of interest and save the returned variable handles. A variable

handle contains an index that allows data to be retrieved efficiently during case processing. While retrieving data values, variables must be referenced via their handles and not their names.

6. To read the next case into the library's internal buffers, call `spssReadCaseRecord`. To get values of variables for a case, call `spssGetValueChar` for string variables and `spssGetValueNumeric` for numeric ones. Repeat from the beginning of this step until all cases are read.
7. To terminate file processing, call `spssCloseRead`.

Utility procedures such as `spssSysmisVal` and any of the `spssConvert` procedures may be called at any time. They are useful primarily while interpreting case data values. The `spssFree...` procedures should also be used where appropriate to free dynamically allocated data returned by the library.

Here, too, it is possible to receive from the DLL complete cases in the form in which the cases would appear in an uncompressed data file. Extracting data values from the case record is entirely up to the caller in this case. For this approach, replace step 5 and step 6 with the following steps:

8. To obtain the size of an uncompressed case record in bytes, call `spssGetCaseSize`. Allocate a buffer of that size.
9. To read the next case into your buffer, call `spssWholeCaseIn`. Extract the values you need from the buffer. Repeat from the beginning of this step until all cases are read.

Direct Access Input

The File I/O API supports direct access to the data in existing files. The basic mechanism is to call `spssSeekNextCase`, specifying a zero-origin case number before calling `spssWholeCaseIn` or `spssReadCaseRecord`. Note that direct reads from compressed SPSS data files require reading all of the data up to the requested case—that is, performance may not be sparkling when retrieving a few cases. Once an index of the cases has been constructed, performance is adequate.

Working with SPSS Data Files

Variable Names and String Values

A user-definable SPSS variable name must be valid in the current locale. In SPSS for Windows, variable names must obey the following rules:

- The name must begin with a letter. The remaining characters may be any letter, any digit, a period, or the symbols @, #, _, or \$.
- Variable names cannot end with a period. Names that end with an underscore should be avoided (to avoid name conflicts with variables automatically created by some procedures).
- The length of the name cannot exceed 64 bytes.
- Blanks and special characters (for example, !, ?, *) cannot be used.
- Each variable name must be unique; duplication is not allowed. Variable names are not case sensitive. The names *NEWVAR*, *NewVar*, and *newvar* are all considered identical.
- Reserved keywords (ALL, NE, EQ, TO, LE, LT, BY, OR, GT, AND, NOT, GE, and WITH) cannot be used.

If the names in an SPSS data file created in another locale are invalid in the current locale (for example, double-byte characters), the I/O DLL will create acceptable names. These names are returned upon inquiry and can be used as legitimate parameters in procedures requiring variable names. The names in the data file will not be changed.

In the I/O DLL, procedures that return variable names return them in upper case as null-terminated strings without any trailing blanks. Procedures that take variable names as input will accept mixed case and any number of trailing blanks without a problem. These procedures change everything to upper case and trim trailing blanks before using the variable names.

Similarly, procedures that return values of string variables return them as null-terminated strings whose lengths are equal to the lengths of the variables. Procedures that take string variable values as input accept any number of trailing blanks and effectively trim the values to the variables' lengths before using them.

Accessing Variable and Value Labels

Beginning with SPSS 7.5, the limit on the length of variable labels was increased from 120 to 256 bytes. There were two ways in which the `spssGetVarLabel` function could be modified to handle the longer labels. First, it could continue to return a maximum of 120 bytes for compatibility with existing applications. Second, it could return a maximum of `SPSS_MAX_VARLABEL` bytes for compatibility with new SPSS data files. The resolution was to continue to return a maximum of 120 bytes and to introduce a new function, `spssGetVarLabelLong`, which permits the client to specify the maximum number of bytes to return. In anticipation of possible future increases in the maximum width of value labels, two parallel functions, `spssGetVarNValueLabelLong` and `spssGetVarCValueLabelLong`, were added for retrieving the value labels of numeric and short string variables.

System-Missing Value

The special floating point value used to encode the system-missing value may differ from platform to platform, and the value encoded in an SPSS data file may differ from the one used on the host platform (one on which the application and the DLL are running). Files written through the DLL use the host system-missing value, which may be obtained by calling `spssSysmisVal`. For files being read using the DLL, data values having the system-missing value encoded in the file are converted to the host system-missing value; the system-missing value used in the data file is invisible to the user of the DLL.

Measurement Level, Column Width, and Alignment

Starting with release 8.0, SPSS supports three additional variable attributes: measurement level, column width, and alignment. These attributes are not necessarily present in an SPSS data file. However, when one attribute is recorded for a variable, all three must be recorded for every variable. Default values are assigned as necessary.

For example, if a new data file is being created and the measurement level attribute is explicitly set for one variable, default values will be assigned to measurement levels of all remaining variables, and column widths and alignments will be assigned to all variables. If no measurement level, column width, or alignment is assigned, the file will be written without values for any attribute.

There are six new file I/O API functions to access to these attributes: `spssGetVarMeasureLevel`, `spssSetVarMeasureLevel`, `spssGetVarColumnWidth`, `spssSetVarColumnWidth`, `spssGetVarAlignment`, and `spssSetVarAlignment`.

Support for Documents

SPSS has a `DOCUMENT` command that can be used to store blocks of text in a data file. Until release 8.0, the I/O API had no support for documents—stored documents, if any, were discarded when opening an existing file, and there was no way to add documents to a new file. Starting with release 8.0, limited support for stored documents is provided that allows the user to retain existing documents.

When a file is opened for reading, its documents record is read and kept; if a file being written out has documents, they are stored in the dictionary. Since there is still no way to explicitly get or set documents, one may wonder how it is possible for an output file to acquire documents. The answer is, by using `spssOpenWriteCopy` to initialize a dictionary or by calling the `spssCopyDocuments` function to copy documents from one file to another. If an output file is created with `spssOpenWriteCopy`, the documents record of the file the dictionary is copied from is retained and written out when the dictionary is.

Coding Your Program

Any source file that references DLL procedures must include the header *spssdio.h*. The latter provides ANSI C prototypes for the DLL procedures and defines useful macros; it does not require any other headers to be included beyond what your program requires. To protect against name clashes, all DLL function names start with *spss* and all macro names are prefixed with *SPSS_*. In addition to the macros explicitly mentioned in the DLL procedures, *spssdio.h* defines macros for the maximum sizes of various SPSS data file objects that may help to make your program a little more readable:

SPSS_MAX_VARNAME	Variable name
SPSS_MAX_SHORTSTRING	Short string variable
SPSS_MAX_IDSTRING	File label string
SPSS_MAX_LONGSTRING	Long string variable
SPSS_MAX_VALLABEL	Value label
SPSS_MAX_VARLABEL	Variable label

16-Bit Versus 32-Bit DLL

Beginning with SPSS 12, there is no longer a 16-bit version of the I/O DLL; there is only a 32-bit version: *spssio32.dll*, *spssio32.lib*, and *spssdio.h*.

Visual Basic Clients

The file *spssdio.bas* contains declarations of most of the API functions in a format that can be used in Visual Basic. The file also contains definitions of symbolic constants for all of the function return codes and the SPSS format codes. Three comments are relevant to this file:

- It is necessary to have a knowledge of Chapter 26, “Calling Procedures in DLLs,” in the *Microsoft Visual Basic Programmer’s Guide*. Note that where the API function parameter should be an int, a 32-bit Visual Basic application should use a long, but a 16-bit application should use an integer. Also, you should be careful to make string parameters suitably long before calling the API.

- Some functions, such as `spssGetVarNames`, are not compatible with being called from Visual Basic. The declarations of these functions are present only as comments.
- Only about 20% of the functions have actually been called from a working Visual Basic program. The inference is that some of the declarations are probably incorrect.

The function `spssGetVarNames` is a little difficult to call from languages other than C because it returns pointers to two vectors. BASIC and FORTRAN are not very well equipped to deal with pointers. Instead, use functions `spssGetNumberOfVariables` and `spssGetVarInfo`, which enable the client program to access the same information in a little different way. Another function, `spssHostSysmisVal`, is provided as an alternative to `spssSysmisVal` to avoid returning a double on the stack.

Borland C++

Borland C++ users can use release 8.0.1 and later of *spssio32.dll* and the associated *spssdio.h*. They cannot, however, use the distributed *spssio32.lib*. It is necessary to generate an import library from the distributed DLL using the *implib.exe* console application, which comes with the compiler using the following syntax:

```
implib -w spssio32.lib spssio32.dll
```

The `-w` switch suppresses almost 100 warnings, such as the following:

Warning duplicate symbol: `spssCloseAppend`

Sample Programs

The developer's tools include a sample Windows MDI application, described in Chapter 5 on p. 107, which utilizes the I/O DLL to read dictionary information from an SPSS data file. The source files for the application are present in both 16-bit and 32-bit versions, as *dictls16* and *dictls*, respectively.

The dictionary sample code was initially written using the Visual C++ IDE (Integrated Development Environment). As distributed, it is composed of two principal C++ source files: *dictlist.cpp*, which contains the boilerplate for the application, the MDI frame window, the child window, and the view; and *dictdoc.cpp*, which implements the document class and contains all of the I/O DLL calls.

Two make files are supplied: *dictls16.mak*, which builds the 16-bit version of the application in a subdirectory named *16* and is compatible with Microsoft Visual C++ 1.52, and *dictls32.mak*, which builds the 32-bit version in a subdirectory named *32* and is compatible with Microsoft Visual C++ 4.0. The make files generate applications using the DLL resident version of the MFC (Microsoft Foundation Classes) run-time. For different compiler versions, you may need to modify the make files. These make files are not those generated by the IDE and should be easy to modify.

The DICTLIST sample application is distributed with the I/O DLL on the SPSS for Windows CD-ROM in *\spss\developer\io_dll\smpl_cpp*.

The developer's tools also include a sample Visual Basic application. This application mimics the sample C++ application in that it displays the dictionaries of SPSS data files. The application consists of an MDI frame window (*dictmdi.frm*), an about box (*dictabou.frm*), and an MDI child window (*dictchld.frm*). The MDI child window does most of the work and makes most of the calls to the API functions. A Visual Basic project file (*dictlist.vbp*) is also provided. When working with this project, the function declarations file (*spssdio.bas*) must be available on the SPSS for Windows CD-ROM in *\spss\developer\io_dll\smpl_vb*.

DLL Procedure Reference

The procedures are listed in alphabetical order.

spssAddFileAttribute

```
int spssAddFileAttribute(
    const int hFile,
    const char* attribName,
    const int attribSub,
    const char* attribText)
```

Description

This function adds a single datafile attribute. If the attribute already exists, it is replaced. The attribute name and its subscript are specified separately. The subscript is unit origin. If the attribute is not subscripted, the subscript must be specified as -1.

Parameter	Description
<i>hfile</i>	Handle to the data file
<i>attribName</i>	Name of the attribute. Not case sensitive.
<i>attribSub</i>	Unit origin subscript or -1
<i>attribText</i>	Text which used as the attribute's value

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDONLY	The file is read-only
SPSS_DICT_COMMIT	spssCommitHeader has already been called
SPSS_INVALID_ATTRDEF	Missing name, missing text, or invalid subscript

SPSS_INVALID_ATTRNAME Lexically invalid attribute name

spssAddMultRespDefC

```
int spssAddMultRespDefC(int handle, const char *mrSetName,
const char *mrSetLabel, int isDichotomy, const char *countedValue,
const char **varNames, int numVars)
```

Description

This function adds a multiple response set definition over short string variables to the dictionary.

Parameter	Description
<i>handle</i>	Handle to the data file.
<i>mrSetName</i>	Name of the multiple response set. A null-terminated string up to 64 bytes long that begins with a dollar sign and obeys the rules for a valid variable name. Case is immaterial.
<i>mrSetLabel</i>	Label for the multiple response set. A null-terminated string up to 256 bytes long. May be NULL or the empty string to indicate that no label is desired.
<i>isDichotomy</i>	Nonzero if the variables in the set are coded as dichotomies, zero otherwise.
<i>countedValue</i>	A null-terminated string containing the counted value. Necessary when <i>isDichotomy</i> is nonzero, in which case it must be 1–8 characters long, and ignored otherwise. May be NULL if <i>isDichotomy</i> is zero.
<i>varNames</i>	Array of null-terminated strings containing the names of the variables in the set. All variables in the list must be short strings. Case is immaterial.
<i>numVars</i>	Number of variables in the list (in <i>varNames</i>). Must be at least two.

Returns

If all goes well, adds the multiple response set to the dictionary and returns zero (SPSS_OK) or negative (a warning). Otherwise, returns a positive error code and does not add anything to the multiple response sets already defined, if any.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	The file is open for input or append
SPSS_DICT_COMMIT	spssCommitHeader has already been called
SPSS_NO_VARIABLES	Fewer than two variables in list
SPSS_EXC_STRVALUE	isDichotomy is nonzero and countedValue is NULL, empty, or longer than 8 characters
SPSS_INVALID_MRSETNAME	The multiple response set name is invalid
SPSS_DUP_MRSETNAME	The multiple response set name is a duplicate
SPSS_INVALID_MRSETDEF	Existing multiple response set definitions are invalid
SPSS_INVALID_VARNAME	One or more variable names in list are invalid
SPSS_VAR_NOTFOUND	One or more variables in list were not found in dictionary
SPSS_SHORTSTR_EXP	At least one variable in the list is numeric or long string
SPSS_NO_MEMORY	Insufficient memory to store the definition

spssAddMultRespDefExt

```
int spssAddMultRespDefExt(
    const int hFile,
    const spssMultRespDef* pSet)
```

Description

This function adds one multiple response set to the dictionary. The set is described in a struct which is defined in *spssdio.h*.

Parameter	Description
<i>hFile</i>	Handle to the data file
<i>pSet</i>	Pointer to the struct defining the set

The struct itself is defined as:

```
typedef struct spssMultRespDef_T
{
    char szMrSetName[SPSS_MAX_VARNAME+1]; /* Null-terminated MR set name */
    char szMrSetLabel[SPSS_MAX_VARLABEL+1]; /* Null-terminated set label */
    int qIsDichotomy; /* Whether a multiple dichotomy set */
    int qIsNumeric; /* Whether the counted value is numeric */
    int qUseCategoryLabels; /* Whether to use var category labels */
    int qUseFirstVarLabel; /* Whether using var label as set label */
    int Reserved[14]; /* Reserved for expansion */
    long nCountedValue; /* Counted value if numeric */
    char* pszCountedValue; /* Null-terminated counted value if string */
    char** ppszVarNames; /* Vector of null-terminated var names */
    int nVariables; /* Number of constituent variables */
} spssMultRespDef;
```

The items in the struct are as follows:

Item	Description
<i>szMrSetName</i>	Null-terminated name for the set. Up to 64 bytes. Must begin with "\$".
<i>szMrSetLabel</i>	Null-terminated label for the set. Up to 256 bytes.
<i>qIsDichotomy</i>	True (non-zero) if this is a multiple dichotomy, i.e. an "MD" set.
<i>qIsNumeric</i>	True if the counted value is numeric. Applicable only to multiple dichotomies.
<i>qUseCategoryLabels</i>	True for multiple dichotomies for which the categories are to be labeled by the value labels corresponding to the counted value.
<i>qUseFirstVarLabel</i>	True for multiple dichotomies for which the label for the set is taken from the variable label of the first constituent variable.
<i>nCountedValue</i>	The counted value for numeric multiple dichotomies.
<i>pszCountedValue</i>	Pointer to the null-terminated counted value for character multiple dichotomies.

<i>ppszVarNames</i>	Pointer to a vector of null-terminated variable names.
<i>nVariables</i>	Number of variables in the set

When adding a set, the set name must be unique, and the variables must exist and be of the appropriate type - numeric or character depending on `qlsNumeric`.

Returns

The function returns `SPSS_OK` or an error value.

Error Code	Description
<code>SPSS_OK</code>	No error
<code>SPSS_INVALID_HANDLE</code>	The file handle is invalid
<code>SPSS_OPEN_RDMODE</code>	The file is not open for output
<code>SPSS_DICT_COMMIT</code>	The dictionary has already been committed
<code>SPSS_INVALID_MRSETNAME</code>	Invalid name for the set
<code>SPSS_INVALID_MRSETDEF</code>	Invalid or inconsistent members of the definition struct.
<code>SPSS_DUP_MRSETNAME</code>	A set with the same name already exists

spssAddMultRespDefN

```
int spssAddMultRespDefN(int handle, const char *mrSetName,
const char *mrSetLabel, int isDichotomy, long countedValue,
const char **varNames, int numVars)
```

Description

This function adds a multiple response set definition over numeric variables to the dictionary.

Parameter	Description
<i>handle</i>	Handle to the data file.
<i>mrSetName</i>	Name of the multiple response set. A null-terminated string up to 64 bytes that begins with a dollar sign and obeys the rules for a valid variable name. Case is immaterial.

<i>mrSetLabel</i>	Label for the multiple response set. A null-terminated string up to 256 bytes long. May be NULL or the empty string to indicate no label is desired.
<i>isDichotomy</i>	Nonzero if the variables in the set are coded as dichotomies, zero otherwise.
<i>countedValue</i>	The counted value. Necessary when <i>isDichotomy</i> is nonzero and ignored otherwise. Note that the value is specified as a long int, not a double.
<i>varNames</i>	Array of null-terminated strings containing the names of the variables in the set. All variables in the list must be numeric. Case is immaterial.
<i>numVars</i>	Number of variables in the list (in <i>varNames</i>). Must be at least two.

Returns

If all goes well, adds the multiple response set to the dictionary and returns zero (SPSS_OK) or negative (a warning). Otherwise, returns a positive error code and does not add anything to the multiple response sets already defined, if any.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	The file is open for input or append
SPSS_DICT_COMMIT	<i>spssCommitHeader</i> has already been called
SPSS_NO_VARIABLES	Fewer than two variables in list
SPSS_INVALID_MRSETNAME	The multiple response set name is invalid
SPSS_DUP_MRSETNAME	The multiple response set name is a duplicate
SPSS_INVALID_MRSETDEF	Existing multiple response set definitions are invalid
SPSS_INVALID_VARNAME	One or more variable names in list are invalid
SPSS_VAR_NOTFOUND	One or more variables in list were not found in dictionary
SPSS_NUME_EXP	At least one variable in the list is not numeric
SPSS_NO_MEMORY	Insufficient memory to store the definition

spssAddVarAttribute

```
int spssAddVarAttribute(  
    const int hFile,  
    const char* varName,  
    const char* attribName,  
    const int attribSub,  
    const char* attribText)
```

Description

This function is analogous to `spssAddFileAttribute`, but it operates on a single variable's set of attributes. If the named attribute does not already exist, it is added to the set of attributes. If it does exist, the existing definition is replaced. If the attribute is not subscripted, the unit origin subscript is specified as -1.

Parameter	Description
<i>hFile</i>	Handle to the data file
<i>varName</i>	Name of the variable. Not case sensitive.
<i>attribName</i>	Name of the attribute. Not case sensitive.
<i>attribSub</i>	Unit origin attribute or -1
<i>attribText</i>	Text which used as the attribute's value

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_VAR_NOTFOUND	Named variable is not in the file
SPSS_OPEN_RDMODE	The file is read-only

SPSS_DICT_COMMIT	spssCommitHeader has already been called
SPSS_INVALID_ATTRDEF	Missing name, missing text, or invalid subscript
SPSS_INVALID_ATTRNAME	Lexically invalid attribute name

spssCloseAppend

int spssCloseAppend (int *handle*)

Description

This function closes the data file associated with *handle*, which must have been opened for appending cases using spssOpenAppend. The file handle *handle* becomes invalid and no further operations can be performed using it.

Parameter	Description
<i>handle</i>	Handle to the data file

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDONLY	File is open for reading, not appending, cases
SPSS_FILE_WERROR	File write error

Example

```
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;             /* error code */
    ...
    error = spssOpenAppend("bank.sav", &fH);
    ...
    error = spssCloseAppend(fH);
    ...
    /* Handle fH is now invalid */
}
```

See also **spssOpenAppend**.

spssCloseRead

int spssCloseRead (int *handle*)

Description

This function closes the data file associated with *handle*, which must have been opened for reading using **spssOpenRead**. The file handle *handle* becomes invalid and no further operations can be performed using it.

Parameter	Description
<i>handle</i>	Handle to the data file

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_WRMODE	File is open for writing, not reading

Example

```
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;             /* error code */
    ...
    error = spssOpenRead("bank.sav", &fH);
    ...
    error = spssCloseRead(fH);
    ...
    /* Handle fH is now invalid */
}
```

See also **spssOpenRead**.

spssCloseWrite

int spssCloseWrite (int *handle*)

Description

This function closes the data file associated with *handle*, which must have been opened for writing using **spssOpenWrite**. The file handle *handle* becomes invalid and no further operations can be performed using it.

Parameter	Description
<i>handle</i>	Handle to the data file

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	File is open for reading, not writing
SPSS_DICT_NOTCOMMIT	Dictionary of the output file has not yet been written with spssCommitHeader

SPSS_FILE_WERROR

File write error

Example

See `spssSetValueNumeric`.

See also `spssOpenWrite`.

spssCommitCaseRecord

int `spssCommitCaseRecord` (int *handle*)

Description

This function writes a case to the data file specified by the *handle*. It must be called after setting the values of variables through `spssSetValueNumeric` and `spssSetValueChar`. Any variables left unset will get the system-missing value if they are numeric and all blanks if they are strings. Unless `spssCommitCaseRecord` is called, the case will not be written out.

Parameter	Description
<i>handle</i>	Handle to the data file

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	File is open for reading, not writing
SPSS_DICT_NOTCOMMIT	Dictionary of the output file has not yet been written with <code>spssCommitHeader</code>
SPSS_FILE_WERROR	File write error

Example

See `spssSetValueNumeric`.

See also `spssSetValueNumeric`, `spssSetValueChar`.

spssCommitHeader

int `spssCommitHeader` (int *handle*)

Description

This function writes the data dictionary to the data file associated with *handle*. Before any case data can be written, the dictionary must be committed; once the dictionary has been committed, no further changes can be made to it.

Parameter	Description
<i>handle</i>	Handle to the data file

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error.
SPSS_INVALID_HANDLE	The file handle is not valid.
SPSS_OPEN_RDONLY	File is open for reading, not writing.
SPSS_DICT_COMMIT	Dictionary has already been written with <code>spssCommitHeader</code> .
SPSS_DICT_EMPTY	No variables defined in the dictionary.
SPSS_FILE_WERROR	File write error. In case of this error, the file associated with <i>handle</i> is closed and <i>handle</i> is no longer valid.
SPSS_NO_MEMORY	Insufficient memory.

SPSS_INTERNAL_VLABS

Internal data structures of the DLL are invalid. This signals an error in the DLL.

Example

```
#include "spssdio.h"

void func()
{
    int fH;                /* file handle */
    int error;             /* error code */
    ...
    error = spssOpenWrite("data.sav", &fH);
    ...
    /* Create some variables */
    error = spssSetVarName(fH, "AGE", SPSS_NUMERIC);
    ...
    /* Label variables -- Not required but useful */
    error = spssSetVarLabel(fH, "AGE", "Age of the Employee");
    ...
    /* Done with dictionary definition; commit dictionary */
    error = spssCommitHeader(fH);
    /* Handle errors... */
    ...
}
```

spssConvertDate

int spssConvertDate (int *day*, int *month*, int *year*, double **spssDate*)

Description

This function converts a Gregorian date expressed as day-month-year to the internal SPSS date format. The time portion of the date variable is set to 0:00. To set the time portion of the date variable to another value, use `spssConvertTime` and add the resulting value to **spssDate*. Dates before October 15, 1582, are considered invalid.

Parameter	Description
<i>day</i>	Day of month (1–31)
<i>month</i>	Month (1–12)
<i>year</i>	Year in full (94 means 94 A.D.)
<i>spssDate</i>	Pointer to date in internal SPSS format

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_DATE	Invalid date

Example

```
#include "spssdio.h"

void func()
{
    int    fH;                /* file handle      */
    int    error;             /* error code       */
    double vH;               /* variable handle  */
    double sDate;            /* SPSS date       */
    double sTime;            /* SPSS time       */
    ...
    error = spssOpenWrite("data.sav", &fH);
    ...
    /* Create a numeric variable and set its print format
    ** to DATETIME28.4
    */
    error = spssSetVarName(fH, "TIMESTMP", SPSS_NUMERIC);
    ...
    error =
    spssSetVarPrintFormat(fH, "TIMESTMP", SPSS_FMT_DATE_TIME, 4, 28);
    ...
    /* Get variable handle for TIMESTMP */
    error = spssGetVarHandle(fH, "TIMESTMP", &vH);
    ...
    /* Set value of TIMESTMP for first case to May 9, 1948,
    ** 10:30 AM. Do this by first using spssConvertDate to get
    ** a date value equal to May 9, 1948, 0:00 and adding to it
    ** a time value for 10:30:00.
    */
    error = spssConvertDate(9, 5, 1948, &sDate);
    ...
    /* Note that the seconds value is double, not int */
    error = spssConvertTime(0L, 10, 30, 0.0, &sTime);
    ...
    /* Set the value of the date variable */
    error = spssSetValueNumeric(fH, vH, sDate+sTime);
    ...
}
```

See also **spssConvertTime**.

spssConvertSPSSDate

int spssConvertSPSSDate (int **day*, int **month*, int **year*, double *spssDate*)

Description

This function converts the date (as distinct from time) portion of a value in internal SPSS date format to Gregorian style.

Parameter	Description
<i>day</i>	Pointer to day of month value
<i>month</i>	Pointer to month value
<i>year</i>	Pointer to year value
<i>spssDate</i>	Date in internal SPSS format

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_DATE	The date value (<i>spssDate</i>) is negative

Example

```

#include <stdio.h>
#include "spssdio.h"
void func()
{
    int    fH;                /* file handle      */
    int    error;             /* error code       */
    int    day, month, year;  /* date components  */
    int    hour, min;         /* time components  */
    long   jday;              /* Julian day       */
    double sec;               /* seconds component*/
    double vH                 /* variable handle  */
    double sDate;             /* SPSS date+time   */
    ...
    error = spssOpenRead("myfile.sav", &fH);
    ...
    /* Get handle for TIMESTMP, a date variable */
    error = spssGetVarHandle(fH, "TIMESTMP" &vH);
    ...
    /* Read first case and print value of TIMESTMP */
    error = spssReadCaseRecord(fH);
    ...
    error = spssGetValueNumeric(fH, vH, &sDate);
    ...
    error = spssConvertSPSSDate(&day, &month, &year, sDate);
    ...
    /* We ignore jday, day number since Oct. 14, 1582 */
    error =
    spssConvertSPSSTime(&jday, &hour, &min, &sec, sDate);
    ...
    printf("Month/Day/Year: %d/%d/%d, H:M:S: %d:%d:%g\n",
           month, day, year, hour, min, sec);
    ...
}

```

spssConvertSPSSTime

```
int spssConvertSPSSTime  
(long *day, int *hour, int *minute, double *second, double spssTime)
```

Description

This function breaks a value in internal SPSS date format into a day number (since October 14, 1582) plus the hour, minute, and second values. Note that the seconds value is stored in a double since it may have a fractional part.

Parameter	Description
<i>day</i>	Pointer to day count value (note that the value is long)
<i>hour</i>	Pointer to hour of day
<i>minute</i>	Pointer to minute of the hour
<i>second</i>	Pointer to second of the minute
<i>spssTime</i>	Date in internal SPSS format

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_TIME	The date value (<i>SpssTime</i>) is negative

Example

See `spssConvertSPSSDate`.

spssConvertTime

```
int spssConvertTime (long day, int hour, int minute, double second, double *spssTime)
```

Description

This function converts a time given as day, hours, minutes, and seconds to the internal SPSS format. The day value is the number of days since October 14, 1582, and is typically zero, especially when this function is used in conjunction with `spssConvertDate`. Note that the seconds value is stored in a double since it may have a fractional part.

Parameter	Description
<i>day</i>	Day (non-negative; note that the value is long)
<i>hour</i>	Hour (0–23)
<i>minute</i>	Minute (0–59)
<i>second</i>	Seconds (non-negative and less than 60)
<i>spssTime</i>	Pointer to time in internal SPSS format

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_TIME	Invalid time

Example

See `spssConvertDate`.

See also `spssSetValueNumeric`.

spssCopyDocuments

int `spssCopyDocuments` (int *fromHandle*, int *toHandle*)

Description

This function copies stored documents, if any, from the file associated with *fromHandle* to that associated with *toHandle*. The latter must be open for output. If the target file already has documents, they are discarded. If the source file has no documents, the target will be set to have none, too.

Parameter	Description
<i>fromHandle</i>	Handle to the file documents are to be copied from.
<i>toHandle</i>	Handle to the file documents are to be copied to. Must be open for output.

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	At least one handle is not valid
SPSS_OPEN_RDMODE	The target file is open for input or append
SPSS_DICT_COMMIT	spssCommitHeader has already been called for the target file

spssFreeAttributes

```
int spssFreeAttributes(
    char** attribNames,
    char** attribText,
    const int nAttributes)
```

Description

This function frees the memory dynamically allocated by either `spssGetFileAttributes` or `spssGetVarAttributes`.

<i>Parameter</i>	Description
------------------	-------------

<i>attribNames</i>	Pointer to the vector of attribute names
<i>attribText</i>	Pointer to the vector of text values
<i>nAttributes</i>	The number of elements in each vector

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_CANNOT_FREE	Program exception attempting to free the memory

spssFreeDateVariables

int spssFreeDateVariables (long* *dateInfo*)

Description

This function is called to return the memory allocated by spssGetDateVariables.

Parameter	Description
<i>dateInfo</i>	Vector of date variable indexes

Returns

Always returns SPSS_OK indicating success.

See also **spssGetDateVariables**.

spssFreeMultRespDefs

int spssFreeMultRespDefs(char **mrespDefs*)

Description

This function releases the memory which was acquired by `spssGetMultRespDefs`.

Parameter	Description
<code>mrespDefs</code>	ASCII string containing the definitions

Returns

The function always succeeds and always returns `SPSS_OK`.

See also `spssGetMultRespDefs`.

spssFreeMultRespDefStruct

```
int spssFreeMultRespDefStruct(
    spssMultRespDef* pSet)
```

Description

This function releases the memory acquired by `spssGetMultRespDefByIndex`. It has a single parameter, a pointer to the allocated struct.

Returns

The function returns `SPSS_OK` or an error code.

Error Code	Description
<code>SPSS_OK</code>	No error
<code>SPSS_CANNOT_FREE</code>	Cannot deallocate the memory, probably an invalid pointer

See also “`spssGetMultRespDefByIndex`” on page 157

spssFreeVarCValueLabels

```
int spssFreeVarCValueLabels (char **values, char **labels, int numLabels)
```

Description

This function frees the two arrays and the value and label strings allocated on the heap by `spssGetVarCValueLabels`.

Parameter	Description
<i>values</i>	Array of pointers to values returned by <code>spssGetVarCValueLabels</code>
<i>labels</i>	Array of pointers to labels returned by <code>spssGetVarCValueLabels</code>
<i>numLabels</i>	Number of values or labels returned by <code>spssGetVarCValueLabels</code>

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_CANNOT_FREE	Unable to free because arguments are illegal or inconsistent (for example, negative <i>numLabels</i>)

Example

See `spssGetVarNValueLabels`.

See also `spssFreeVarCValueLabels`.

spssFreeVariableSets

```
int spssFreeVariableSets (char *varSets)
```

Description

This function is called to return the memory allocated by `spssGetVariableSets`.

Parameter	Description
<i>varSets</i>	The string defining the variable sets

Returns

Always returns SPSS_OK indicating success.

See also **spssGetVariableSets**.

spssFreeVarNValueLabels

int spssFreeVarNValueLabels (double **values*, char ***labels*, int *numLabels*)

Description

This function frees the two arrays and the label strings allocated on the heap by **spssGetVarNValueLabels**.

Parameter	Description
<i>values</i>	Array of values returned by spssGetVarNValueLabels
<i>labels</i>	Array of pointers to labels returned by spssGetVarNValueLabels
<i>numLabels</i>	Number of values or labels returned by spssGetVarNValueLabels

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_CANNOT_FREE	Unable to free because arguments are illegal or inconsistent (for example, negative <i>numLabels</i>)

Example

See **spssGetVarNValueLabels**.

See also **spssFreeVarCValueLabels**.

spssFreeVarNames

```
int spssFreeVarNames (char **varNames, int *varTypes, int numVars)
```

Description

This function frees the two arrays and the name strings allocated on the heap by spssGetVarNames.

Parameter	Description
<i>varNames</i>	Array of pointers to names returned by spssGetVarNames
<i>varTypes</i>	Array of variable types returned by spssGetVarNames
<i>numVars</i>	Number of variables returned by spssGetVarNames

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_CANNOT_FREE	Unable to free because arguments are illegal or inconsistent (for example, negative <i>numVars</i>)

Example

See spssGetVarNames.

spssGetCaseSize

```
int spssGetCaseSize (int handle, long *caseSize)
```

Description

This function reports the size of a raw case record for the file associated with *handle*. The case size is reported in bytes and is meant to be used in conjunction with the low-level case input/output procedures `spssWholeCaseIn` and `spssWholeCaseOut`.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>caseSize</i>	Pointer to size of case in bytes

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_DICT_NOTCOMMIT	The file is open for output, and the dictionary has not yet been written with <code>spssCommitHeader</code>

Example

See `spssWholeCaseIn`.

See also `spssWholeCaseIn`, `spssWholeCaseOut`.

spssGetCaseWeightVar

```
int spssGetCaseWeightVar (int handle, const char *varName)
```

Description

This function reports the name of the case weight variable. The name is copied to the buffer pointed to by *varName* as a null-terminated string. Since a variable name can be up to 64 bytes in length, the size of the buffer must be at least 65.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varName</i>	Pointer to the buffer to hold name of the case weight variable

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error.
SPSS_NO_CASEWGT	A case weight variable has not been defined for this file (warning).
SPSS_INVALID_HANDLE	The file handle is not valid.
SPSS_INVALID_CASEWGT	The given case weight variable is invalid. This error signals an internal problem in the implementation of the DLL and should never occur.

Example

```
#include <stdio.h>
#include "spssdio.h"
void func()
{
    int fH;                /* file handle          */
    int error;             /* error code           */
    char caseWeight[9];    /* case weight variable */
    ...
    error = spssOpenRead("bank.sav", &fH);
    ...
    /* Get and print the case weight variable of this file */
    error = spssGetCaseWeightVar(fH, caseWeight);
    if (error == SPSS_NO_CASEWGT)
        printf("The file is unweighted.\n");
    else if (error == SPSS_OK)
        printf("The case weight variable is: %s\n", caseWeight);
    else /* Handle error */
        ...
}
```

spssGetCompression

int spssGetCompression (int *handle*, int **compSwitch*)

Description

This function reports the compression attribute of an SPSS data file.

Parameter	Description
<i>handle</i>	Handle to the data file.
<i>compSwitch</i>	Pointer to compression attribute. Upon return, * <i>compSwitch</i> is 1 if the file is compressed; 0 otherwise.

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid

Example

```
#include <stdio.h>
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;              /* error code */
    int compSwitch;         /* compression switch */
    ...
    error = spssOpenRead("bank.sav", &fH);
    ...
    /* Print whether the data file is compressed. */
    error = spssGetCompression(fH, &compSwitch);
    if (error == SPSS_OK)
    {
        printf("File is ");
        if (compSwitch)
            printf("compressed.\n");
        else
            printf("uncompressed.\n");
    }
}
```

spssGetDateVariables

```
int spssGetDateVariables (int handle, int *numofElements, long **dateInfo)
```

Description

This function reports the Trends date variable information, if any, in an SPSS data file. It places the information in a dynamically allocated long array, sets **numofElements* to the number of elements in the array, and sets **dateInfo* to point to the array. The caller is expected to free the array by calling *spssFreeDateVariables* when it is no longer needed. The variable information is copied directly from record 7, subtype 3. Its first six elements comprise the “fixed” information, followed by a sequence of one or more three-element groups.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>numofElements</i>	Number of elements in allocated array
<i>dateInfo</i>	Pointer to first element of the allocated array

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_NO_DATEINFO	There is no Trends date variable information in the file (warning)
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_NO_MEMORY	Insufficient memory

Example

```
#include <stdio.h>
#include <stdlib.h>
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int numD;              /* number of elements */
    long *dateInfo;        /* pointer to date variable info. */
    ...
    error = spssOpenRead("bank.sav", &fH);
    ...
    /* Get & print TRENDS date variables info. */
    error = spssGetDateVariables(fH, &numD, &dateInfo);
    if (error == SPSS_NO_DATEINFO)
        printf("No TRENDS information.\n");
    else if (error == SPSS_OK)
    {
        if (numD < 6 || numD%3 != 0)
        {
            /* Should never happen */
            printf("Date info format error.\n");
            free(dateInfo);
            return;
        }
        /*Print the first six elements followed by groups of three */
        ...
        /* Remember to free array */
        spssFreeDateVariables(dateInfo);
    }
    ...
}
```

See also **spssSetDateVariables**, **spssFreeDateVariables**.

spssGetDEWFirst

int spssGetDEWFirst (const int *handle*, void **pData*, const long *maxData*, long **nData*)

Description

The client can retrieve DEW information (file information that is private to the SPSS Data Entry product) from a file in whatever increments are convenient. The first such increment is retrieved by calling spssGetDEWFirst, and subsequent segments are retrieved by calling spssGetDEWNext as many times as necessary. As with spssGetDEWInfo, spssGetDEWFirst will return SPSS_NO_DEW if the file was written with a byte order that is the reverse of that of the host.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>pData</i>	Returned as data from the file
<i>maxData</i>	Maximum bytes to return
<i>nData</i>	Returned as number of bytes returned

Returns

Returns one of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_NO_DEW	File contains no DEW information (warning)
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_FILE_BADTEMP	Error accessing the temporary file

See also **spssGetDEWInfo**, **spssGetDEWNext**.

spssGetDEWGUID

int spssGetDEWGUID (const int *handle*, char* *asciiGUID*)

Description

Data Entry for Windows maintains a GUID in character form as a uniqueness indicator. Two files have identical dictionaries and DEW information if they have the same GUID. Note that the `spssOpenWriteCopy` function will not copy the source file's GUID. `spssGetDEWGUID` allows the client to read a file's GUID, if any. The client supplies a 257 byte string in which the null-terminated GUID is returned.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>asciiGUID</i>	Returned as the file's GUID in character form or a null string if the file contains no GUID

Returns

The GUID is returned as a null-terminated string in parameter `asciiGUID`. If the file does not contain a GUID (and most do not), a null string is returned. When a null string is returned, the function result will still be `SPSS_OK`.

Error Code	Description
<i>SPSS_OK</i>	No error
<i>SPSS_INVALID_HANDLE</i>	The file handle is not valid

See also `spssSetDEWGUID`.

spssGetDEWInfo

```
int spssGetDEWInfo (const int handle, long *pLength, long *pHashTotal)
```

Description

This function can be called before actually retrieving DEW information (file information that is private to the SPSS Data Entry product) from a file, to obtain some attributes of that information—specifically its length in bytes and its hash total. The hash total is, by convention, contained in the last four bytes to be written. Because it is not cognizant of the structure of the DEW information, the I/O DLL is unable to correct the byte order of numeric information generated on a foreign host. As a result, the

DEW information is discarded if the file has a byte order that is the reverse of that of the host, and calls to `spssGetDEWInfo` will return `SPSS_NO_DEW`.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>pLength</i>	Returned as the length in bytes
<i>pHashTotal</i>	Returned as the hash total

Returns

Returns one of the following codes. Success is indicated by zero (`SPSS_OK`), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
<code>SPSS_OK</code>	No error
<code>SPSS_INVALID_HANDLE</code>	The file handle is not valid
<code>SPSS_NO_DEW</code>	File contains no DEW information (warning)

spssGetDEWNext

`int spssGetDEWNext (const int handle, void *pData, const long maxData, long *nData)`

Description

The client can retrieve DEW information (file information that is private to the SPSS Data Entry product) from a file in whatever increments are convenient. The first such increment is retrieved by calling `spssGetDEWFirst`, and subsequent segments are retrieved by calling `spssGetDEWNext` as many times as necessary. As with `spssGetDEWInfo`, `spssGetDEWFirst` will return `SPSS_NO_DEW` if the file was written with a byte order that is the reverse of that of the host.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>pData</i>	Returned as data from the file
<i>maxData</i>	Maximum bytes to return
<i>nData</i>	Returned as number of bytes returned

Returns

Returns one of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_DEW_NOFIRST	spssGetDEWFirst was never called
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_FILE_BADTEMP	Error accessing the temporary file

See also **spssGetDEWInfo**, **spssGetDEWFirst**.

spssGetEstimatedNofCases

```
int spssGetEstimatedNofCases(const int handle, long *caseCount)
```

Description

Although not strictly required for direct access input, this function helps in reading SPSS data files from releases earlier than 6.0. Some of these data files did not contain number of cases information, and **spssGetNumberOfCases** will return –1 cases. This function will return a precise number for uncompressed files and an estimate (based on overall file size) for compressed files. It cannot be used on files open for appending data.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>caseCount</i>	Returned as estimated <i>n</i> of cases

Returns

Returns one of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_WRMODE	The file is open for writing, not reading
SPSS_FILE_ERROR	Error reading the file

See also `spssGetNumberOfCases`.

spssGetFileAttributes

```
int spssGetFileAttributes(  
    const int hFile,  
    char*** attribNames,  
    char*** attribText,  
    int* nAttributes)
```

Description

This function returns all the datafile attributes. It allocates the memory necessary to hold the attribute names and values. For subscripted attributes, the names include the unit origin subscripts enclosed in square brackets, for example `Prerequisite[11]`. The acquired memory must be released by calling `spssFreeAttributes`.

Parameter	Description
<i>hFile</i>	handle to the data file
<i>attribNames</i>	Returned as a pointer to a vector of attribute names
<i>attribText</i>	Returned as a pointer to a vector of attribute values
<i>nAttributes</i>	Returned as the number of element in each vector

Returns

Returns one of the following codes. Success is indicated by zero (`SPSS_OK`), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error

SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_NO_MEMORY	Insufficient memory for the vectors

spssGetIdString

int spssGetIdString (int *handle*, char **id*)

Description

This function copies the file label of the SPSS data file associated with *handle* into the buffer pointed to by *id*. The label is at most 64 characters long and null-terminated. Thus, the size of the buffer should be at least 65. If an input data file is associated with the handle, the label will be exactly 64 characters long, padded with blanks as necessary.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>id</i>	File label buffer

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid

Example

```
#include <stdio.h>
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;             /* error code  */
    char id[65];           /* file label  */
    ...
    error = spssOpenRead("bank.sav", &fH);
    ...
    error = spssGetIdString(fH, id);
    if (error == SPSS_OK)
        printf("File label: %s\n", id);
    ...
}
```

spssGetMultRespCount

```
int spssGetMultRespCount(
    const int hFile,
    int* nSets)
```

Description

This function obtains a count of the number of multiple response sets stored in the dictionary.

Parameter	Description
<i>hFile</i>	Handle to the data file
<i>nSets</i>	Returned as the number of sets

Returns

The function returns SPSS_OK or an error value:

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is invalid
SPSS_OPEN_WRMODE	The file is not open for input

spssGetMultRespDefByIndex

```
int spssGetMultRespDefByIndex(
    const int hFile,
    const int iSet,
    spssMultRespDef** ppSet)
```

Description

This function obtains a description of a single multiple response set. The set is specified via a zero origin index, and the description is returned in a struct for which the memory is allocated by the function.

Parameter	Description
<i>hFile</i>	Handle to the data set
<i>iSet</i>	Zero origin index of the set
<i>ppSet</i>	Returned as a pointer to the set's description

For information on the set description struct, see “spssAddMultRespDefExt” on page 123. The memory for the struct must be freed by calling spssFreeMultRespDefStruct.

Returns

The function returns SPSS_OK or an error code.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is invalid
SPSS_OPEN_WRMODE	The file is not open for input
SPSS_INVALID_MRSETINDEX	The index is out of range
SPSS_NO_MEMORY	Insufficient memory to allocate the struct

spssGetMultRespDefs

```
int spssGetMultRespDefs (const int handle, char **mrespDefs)
```

Description

This function retrieves the definitions from an SPSS data file. The definitions are stored as a null-terminated ASCII string that is very similar to that containing the variable set definitions. The memory allocated by this function to contain the string must be freed by calling `spssFreeMultRespDefs`. If the file contains no multiple response definitions, **mrespDefs* is set to NULL, and the function returns the warning code `SPSS_NO_MULTRESP`.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>mrespDefs</i>	Returned as a pointer to a string

Returns

Returns one of the following codes. Success is indicated by zero (`SPSS_OK`), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
<code>SPSS_OK</code>	No error
<code>SPSS_NO_MULTRESP</code>	No definitions on the file (warning)
<code>SPSS_INVALID_HANDLE</code>	The file handle is not valid
<code>SPSS_NO_MEMORY</code>	Insufficient memory to contain the string

See also `spssFreeMultRespDefs`.

spssGetNumberOfCases

`int spssGetNumberOfCases (int handle, long *numofCases)`

Description

This function reports the number of cases present in a data file open for reading.

Parameter	Description
<i>handle</i>	Handle to the data file

numofCases Pointer to number of cases

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_WRMODE	File is open for writing, not reading

Example

```
#include <stdio.h>
#include "spssdio.h"
void func()
{
    int fH;                /* file handle      */
    int error;             /* error code       */
    long count;            /* Number of cases  */
    ...
    error = spssOpenRead("bank.sav", &fH);
    ...
    /* Get & print the number of cases present in the file. */
    error = spssGetNumberOfCases(fH, &count);
    if (error == SPSS_OK)
        printf("Number of cases: %ld\n");
    ...
}
```

spssGetNumberOfVariables

int spssGetNumberOfVariables (int *handle*, long **numVars*)

Description

This function reports the number of variables present in a data file.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>numVars</i>	Pointer to number of variables

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_DICT_NOTCOMMIT	Dictionary has not been committed
SPSS_INVALID_FILE	Data file contains no variables

Example

```
#include <stdio.h>
#include "spssdio.h"
void func()
{
    int fH;                /* file handle          */
    int error;             /* error code           */
    long count;            /* Number of variables */
    ...
    error = spssOpenRead("bank.sav", &fH);
    ...
    /* Get & print the number of variables present in the file. */
    error = spssGetNumberOfVariables(fH, &count);
    if (error == SPSS_OK)
        printf("Number of variables: %ld\n");
    ...
}
```

spssGetReleaseInfo

```
int spssGetReleaseInfo (int handle, int relinfo[])
```

Description

This function reports release- and machine-specific information about the file associated with *handle*. The information consists of an array of eight int values copied from record type 7, subtype 3 of the file, and is useful primarily for debugging. The array elements are, in order, release number (index 0), release subnumber (1), special release identifier number (2), machine code (3), floating-point representation code (4), compression scheme code (5), big/little-endian code (6), and character representation code (7).

Parameter	Description
<i>handle</i>	Handle to the data file.
<i>relinfo</i>	Array of int in which release- and machine-specific data will be stored. This array must have at least eight elements.

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values (with one exception noted below).

Error Code	Description
SPSS_OK	No error.
SPSS_INVALID_HANDLE	The file handle is not valid.
SPSS_NO_TYPE73	There is no type 7, subtype 3 record present. This code should be regarded as a warning even though it is positive. Files without this record are valid.

Example

```
#include <stdio.h>
#include "spssdio.h"
void func()
{
    int fH;                /* file handle      */
    int error;              /* error code       */
    int relInfo[8];         /* release info     */
    ...
    error = spssOpenRead("bank.sav", &fH);
    ...
    /* Get & print release and machine-specific info. */
    error = spssGetReleaseInfo(fH, relInfo);
    if (error == SPSS_OK)
    {
        printf("Release & machine information:\n");
        int i;
        for (i = 0; i < 8; ++i)
            printf("  Element %d: %d\n", i, relInfo[i]);
    }
    ...
}
```

spssGetSystemString

int spssGetSystemString (int *handle*, char **sysName*)

Description

This function returns the name of the system under which the file was created. It is a 40-byte blank-padded character field corresponding to the last 40 bytes of record type 1. Thus, in order to accommodate the information, the parameter *sysName* must be at least 41 bytes in length plus the terminating null character.

Parameter	Description
<i>handle</i>	Handle to the data file

sysName The originating system name

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid

Example

```
#include <stdio.h>
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;             /* error code */
    char sysName[41];      /* originating system */
    ...
    error = spssOpenRead("bank.sav", &fH);
    ...
    error = spssGetIdString(fH, sysName);
    if (error == SPSS_OK)
        printf("Originating System: %s\n", sysName);
    ...
}
```

spssGetTextInfo

int spssGetTextInfo (int *handle*, char **textInfo*)

Description

This function places the text data created by TextSmart as a null-terminated string in the user-supplied buffer *textInfo*. The buffer is assumed to be at least 256 characters long; the text data may be up to 255 characters long. If text data are not present in the file, the first character in *textInfo* is set to NULL.

Parameter	Description
<i>handle</i>	Handle to the data file

textInfo Buffer for text data

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid

spssGetTimeStamp

int spssGetTimeStamp (int *handle*, char **fileDate*, char **fileTime*)

Description

This function returns the creation date of the file as recorded in the file itself. The creation date is a null-terminated 9-byte character field in dd mmm yy format (27 Feb 96), and the receiving field must be at least 10 bytes in length. The creation time is a null-terminated 8-byte character field in hh:mm:ss format (13:12:15), and the receiving field must be at least 9 bytes in length.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>fileDate</i>	File creation date
<i>fileTime</i>	File creation time

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error

SPSS_INVALID_HANDLE

The file handle is not valid

spssGetValueChar

```
int spssGetValueChar (int handle, double varHandle, char *value, int valueSize)
```

Description

This function gets the value of a string variable for the current case, which is the case read by the most recent call to `spssReadCaseRecord`. The value is returned as a null-terminated string in the caller-provided buffer *value*; the length of the string is the length of the string variable. The argument *valueSize* is the allocated size of the buffer *value*, which must be at least the length of the variable plus 1.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varHandle</i>	Handle of the variable
<i>value</i>	Buffer for the value of the string variable
<i>valueSize</i>	Size of <i>value</i>

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error.
SPSS_INVALID_HANDLE	The file handle is not valid.
SPSS_OPEN_WRMODE	File is open for writing, not reading.
SPSS_INVALID_CASE	Current case is not valid. This may be because no <code>spssReadCaseRecord</code> calls have been made yet or because the most recent call failed with error or encountered the end of file.
SPSS_STR_EXP	Variable associated with the handle is numeric.

SPSS_BUFFER_SHORT

Buffer *value* is too short to hold the value.

Example

```

#include <stdio.h>
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;             /* error code */
    int numV;              /* number of variables */
    int *typesV;           /* variable types */
    char **namesV;         /* variable names */
    double handlesV[100]; /* assume no more than 100 variables */
    char cValue[256];      /* long enough for any string variable */
    long nCases;           /* number of cases */
    long casesPrint;       /* number of cases to print */
    long case;             /* case index */
    double nValue;         /* numeric value */
    int i;                /* variable index */
    ...
    error = spssOpenRead("bank.sav", &fH);
    ...
    /* Get variable names and types */
    error = spssGetVarNames(fH, &numV, &namesV, &typesV);
    ...
    if (numV > 100)
    {
        printf("Too many variables; increase program capacity.\n");
        spssFreeVarNames(namesV, typesV, numV);
        return;
    }
    /* Get & store variable handles */
    for (i = 0; i < numV; ++i)
    {
        error = spssGetVarHandle(fH, namesV[i], &handlesV[i]);
        if (error != SPSS_OK) ...
    }
    /* Get the number of cases */
    error = spssGetNumberOfCases(fH, &nCases);
    ...
    /* Print at most the first ten cases */
    casesPrint = (nCases < 10) ? nCases : 10;
    for (case = 1; case <= casesPrint; ++case)
    {
        error = spssReadCaseRecord(fH);
        ...
        printf("Case %ld\n", case);
        for (i = 0; i < numV; ++i)
        {
            if (typesV[i] == 0)
            {
                /* Numeric */
                error = spssGetValueNumeric(fH, handlesV[i], &nValue);
                if (error == SPSS_OK)
                    printf("    %ld\n", nValue);
                else ...
            }
            else
            {
                /* String */
                error = spssGetValueChar(fH, handlesV[i], cValue, 256);
                if (error == SPSS_OK)
                    printf("    %s\n", cValue);
                else ...
            }
        }
    }
}

```

```
/* Free the variable names & types */
    spssFreeVarNames(namesV, typesV, numV);
}
```

spssGetValueNumeric

```
int spssGetValueNumeric (int handle, double varHandle, double *value)
```

Description

This function gets the value of a numeric variable for the current case, which is the case read by the most recent call to `spssReadCaseRecord`.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varHandle</i>	Handle to the variable
<i>value</i>	Pointer to the value of the numeric variable

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error.
SPSS_INVALID_HANDLE	The file handle is not valid.
SPSS_OPEN_WRMODE	File is open for writing, not reading.
SPSS_INVALID_CASE	Current case is not valid. This may be because no <code>spssReadCaseRecord</code> calls have been made yet or because the most recent call failed with error or encountered the end of file.
SPSS_NUME_EXP	Variable associated with the handle is not numeric.

Example

See `spssGetValueChar`.

spssGetVarAttributes

```
int spssGetVarAttributes(  
    const int hFile,  
    const char* varName,  
    char*** attribNames,  
    char*** attribText,  
    int* nAttributes)
```

Description

This function is analogous to `spssGetFileAttributes`. It returns all the attributes for a single variable.

Parameter	Description
<i>hFile</i>	Handle to the data file
<i>varName</i>	The name of the variable
<i>attribNames</i>	Returned as a pointer to a vector of attribute names
<i>attribText</i>	Returned as a pointer to a vector of attribute values
<i>nAttributes</i>	Returned as the number of element in each vector

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_VAR_NOTFOUND	Named variable is not in the file
SPSS_NO_MEMORY	Insufficient memory for the vectors

spssGetVarAlignment

```
int spssGetVarAlignment (int handle, const char *varName, int *alignment)
```

Description

This function reports the value of the alignment attribute of a variable.

Parameter	Description
<i>handle</i>	Handle to the data file.
<i>varName</i>	Variable name.
<i>alignment</i>	Pointer to alignment. Set to SPSS_ALIGN_LEFT, SPSS_ALIGN_RIGHT, or SPSS_ALIGN_CENTER.

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist

spssGetVarCMissingValues

```
int spssGetVarCMissingValues
(int handle, const char *varName, int *missingFormat,
char *missingVal1, char *missingVal2, char *missingVal3)
```

Description

This function reports the missing values of a short string variable. The value of **missingFormat* will be in the range 0–3, indicating the number of missing values. The appropriate number of missing values is copied to the buffers *missingVal1*, *missingVal2*, and *missingVal3*. The lengths of the null-terminated missing value strings will be the length of the short string variable in question. Since the latter can be at most 8 characters long, 9-character buffers are adequate for any short string variable.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varName</i>	Variable name
<i>missingFormat</i>	Pointer to missing value format code
<i>missingVal1</i>	Buffer for first missing value
<i>missingVal2</i>	Buffer for second missing value
<i>missingVal3</i>	Buffer for third missing value

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist
SPSS_STR_EXP	The variable is numeric
SPSS_SHORTSTR_EXP	The variable is a long string (length > 8)

Example

```

#include <stdio.h>
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;             /* error code */
    int type;              /* missing format type */
    int numV;              /* number of variables */
    int *typesV;           /* variable types */
    char **namesV;         /* variable names */
    char cMiss1[9];        /* first missing value */
    char cMiss2[9];        /* second missing value */
    char cMiss3[9];        /* third missing value */

    ...
    error = spssOpenRead("bank.sav", &fH);
    ...
    /* Print missing value information for all short string      ** variables
*/
    error = spssGetVarNames(fH, &numV, &namesV, &typesV);
    if (error == SPSS_OK)
    {
        int i;
        for (i = 0; i < numV; ++i)
        {
            if (0 < typesV[i] && typesV[i] <= 8)
            {
                /* Short string variable */
                error = spssGetVarCMissingValues
                    (fH, namesV[i], &type, cMiss1, cMiss2, cMiss3);
                if (error != SPSS_OK) continue; /* Ignore error */
                printf("Variable %s, missing values: ", namesV[i]);
                switch (type)
                {
                    case 0:
                        printf("None\n");
                        break;
                    case 1:
                        printf("%s\n", cMiss1);
                        break;
                    case 2:
                        printf("%s, %s\n", cMiss1, cMiss2);
                        break;
                    case 3:
                        printf("%s, %s, %s\n", cMiss1, cMiss2, cMiss3);
                        break;
                    default: /* Should never come here */
                        printf("Invalid format code\n");
                        break;
                }
            }
        }
        spssFreeVarNames(namesV, typesV, numV);
    }
}

```

See also **spssGetVarNMissingValues**.

spssGetVarColumnWidth

int spssGetVarColumnWidth (int *handle*, const char **varName*, int **columnWidth*)

Description

This function reports the value of the column width attribute of a variable. A value of zero is special and means that the SPSS Data Editor, which is the primary user of this attribute, will set an appropriate width using its own algorithm.

Parameter	Description
<i>handle</i>	Handle to the data file.
<i>varName</i>	Variable name.
<i>columnWidth</i>	Pointer to column width. Non-negative.

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist

spssGetVarCompatName

int spssGetVarCompatName (const int handle, const char* longName, char* shortName)

Description

When writing an SPSS data file, the I/O DLL creates variable names which are compatible with legacy releases of SPSS. These names are no more than 8 bytes in length, are all upper case, and are unique within the file. spssGetVarCompatName

allows access to these "mangled" name for input files and for output files after `spssCommitHeader` has been called.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>longName</i>	The variable's extended name as a null-terminated string
<i>shortName</i>	A 9 byte character variable to receive the mangled name as a null-terminate string

Returns

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_DICT_NOTCOMMIT	<code>spssCommitHeader</code> has not been called for an output file
SPSS_VAR_NOTFOUND	Variable <i>longName</i> does not exist

spssGetVarCValueLabel

`int spssGetVarCValueLabel`
(`int handle`, `const char *varName`, `const char *value`, `char *label`)

Description

This function gets the value label for a given value of a short string variable. The label is copied as a null-terminated string into the buffer *label*, whose size must be at least 61 to hold the longest possible value label (60 characters plus the null terminator). To get value labels more than 60 characters long, use the `spssGetVarCValueLabelLong` function.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varName</i>	Variable name
<i>value</i>	Short string value for which the label is wanted

label Label for the value

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_NO_LABELS	The variable has no labels (warning)
SPSS_NO_LABEL	There is no label for the given value (warning)
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist
SPSS_STR_EXP	The variable is numeric
SPSS_SHORTSTR_EXP	The variable is a long string (length > 8)
SPSS_EXC_STRVALUE	The value is longer than the length of the variable

Example

```
#include <stdio.h>
#include "spssdio.h"
void func()
{
    int fH;                /* file handle          */
    int error;             /* error code           */
    char vLab[61];         /* label for the value  */
    ...
    error = spssOpenRead("myfile.sav", &fH);
    ...
    /* Get and print the label for value "IL" of variable STATE */
    error = spssGetVarCValueLabel(fH, "STATE", "IL", vLab);
    if (error == SPSS_OK)
        printf("Value label for variable STATE, value \"IL\": %s\n", vLab);
    ...
}
```

spssGetVarCValueLabelLong

```
int spssGetVarCValueLabelLong
(int handle, const char *varName, const char *value, char *labelBuff,
int lenBuff, int *lenLabel)
```

Description

This function returns a null-terminated value label corresponding to one value of a specified variable whose values are short strings. The function permits the client to limit the number of bytes (including the null terminator) stored and returns the number of data bytes (excluding the null terminator) actually stored. If an error is detected, the label is returned as a null string, and the length is returned as 0.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varname</i>	Null-terminated variable name
<i>value</i>	Null-terminated value for which label is requested
<i>labelBuff</i>	Returned as null-terminated label
<i>lenBuff</i>	Overall size of <i>labelBuff</i> in bytes
<i>lenLabel</i>	Returned as bytes stored excluding terminator

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_NO_LABELS	The variable has no labels (warning)
SPSS_NO_LABEL	The given value has no label (warning)
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist
SPSS_STR_EXP	The specified variable has numeric values
SPSS_SHORTSTR_EXP	The specified variable has long string values
SPSS_EXC_STRVALUE	The specified value is longer than the variable's data

spssGetVarCValueLabels

```
int spssGetVarCValueLabels
(int handle, const char *varName, char ***values, char ***labels, int *numLabels)
```

Description

This function gets the set of labeled values and associated labels for a short string variable. The number of values is returned as **numLabels*. Values are stored into an array of **numLabels* pointers, each pointing to a char string containing a null-terminated value, and **values* is set to point to the first element of the array. Each value string is as long as the variable. The corresponding labels are structured as an array of **numLabels* pointers, each pointing to a char string containing a null-terminated label, and **labels* is set to point to the first element of the array.

The two arrays and the value and label strings are allocated on the heap. When they are no longer needed, *spssFreeVarCValueLabels* should be called to free the memory.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varName</i>	Variable name
<i>values</i>	Pointer to array of pointers to values
<i>labels</i>	Pointer to array of pointers to labels
<i>numLabels</i>	Pointer to number of values or labels

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_NO_LABELS	The variable has no labels (warning)
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist

SPSS_STR_EXP	The variable is numeric
SPSS_SHORTSTR_EXP	The variable is a long string (length > 8)
SPSS_NO_MEMORY	Insufficient memory

Example

```
#include <stdio.h>
#include "spssdio.h"
void func()
{
    int fH; /* file handle */
    int error; /* error code */
    int numL; /* number of values or labels */
    char **cValuesL; /* values */
    char **labelsL; /* labels */
    ...
    error = spssOpenRead("myfile.sav", &fH);
    ...
    /* Get and print value labels for short string variable STATE */
    error = spssGetVarCValueLabels(fH, "STATE",
        &cValuesL, &labelsL, &numL);
    if (error == SPSS_OK)
    {
        int i;
        printf("Value labels for STATE\n");
        for (i = 0; i < numL; ++i)
        {
            printf("Value: %s, Label: %s\n", cValuesL[i], labelsL[i]);
        }
        /* Free the values & labels */
        spssFreeVarCValueLabels(cValuesL, labelsL, numL);
    }
}
```

See also `spssFreeVarCValueLabels`.

spssGetVarHandle

```
int spssGetVarHandle (int handle, const char *varName, double *varHandle)
```

Description

This function returns a handle for a variable, which can then be used to read or write (depending on how the file was opened) values of the variable. If *handle* is associated with an output file, the dictionary must be written with `spssCommitHeader` before variable handles can be obtained via `spssGetVarHandle`.

Parameter	Description
-----------	-------------

<i>handle</i>	Handle to the data file.
<i>varName</i>	Variable name.
<i>varHandle</i>	Pointer to handle for the variable. Note that the variable <i>handle</i> is a double, and not int or long.

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_DICT_NOTCOMMIT	Dictionary of the output file has not yet been written with <code>spssCommitHeader</code>
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist
SPSS_NO_MEMORY	No memory available

Example

See `spssGetValueChar`.

spssGetVariableSets

```
int spssGetVariableSets (int handle, char **varSets)
```

Description

This function reports the variable sets information in the data file. Variable sets information is stored in a null-terminated string and a pointer to the string is returned in **varSets*. Since the variable sets string is allocated on the heap, the caller should free it by calling `spssFreeVariableSets` when it is no longer needed.

Parameter	Description
-----------	-------------

<i>handle</i>	Handle to the data file
<i>varSets</i>	Pointer to pointer to variable sets string

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_NO_VARSETS	There is no variable sets information in the file (warning)
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_NO_MEMORY	Insufficient memory

Example

```
#include <stdio.h>
#include <stdlib.h>
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;              /* error code */
    char *vSets;           /* ptr to variable sets info.*/
    ...
    error = spssOpenRead("bank.sav", &fH);
    ...
    /* Get & print variable sets information. */
    error = spssGetVariableSets(fH, &vSets);
    if (error == SPSS_NO_VARSETS)
    {
        printf("No variable sets information in file.\n");
    }
    else if (error == SPSS_OK)
    {
        /* In real life, we would format the variable sets
        ** information better
        */
        printf("Variable sets:\n%s", vSets);
        /* Remember to free variable set string */
        spssFreeVariableSets(vSets);
    }
    ...
}
```

See also **spssFreeVariableSets**.

spssGetVarInfo

int spssGetVarInfo (int *handle*, int *iVar*, char **varName*, int **varType*)

Description

This function gets the name and type of one of the variables present in a data file. It serves the same purpose as spssGetVarNames but returns the information one variable at a time and, therefore, can be passed to a Visual Basic program. The storage to receive the variable name must be at least 65 bytes in length because the name is returned as a null-terminated string. The type code is an integer in the range 0–32767, 0 indicating a numeric variable and a positive value indicating a string variable of that size.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>iVar</i>	Zero-origin variable number
<i>varName</i>	Returned as the variable name
<i>varType</i>	Returned as the variable type

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_INVALID_FILE	The data file contains no variables
SPSS_NO_MEMORY	Insufficient memory
SPSS_VAR_NOTFOUND	Parameter <i>iVar</i> is invalid

Example

```

#include <stdio.h>
#include "spssdio.h"
void func()
{
    int fH;                /* file handle          */
    int error;             /* error code           */
    long count;            /* number of variables  */
    int *typeV;            /* variable type        */
    char *nameV;           /* variable name        */
    ...
    error = spssOpenRead("bank.sav", &fH);
    ...
    /* Get number of variables */
    error = spssGetNumberOfVariables(fH, &count);
    if (error == SPSS_OK)
    /* Get & print variable names and types */
    {
        int i;
        for (i = 0; i < count; ++i)
        {error = spssGetVarInfo(fH, i, nameV, typeV);
        if (error == SPSS_OK)
            printf("Variable name: %s, type: %d\n", nameV, typeV);
        }
    }
}

```

spssGetVarLabel

int spssGetVarLabel (int *handle*, const char **varName*, char **varLabel*)

Description

This function copies the label of variable *varName* into the buffer pointed to by *varLabel*. Since the variable label is at most 120 characters long and null-terminated, the size of the buffer should be at least 121. To get labels more than 120 characters long, use the spssGetVarLabelLong function.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varName</i>	Variable name
<i>varLabel</i>	Variable label buffer

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_NO_LABEL	The variable does not have a label (warning)
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist

Example

```
#include <stdio.h>
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;             /* error code */
    char vLabel[121];      /* variable label */
    ...
    error = spssOpenRead("bank.sav", &fH);
    ...
    /* Get and print the label of the variable AGE */
    error = spssGetVarLabel(fH, "AGE", vLabel);
    if (error == SPSS_OK)
        printf("Variable label of AGE: %s\n", vLabel);
    ...
}
```

spssGetVarLabelLong

```
int spssGetVarLabelLong (int handle, const char *varName, char *labelBuff,
int lenBuff, int *lenLabel)
```

Description

This function returns the null-terminated label associated with the specified variable but restricts the number of bytes (including the null terminator) returned to *lenBuff* bytes. This length can be conveniently specified as *sizeof(labelBuff)*. The function also

returns the number of data bytes (this time excluding the null terminator) stored. If an error is detected, the label is returned as a null string, and the length is returned as 0.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varName</i>	Null-terminated variable name
<i>labelBuff</i>	Buffer to receive the null-terminated label
<i>lenBuff</i>	Overall size of <i>labelBuff</i> in bytes
<i>lenLabel</i>	Returned as bytes stored excluding terminator

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_NO_LABEL	The variable does not have a label (warning)
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist

spssGetVarMeasureLevel

int spssGetVarMeasureLevel (int *handle*, const char **varName*, int **measureLevel*)

Description

This function reports the value of the measurement level attribute of a variable.

Parameter	Description
<i>handle</i>	Handle to the data file.
<i>varName</i>	Variable name.

measureLevel Pointer to measurement level. Set to SPSS_MLVL_NOM, SPSS_MLVL_ORD, or SPSS_MLVL_RAT, for nominal, ordinal, and scale (ratio), respectively.

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist

spssGetVarNMissingValues

```
int spssGetVarNMissingValues
(int handle, const char *varName, int *missingFormat,
double *missingVal1, double *missingVal2, double *missingVal3)
```

Description

This function reports the missing values of a numeric variable. The value of **missingFormat* determines the interpretation of **missingVal1*, **missingVal2*, and **missingVal3*. If **missingFormat* is SPSS_MISS_RANGE, **missingVal1* and **missingVal2* represent the upper and lower limits, respectively, of the range, and **missingVal3* is not used. If **missingFormat* is SPSS_MISS_RANGEANDVAL, **missingVal1* and **missingVal2* represent the range and **missingVal3* is the discrete missing value. If **missingFormat* is neither of the above, it will be in the range 0–3, indicating the number of discrete missing values present. (The macros SPSS_NO_MISSVAL, SPSS_ONE_MISSVAL, SPSS_TWO_MISSVAL, and SPSS_THREE_MISSVAL may be used as synonyms for 0–3.)

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varName</i>	Variable name

<i>missingFormat</i>	Pointer to missing value format code
<i>missingVal1</i>	Pointer to first missing value
<i>missingVal2</i>	Pointer to second missing value
<i>missingVal3</i>	Pointer to third missing value

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist
SPSS_NUME_EXP	The variable is not numeric

Example

```

#include <stdio.h>
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;             /* error code */
    int type;              /* missing format type */
    int numV;              /* number of variables */
    int *typesV;           /* variable types */
    char **namesV;         /* variable names */
    double nMiss1;         /* first missing value */
    double nMiss2;         /* second missing value */
    double nMiss3;         /* third missing value */

    ...

    error = spssOpenRead("bank.sav", &fH);
    ...
    /*Print missing value information for all numeric variables */
    error = spssGetVarNames(fH, &numV, &namesV, &typesV);
    if (error == SPSS_OK)
    {
        int i;
        for (i = 0; i < numV; ++i)
        {
            if (typesV[i] == 0)
            {
                /* Numeric variable */
                error = spssGetVarNMissingValues
                    (fH, namesV[i], &type, &nMiss1, &nMiss2, &nMiss3);
                if (error != SPSS_OK) continue; /* Ignore error */
                printf("Variable %s, missing values: ", namesV[i]);
                switch (type)
                {
                    case SPSS_MISS_RANGE:
                        printf("%e through %e\n", nMiss1, nMiss2);
                        break;
                    case SPSS_MISS_RANGEANDVAL:
                        printf("%e through %e, %e\n", nMiss1, nMiss2, nMiss3);
                        break;
                    case 0:
                        printf("None\n");
                        break;
                    case 1:
                        printf("%e\n", nMiss1);
                        break;
                    case 2:
                        printf("%e, %e\n", nMiss1, nMiss2);
                        break;
                    case 3:
                        printf("%e, %e, %e\n", nMiss1, nMiss2, nMiss3);
                        break;
                    default: /* Should never come here */
                        printf("Invalid format code\n");
                        break;
                }
            }
        }
        spssFreeVarNames(namesV, typesV, numV);
    }
}

```

See also `spssGetVarCMissingValues`.

spssGetVarNValueLabel

```
int spssGetVarNValueLabel
(int handle, const char *varName, double value, char *label)
```

Description

This function gets the value label for a given value of a numeric variable. The label is copied as a null-terminated string into the buffer *label*, whose size must be at least 61 to hold the longest possible value label (60 characters) plus the terminator. To get value labels more than 60 characters long, use the `spssGetVarNValueLabelLong` function.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varName</i>	Variable name
<i>value</i>	Numeric value for which the label is wanted
<i>label</i>	Label for the value

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_NO_LABELS	The variable has no labels (warning)
SPSS_NO_LABEL	There is no label for the given value (warning)
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist
SPSS_NUME_EXP	The variable is not numeric

Example

```
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;             /* error code */
    char vLab[61];         /* label for the value */
    ...
    error = spssOpenRead("bank.sav", &fH);
    ...
    /* Get and print the label for value 0.0 of variable SEX */
    error = spssGetVarNValueLabel(fH, "SEX", 0.0, vLab);
    if (error == SPSS_OK)
        printf("Value label for variable SEX, value 0.0: %s\n", vLab);
    ...
}
```

spssGetVarNValueLabelLong

```
int spssGetVarNValueLabelLong
(int handle, const char *varName, double value, char *labelBuff, int lenBuff, int *lenLabel)
```

Description

This function returns a null-terminated value label corresponding to one value of a specified numeric variable. It permits the client to limit the number of bytes (including the null terminator) stored and returns the number of data bytes (excluding the null terminator) actually stored. If an error is detected, the label is returned as a null string, and the length is returned as 0.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varName</i>	Null-terminated variable name
<i>value</i>	Value for which label is requested
<i>labelBuff</i>	Returned as null-terminated label
<i>lenBuff</i>	Overall size of <i>labelBuff</i> in bytes
<i>lenLabel</i>	Returned as bytes stored excluding terminator

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_NO_LABELS	The variable has no labels (warning)
SPSS_NO_LABEL	The given value has no label (warning)
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist
SPSS_NUME_EXP	The specified variable has string values

spssGetVarNValueLabels

```
int spssGetVarNValueLabels
(int handle, const char *varName, double **values, char ***labels, int *numLabels)
```

Description

This function gets the set of labeled values and associated labels for a numeric variable. The number of values is returned as **numLabels*. Values are stored into an array of **numLabels* double elements, and **values* is set to point to the first element of the array. The corresponding labels are structured as an array of **numLabels* pointers, each pointing to a char string containing a null-terminated label, and **labels* is set to point to the first element of the array.

The two arrays and the label strings are allocated on the heap. When they are no longer needed, `spssFreeVarNValueLabels` should be called to free the memory.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varName</i>	Variable name
<i>values</i>	Pointer to array of double values
<i>labels</i>	Pointer to array of pointers to labels
<i>numLabels</i>	Pointer to number of values or labels

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_NO_LABELS	The variable has no labels (warning)
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist
SPSS_NUME_EXP	The variable is not numeric
SPSS_NO_MEMORY	Insufficient memory

Example

```
#include <stdio.h>
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;             /* error code */
    int numL;              /* number of values or labels */
    double *nValuesL;      /* values */
    char **labelsL;        /* labels */
    ...
    error = spssOpenRead("bank.sav", &fH);
    ...
    /* Get and print value labels for numeric variable SEX */
    error = spssGetVarNValueLabels(fH, "SEX",
                                   &nValuesL, &labelsL, &numL);
    if (error == SPSS_OK)
    {
        int i;
        printf("Value labels for SEX\n");
        for (i = 0; i < numL; ++i)
        {
            printf("Value: %g, Label: %s\n", valuesL[i], labelsL[i]);
        }
        /* Free the values & labels */
        spssFreeVarNValueLabels(nValuesL, labelsL, numL);
    }
}
```

See also **spssFreeVarNValueLabels**.

spssGetVarNames

```
int spssGetVarNames (int handle, int *numVars, char ***varNames, int **varTypes)
```

Description

This function gets the names and types of all the variables present in a data file. The number of variables is returned as **numVars*. Variable names are structured as an array of **numVars* pointers, each pointing to a char string containing a variable name, and **varNames* is set to point to the first element of the array. Variable types are stored into a corresponding array of **numVars* in elements, and **varTypes* is set to point to the first element of the array. The type code is an integer in the range 0–32767, 0 indicating a numeric variable and a positive value indicating a string variable of that size.

The two arrays and the variable name strings are allocated on the heap. When they are no longer needed, *spssFreeVarNames* should be called to free the memory.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>numVars</i>	Pointer to number of variables
<i>varNames</i>	Pointer to array of pointers to variable names
<i>varTypes</i>	Pointer to array of variable types

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_INVALID_FILE	The data file contains no variables
SPSS_NO_MEMORY	Insufficient memory

Example

```

#include <stdio.h>
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;             /* error code */
    int numV;              /* number of variables */
    int *typesV;           /* variable types */
    char **namesV;         /* variable names */
    ...
    error = spssOpenRead("bank.sav", &fH);
    ...
    /* Get & print variable names and types */
    error = spssGetVarNames(fH, &numV, &namesV, &typesV);
    if (error == SPSS_OK)
    {
        int i;
        for (i = 0; i < numV; ++i)
        {
            printf("Variable name: %s, type: %d\n", namesV[i], typesV[i]);
        }
        /* Free the variable names & types */
        spssFreeVarNames(namesV, typesV, numV);
    }
}

```

See also **spssFreeVarNames**.

spssGetVarPrintFormat

```

int spssGetVarPrintFormat
(int handle, const char *varName, int *printType, int *printDec, int *printWid)

```

Description

This function reports the print format of a variable. Format type, number of decimal places, and field width are returned as **printType*, **printDec*, and **printWid*, respectively.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varName</i>	Variable name
<i>printType</i>	Pointer to print format type code (file <i>spssdio.h</i> defines macros of the form SPSS_FMT_... for all valid format type codes)
<i>printDec</i>	Pointer to number of digits after the decimal
<i>printWid</i>	Pointer to print format width

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist

Example

```
#include <stdio.h>
#include "spssdio.h"
void func()
{
    int  fH;                /* file handle          */
    int  error;             /* error code           */
    int  type;              /* print format type    */
    int  dec;               /* digits after decimal */
    int  wid;               /* print format width   */

    error = spssOpenRead("bank.sav", &fH);
    ...
    /* Get & print the print format of variable AGE */
    error = spssGetVarPrintFormat(fH, "AGE", &type, &dec, &wid);
    if (error == SPSS_OK)
    {
        printf("Variable AGE, format code %d, width.dec %d.%d\n",
               type, wid, dec);
    }
}
```

spssGetVarWriteFormat

int spssGetVarWriteFormat
(int *handle*, const char **varName*, int **writeType*, int **writeDec*, int **writeWid*)

Description

This function reports the write format of a variable. Format type, number of decimal places, and field width are returned as **writeType*, **writeDec*, and **writeWid*, respectively.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varName</i>	Variable name
<i>writeType</i>	Pointer to write format type code (file <i>spssdio.h</i> defines macros of the form SPSS_FMT_... for all valid format type codes)
<i>writeDec</i>	Pointer to number of digits after the decimal
<i>writeWid</i>	Pointer to write format width

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist

Example

```
#include <stdio.h>
#include "spssdio.h"
void func()
{
    int  fH;                /* file handle          */
    int  error;             /* error code           */
    int  type;              /* write format type    */
    int  dec;               /* digits after decimal */
    int  wid;               /* write format width   */

    error = spssOpenRead("bank.sav", &fH);
    ...
    /* Get & print the write format of variable AGE */
    error = spssGetVarWriteFormat(fH, "AGE", &type, &dec, &wid);
    if (error == SPSS_OK)
    {
        printf("Variable AGE, format code %d, width.dec %d.%d\n",
               type, wid, dec);
    }
}
```

spssHostSysmisVal

```
void spssHostSysmisVal(double *missVal)
```

Description

This function accesses the same information as `spssSysmisVal` but returns the information via a parameter rather than on the stack as the function result. The problem being addressed is that not all languages return doubles from functions in the same fashion.

Parameter	Description
<i>missval</i>	Returned as the system missing value

Returns

The function always succeeds, and there is no return code.

See also `spssSysmisVal`.

spssLowHighVal

```
void spssLowHighVal (double *lowest, double *highest)
```

Description

This function returns the “lowest” and “highest” values used for numeric missing value ranges on the host system. It may be called at any time.

Parameter	Description
<i>lowest</i>	Pointer to “lowest” value
<i>highest</i>	Pointer to “highest” value

Returns

None

Example

```

#include "spssdio.h"
void func()
{
    int    fH;                      /* file handle    */
    int    error;                   /* error code     */
    double lowest, highest;
    ...
    error = spssOpenWrite("data.sav", &fH);
    ...
    /* Create numeric variable SALARY and set range "lowest"
    ** through 0 as missing
    */
    error = spssSetVarName(fH, "SALARY", SPSS_NUMERIC);
    if (error == SPSS_OK)
    {
        spssLowHighVal(&lowest, &highest);
        /* Last arg. is a placeholder since we are defining a range
        ** only
        */
        error = spssSetVarNMissingValues(fH, "SALARY",
            SPSS_MISS_RANGE, lowest, 0.0, 0.0);
        ...
    }
}

```

spssOpenAppend

```
int spssOpenAppend (const char *fileName, int *handle)
```

Description

This function opens an SPSS data file for appending cases and returns a handle that should be used for subsequent operations on the file. (Note: this function will not work correctly on compressed data files created by SPSS systems prior to release 14.0.)

Parameter	Description
<i>fileName</i>	Name of the file
<i>handle</i>	Pointer to handle to be returned

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_FITAB_FULL	File table full (too many open SPSS data files)
SPSS_FILE_OERROR	Error opening file
SPSS_NO_MEMORY	Insufficient memory
SPSS_FILE_RERROR	Error reading file
SPSS_INVALID_FILE	File is not a valid SPSS data file
SPSS_NO_TYPE2	File is not a valid SPSS data file (no type 2 record)
SPSS_NO_TYPE999	File is not a valid SPSS data file (missing type 999 record)
SPSS_INCOMPAT_APPEND	File created on an incompatible system.

Example

```
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;             /* error code */
    ...
    error = spssOpenAppend("bank.sav", &fH);
    if (error == 0)
    {
        /* fH is a valid handle; process and */
        ...
        /* close file */
        error = spssCloseAppend(fH);
        ...
    }
    else
    {
        /* Handle error*/
        ...
    }
}
```

See also **spssCloseAppend**.

spssOpenRead

int spssOpenRead (const char **fileName*, int **handle*)

Description

This function opens an SPSS data file for reading and returns a handle that should be used for subsequent operations on the file.

Parameter	Description
<i>fileName</i>	Name of the file
<i>handle</i>	Pointer to handle to be returned

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_FITAB_FULL	File table full (too many open SPSS data files)
SPSS_FILE_OERROR	Error opening file
SPSS_NO_MEMORY	Insufficient memory
SPSS_FILE_RERROR	Error reading file
SPSS_INVALID_FILE	File is not a valid SPSS data file
SPSS_NO_TYPE2	File is not a valid SPSS data file (no type 2 record)
SPSS_NO_TYPE999	File is not a valid SPSS data file (missing type 999 record)

Example

```
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;             /* error code */
    ...
    error = spssOpenRead("bank.sav", &fH);
    if (error == 0)
    {
        /* fH is a valid handle; process and */
        ...
        /* close file */
        error = spssCloseRead(fH);
        ...
    }
    else
    {
        /* Handle error*/
        ...
    }
}
```

See also **spssCloseRead**.

spssOpenWrite

int spssOpenWrite (const char **filename*, int **handle*)

Description

This function opens a file in preparation for creating a new SPSS data file and returns a handle that should be used for subsequent operations on the file.

Parameter	Description
<i>filename</i>	Name of the data file
<i>handle</i>	Pointer to handle to be returned

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_FITAB_FULL	File table full (too many open SPSS data files)
SPSS_FILE_OERROR	Error opening file
SPSS_NO_MEMORY	Insufficient memory

Example

```
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;             /* error code */
    ...
    error = spssOpenWrite("dat.sav", &fH);
    if (error == 0)
    {
        /* fH is a valid handle; process and */
        ...
        /* close file */
        error = spssCloseWrite(fH);
        ...
    }
    else
    {
        /* Handle error*/
        ...
    }
}
```

See also **spssCloseWrite**.

spssOpenWriteCopy

int spssOpenWriteCopy (const char **fileName*, const char **dictFileName*, int **handle*)

Description

This function opens a file in preparation for creating a new SPSS data file and initializes its dictionary from that of an existing SPSS data file. It is useful when you want to modify the dictionary or data of an existing file or replace all of its data. The typical sequence of operations is to call **spssOpenWriteCopy** (*newFileName*, *oldFileName*, ...) to open a new file initialized with a copy of the old file's dictionary, then **spssOpenRead** (*oldFileName*, ...) to open the old file to access its data.

Parameter	Description
<i>fileName</i>	Name of the new file
<i>dictFileName</i>	Name of existing file
<i>handle</i>	Pointer to handle to be returned

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_FITAB_FULL	File table full (too many open SPSS data files)
SPSS_FILE_OERROR	Error opening new file for output
SPSS_NO_MEMORY	Insufficient memory
SPSS_FILE_RERROR	Error reading existing file
SPSS_INVALID_FILE	File is not a valid SPSS data file
SPSS_NO_TYPE2	File is not a valid SPSS data file (no type 2 record)
SPSS_NO_TYPE999	File is not a valid SPSS data file (missing type 999 record)

spssQueryType7

```
int spssQueryType7(const int handle, const int subType, int *bFound)
```

Description

This function can be used to determine whether a file opened for reading or append contains a specific “type 7” record. The following type 7 subtypes might be of interest:

Subtype 3. Release information

Subtype 4. Floating point constants including the system missing value

Subtype 5. Variable set definitions

Subtype 6. Date variable information

Subtype 7. Multiple response set definitions

Subtype 8. Data Entry for Windows (DEW) information

Subtype 10. TextSmart information

Subtype 11. Measurement level, column width, and alignment for each variable

Parameter	Description
<i>handle</i>	Handle to the data file
<i>subtype</i>	Specific subtype record
<i>bFound</i>	Returned set if the specified subtype was encountered

Returns

The result of the query is returned in parameter *bfound*—TRUE if the record subtype was encountered when reading the file's dictionary; FALSE otherwise.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_WRMODE	The file was opened for writing
SPSS_INVALID_7SUBTYPE	Parameter subtype not between 1 and MAX7SUBTYPE

spssReadCaseRecord

int spssReadCaseRecord (int *handle*)

Description

This function reads the next case from a data file into internal buffers. Values of individual variables for the case may then be obtained by calling the `spssGetValueNumeric` and `spssGetValueChar` procedures.

Parameter	Description
<i>handle</i>	Handle to the data file

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_FILE_END	End of the file reached; no more cases (warning)
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_WRMODE	File is open for writing, not reading
SPSS_FILE_ERROR	Error reading file

Example

See `spssGetValueChar`.

spssSeekNextCase

```
int spssSeekNextCase(const int handle, const long caseNumber)
```

Description

This function sets the file pointer of an input file so that the next data case read will be the one specified via the *caseNumber* parameter. A zero-origin scheme is used. That is, the first case is number 0. The next case can be read by calling either `spssWholeCaseIn` or `spssReadCaseRecord`. If the specified case is greater than or equal to the number of cases in the file, the call to the input function will return `SPSS_FILE_END`.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>caseNumber</i>	Zero-origin case number

Returns

Returns one of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_WRMODE	The file is open for writing, not reading
SPSS_NO_MEMORY	Insufficient memory
SPSS_FILE_ERROR	Error reading the file
SPSS_INVALID_FILE	The file is not a valid SPSS data file

See also **spssWholeCaseIn**, **spssReadCaseRecord**.

spssSetCaseWeightVar

int spssSetCaseWeightVar (int *handle*, const char **varName*)

Description

This function defines variable *varName* as the case weight variable for the data file specified by the *handle*.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varName</i>	The name of the case weight variable

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	File is open for reading, not writing
SPSS_DICT_COMMIT	Dictionary has already been written with <code>spssCommitHeader</code>
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist
SPSS_NUME_EXP	The variable is not numeric
SPSS_NO_MEMORY	Insufficient memory

Example

```
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;             /* error code */
    ...
    error = spssOpenWrite("data.sav", &fH);
    ...
    /* Define variables */
    error = spssSetVarName(fH, "NUMCHILD", SPSS_NUMERIC);
    if (error == SPSS_OK)
        error = spssSetVarName(fH, "TOYPREF", SPSS_NUMERIC);
    ...
    /* Set NUMCHILD as case weight */
    error = spssSetCaseWeightVar(fH, "NUMCHILD");
    if (error != SPSS_OK)
    {
        /* Handle error */
    }
}
```

spssSetCompression

int `spssSetCompression` (int *handle*, int *compSwitch*)

Description

This function sets the compression attribute of an SPSS data file. Compression is set on if *compSwitch* is one and off if it is zero. If this function is not called, the output file will be uncompressed by default.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>compSwitch</i>	Compression switch

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	File is open for reading, not writing
SPSS_DICT_COMMIT	Dictionary has already been written with <code>spssCommitHeader</code>
SPSS_INVALID_COMPSW	Invalid compression switch (other than 0 or 1)

Example

```
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;             /* error code */
    ...
    error = spssOpenWrite("data.sav", &fH);
    ...
    /* Set data compression on */
    error = spssSetCompression(fH, 1);
    ...
}
```

spssSetDateVariables

int spssSetDateVariables (int *handle*, int *numofElements*, const long **dateInfo*)

Description

This function sets the Trends date variable information. The array at *dateInfo* is assumed to have *numofElements* elements that correspond to the data array portion of record 7, subtype 3. Its first six elements comprise the “fixed” information, followed by a sequence of one or more three-element groups. Since very little validity checking is done on the input array, this function should be used with caution and is recommended only for copying Trends information from one file to another.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>numofElements</i>	Size of the array <i>dateInfo</i>
<i>dateInfo</i>	Array containing date variables information

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	File is open for reading, not writing
SPSS_DICT_COMMIT	Dictionary has already been written with <i>spssCommitHeader</i>
SPSS_INVALID_DATEINFO	The date variable information is invalid
SPSS_NO_MEMORY	Insufficient memory

Example

```

#include <stdlib.h>
#include "spssdio.h"
void func()
{
    int fHIn, fHOut; /* input & output file handles */
    int error;        /* error code */
    long *dateInfo;   /* pointer to date variable info. */
    int nElements;    /* number of elements in date info. array */
    ...
    /* Open one file for reading and one for writing. */
    error = spssOpenRead("bank.sav", &fHIn);
    ...
    error = spssOpenWrite("bankcopy.sav", &fHOut);
    ...
    /* Get the list of variables in input file;
    ** define variables in output file
    */
    ...
    /* Get date variable information from input file and copy
    ** it to output file
    */
    error = spssGetDateVariables(fHIn, &nElements, &dateInfo);
    if (error == SPSS_OK)
    {
        error = spssSetDateVariables(fHOut, nElements, dateInfo);
        ...
        free(dateInfo);
    }
    ...
}

```

See also **spssGetDateVariables**.

spssSetDEWFirst

int spssSetDEWFirst (const int *handle*, const void **pData*, const long *nBytes*)

Description

DEW information (file information which is private to the SPSS Data Entry product) can be delivered to the I/O DLL in whatever segments are convenient for the client. The spssSetDEWFirst function is called to deliver the first such segment, and subsequent segments are delivered by calling spssSetDEWNext as many times as necessary.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>pData</i>	Pointer to the data to be written
<i>nBytes</i>	Number of bytes to write

Returns

Returns one of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_EMPTY_DEW	Zero bytes to be written (warning)
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_READ_MODE	The file is not open for writing
SPSS_DICT_COMMIT	spssCommitHeader has already been called
SPSS_NO_MEMORY	Insufficient memory for control blocks
SPSS_FILE_BADTEMP	Cannot open or write to temporary file

See also **spssSetDEWNext**.

spssSetDEWGUID

int spssSetDEWGUID (const int handle, const char* asciiGUID)

Description

This function stores the Data Entry for Windows uniqueness indicator on the data file. It should only be used by the DEW product.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>asciiGUID</i>	The GUID (as a null-terminated string) to be stored on the file

Returns

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	The file is open for input or append

SPSS_DICT_COMMIT	spssCommitHeader has already been called
SPSS_NO_MEMORY	Insufficient memory to store the GUID

spssSetDEWNext

int spssSetDEWNext (const int *handle*, const void **pData*, const long *nBytes*)

Description

The DEW information (file information that is private to the SPSS Data Entry product) can be delivered to the I/O DLL in whatever segments are convenient for the client. The spssSetDEWFirst function is called to deliver the first such segment, and subsequent segments are delivered by calling spssSetDEWNext as many times as necessary.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>pData</i>	Pointer to the data to be written
<i>nBytes</i>	Number of bytes to write

Returns

Returns one of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_DEW_NOFIRST	spssSetDEWFirst was never called
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_READ_MODE	The file is not open for writing
SPSS_DICT_COMMIT	spssCommitHeader has already been called
SPSS_NO_MEMORY	Insufficient memory for control blocks
SPSS_FILE_BADTEMP	Cannot open or write to temporary file

See also **spssSetDEWFirst**.

spssSetFileAttributes

```
int spssSetFileAttributes(
    const int hFile,
    const char** attribNames,
    const char** attribText,
    const int nAttributes)
```

Description

This function replaces all the datafile attributes. It is the converse of **spssGetFileAttributes**, and the names of subscripted attributes must contain the unit origin subscripts in square brackets as in **Prerequisite[11]**. If the number of attributes is zero, the vector pointers can be **NULL**, and all attributes will be discarded.

Parameter	Description
<i>hFile</i>	Handle to the data file
<i>attribNames</i>	Pointer to a vector of attribute names
<i>attribText</i>	Pointer to a vector of attribute values
<i>nAttributes</i>	The number of element in each vector

Returns

Returns one of the following codes. Success is indicated by zero (**SPSS_OK**), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDONLY	The file is read-only
SPSS_DICT_COMMIT	spssCommitHeader has already been called
SPSS_INVALID_ATTRDEF	Missing name, missing text, or invalid subscript
SPSS_INVALID_ATTRNAME	Lexically invalid attribute name

spssSetIdString

int spssSetIdString (int *handle*, const char **id*)

Description

This function sets the file label of the output SPSS data file associated with *handle* to the given string *id*.

Parameter	Description
<i>handle</i>	Handle to the data file.
<i>id</i>	File label. The length of the string should not exceed 64 characters.

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_EXC_LEN64	Label length exceeds 64; truncated and used (warning)
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	File is open for reading, not writing
SPSS_DICT_COMMIT	Dictionary has already been written with spssCommitHeader

Example

```

include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;             /* error code */
    char id[] = "This is a file label.";
    ...
    error = spssOpenWrite("data.sav", &fH);
    ...
    error = spssSetIdString(fH, id);
    if (error == SPSS_OK)
    {
        /* The label of the data file is now the string
        ** "This is a file label."
        */
        ...
    }
}

```

spssSetLocale

```

char* spssSetLocale(
    const int iCategory,
    const char* pszLocale)

```

Description

The I/O DLL's locale is separate from that of the client application. When the DLL is first loaded, its locale is set to the system default. The `spssSetLocale` function gives the client application control over the DLL's locale. The parameters and return value are identical to those for the C run-time function `setlocale`.

Parameter	Description
<i>iCategory</i>	A locale category, for example <code>LC_ALL</code> or <code>LC_CTYPE</code> . These are defined in the header file <i>locale.h</i> .
<i>pszLocale</i>	A locale, for example "Japanese.932".

Returns

The function returns the resulting locale, for example "French_Canada.1252"

spssSetMultRespDefs

```
int spssSetMultRespDefs(const int handle, const char *mrespDefs)
```

Description

This function is used to write multiple response definitions to the file. The definitions consist of a single null-terminated ASCII string which is similar to that containing the variable set definitions.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>mrespDefs</i>	ASCII string containing definitions

Returns

Returns one of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_EMPTY_MULTRESP	The string contains no definitions (warning)
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	The file is open for input or append
SPSS_DICT_COMMIT	spssCommitHeader has already been called
SPSS_NO_MEMORY	Insufficient memory to store the definitions

spssSetTempDir

```
int spssSetTempDir (const char* dirName)
```

Description

The I/O DLL spills some large object to temporary files. Normally these files reside in the directory supplied by the Windows GetTempPath function. The `spssSetTempDir` function permits the I/O DLL client to specify a different directory.

Parameter	Description
<i>dirName</i>	Fully-qualified directory name as a null-terminated string

Returns

Error Code	Description
SPSS_OK	No error
SPSS_NO_MEMORY	Insufficient memory to store the path

spssSetTextInfo

int `spssSetTextInfo` (int *handle*, const char **textInfo*)

Description

This function sets the text data from the null-terminated string in `textInfo`. If the string is longer than 255 characters, only the first 255 are (quietly) used. If `textInfo` contains the empty string, existing text data, if any, are deleted.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>textInfo</i>	Text data

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid

SPSS_OPEN_RDMODE	The file is open for input or append
SPSS_DICT_COMMIT	spssCommitHeader has already been called
SPSS_NO_MEMORY	Insufficient memory

spssSetValueChar

int spssSetValueChar (int *handle*, double *varHandle*, const char **value*)

Description

This function sets the value of a string variable for the current case. The current case is not written out to the data file until spssCommitCaseRecord is called.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varHandle</i>	Handle to the variable
<i>value</i>	Value of the variable as a null-terminated string. The length of the string (ignoring trailing blanks, if any) should be less than or equal to the length of the variable.

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	File is open for reading, not writing
SPSS_DICT_NOTCOMMIT	Dictionary of the output file has not yet been written with spssCommitHeader
SPSS_STR_EXP	Variable associated with the handle is numeric
SPSS_EXC_STRVALUE	The value is longer than the length of the variable

Example

See **spssSetValueNumeric**.

See also **spssCommitCaseRecord**.

spssSetValueNumeric

```
int spssSetValueNumeric (int handle, double varHandle, double value)
```

Description

This function sets the value of a numeric variable for the current case. The current case is not written out to the data file until **spssCommitCaseRecord** is called.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varHandle</i>	Handle to the variable
<i>value</i>	Value of the variable

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	File is open for reading, not writing
SPSS_DICT_NOTCOMMIT	Dictionary of the output file has not yet been written with spssCommitHeader
SPSS_NUME_EXP	Variable associated with the handle is not numeric

Example

```

#include "spssdio.h"
void func()
{
    int    fH;                /* file handle */
    int    error;             /* error code */
    double ageH, titleH;      /* variable handles */
    double age;               /* value of AGE */
    ...
    error = spssOpenWrite("data.sav", &fH);
    ...
    /* Create numeric variable AGE and long string variable
    ** TITLE
    */
    error = spssSetVarName(fH, "AGE", SPSS_NUMERIC);
    ...
    error = spssSetVarName(fH, TITLE, SPSS_STRING(20));
    ...
    /* Done with dictionary definition; commit dictionary */
    error = spssCommitHeader(fH);
    ...
    /* Get variable handles */
    error = spssGetVarHandle(fH, "AGE", &ageH);
    ...
    error = spssGetVarHandle(fH, "TITLE", &titleH);
    ...
    /* Construct & write cases, with AGE set to 20, 21, ... 46
    ** and TITLE set to "Super salesman"
    */
    for (age = 20.0; age <= 46.0; ++age)
    {
        error = spssSetValueNumeric(fH, ageH, age);
        ...
        error = spssSetValueChar(fH, titleH, "Super salesman");
        ...
        error = spssCommitCaseRecord(fH);
        ...
    }
    error = spssCloseWrite(fH);
    ...
}

```

See also `spssConvertDate`, `spssConvertTime`, `spssCommitCaseRecord`.

spssSetVarAlignment

`int spssSetVarAlignment (int handle, const char *varName, int alignment)`

Description

This function sets the value of the alignment attribute of a variable.

Parameter	Description
<i>handle</i>	Handle to the data file.
<i>varName</i>	Variable name.
<i>alignment</i>	Alignment. Must be one of SPSS_ALIGN_LEFT, SPSS_ALIGN_RIGHT, or SPSS_ALIGN_CENTER. If not a legal value, alignment is set to a type-appropriate default.

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	The file is open for input or append
SPSS_DICT_COMMIT	spssCommitHeader has already been called
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist

spssSetVarAttributes

```
int spssSetVarAttributes(
    const int hFile,
    const char* varName,
    const char** attribNames,
    const char** attribText,
    const int nAttributes)
```

Description

This function is analogous to spssSetFileAttributes. It replaces all the attributes for one variable.

Parameter	Description
<i>hFile</i>	Handle to the data file

<i>varName</i>	Name of the variable
<i>attribNames</i>	Pointer to a vector of attribute names
<i>attribText</i>	Pointer to a vector of attribute values
<i>nAttributes</i>	The number of element in each vector

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_VAR_NOTFOUND	Named variable is not in the file
SPSS_OPEN_RDONLY	The file is read-only
SPSS_DICT_COMMIT	spssCommitHeader has already been called
SPSS_INVALID_ATTRDEF	Missing name, missing text, or invalid subscript
SPSS_INVALID_ATTRNAME	Lexically invalid attribute name

spssSetVarCMissingValues

```
int spssSetVarCMissingValues
(int handle, const char *varName, int missingFormat,
const char *missingVal1, const char *missingVal2, const char *missingVal3)
```

Description

This function sets missing values for a short string variable. The argument *missingFormat* must be set to a value in the range 0–3 to indicate the number of missing values supplied. When fewer than three missing values are to be defined, the redundant arguments must still be present, although their values are not inspected. For example, if *missingFormat* is 2, *missingVal3* is unused. The supplied missing values must be null-terminated and not longer than the length of the variable unless the excess length is

made up of blanks, which are ignored. If the missing value is shorter than the length of the variable, trailing blanks are assumed.

Parameter	Description
<i>handle</i>	The handle to the data file
<i>varName</i>	Variable name
<i>missingFormat</i>	Missing format code
<i>missingVal1</i>	First missing value
<i>missingVal2</i>	Second missing value
<i>missingVal3</i>	Third missing value

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	File is open for reading, not writing
SPSS_DICT_COMMIT	Dictionary has already been written with <code>spssCommitHeader</code>
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist
SPSS_STR_EXP	The variable is numeric
SPSS_SHORTSTR_EXP	The variable is a long string (length > 8)
SPSS_INVALID_MISSFOR	Invalid missing values specification (<i>missingFormat</i> is not in the range 0–3)
SPSS_EXC_STRVALUE	A missing value is longer than the length of the variable
SPSS_NO_MEMORY	Insufficient memory

Example

```

#include <stddef.h>
#include "spssdio.h"
void func()
{
    int fH;                                /* file handle */
    int error;                             /* error code */
    ...
    error = spssOpenWrite("data.sav", &fH);
    ...
    /* Create short string variable TITLE and define values
    ** consisting of blanks or periods only as missing
    */
    error = spssSetVarName(fH, "TITLE", SPSS_STRING(6));
    if (error == SPSS_OK)
    {
        /* Last arg. is a placeholder since we are defining only two
        ** missing values
        */
        error = spssSetVarCMissingValues(fH, "TITLE", 2,
            ".....", " ", NULL);
        ...
    }
}

```

spssSetVarColumnWidth

int spssSetVarColumnWidth (int *handle*, const char **varName*, int *columnWidth*)

Description

This function sets the value of the column width attribute of a variable. A value of zero is special and means that the SPSS Data Editor, which is the primary user of this attribute, is to set an appropriate width using its own algorithm.

Parameter	Description
<i>handle</i>	Handle to the data file.
<i>varName</i>	Variable name.
<i>columnWidth</i>	Column width. If negative, a value of zero is (quietly) used instead.

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	The file is open for input or append
SPSS_DICT_COMMIT	spssCommitHeader has already been called
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist

spssSetVarCValueLabel

```
int spssSetVarCValueLabel
(int handle, const char *varName, const char *value, const char *label)
```

Description

This function changes or adds a value label for the specified value of a short string variable. The label should be a null-terminated string not exceeding 60 characters in length.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varName</i>	Variable name
<i>value</i>	Value to be labeled
<i>label</i>	Label

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_INVALID_HANDLE	The file handle is not valid.
SPSS_OPEN_RDMODE	File is open for reading, not writing.
SPSS_DICT_COMMIT	Dictionary has already been written with spssCommitHeader.
SPSS_INVALID_VARNAME	Variable name is invalid.
SPSS_VAR_NOTFOUND	A variable with the given name does not exist.
SPSS_STR_EXP	The variable is numeric.
SPSS_SHORTSTR_EXP	The variable is a long string (length > 8).
SPSS_EXC_STRVALUE	The value (<i>*value</i>) is longer than the length of the variable.
SPSS_NO_MEMORY	Insufficient memory.
SPSS_INTERNAL_VLABS	Internal data structures of the DLL are invalid. This signals an error in the DLL.

Example

```
#include "spssdio.h"
void func()
{
    int    fH;                /* file handle    */
    int    error;             /* error code     */
    ...
    error = spssOpenWrite("data.sav", &fH);
    ...
    /* Create short string variable TITLE and label the value
    ** consisting of all blanks as "Did not want title"
    */
    error = spssSetVarName(fH, "TITLE", SPSS_STRING(6));
    if (error == SPSS_OK)
    {
        error = spssSetVarCValueLabel(fH, "TITLE", "      ",
        "Did not want title");
    }
}
```

See also **spssSetVarCValueLabels**.

spssSetVarCValueLabels

```
int spssSetVarCValueLabels
(int handle, const char **varNames, int numVars,
const char **values, const char **labels, int numLabels)
```

Description

This function defines a set of value labels for one or more short string variables. Value labels already defined for any of the given variable(s), if any, are discarded (if the labels are shared with other variables, they remain associated).

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varNames</i>	Array of pointers to variable names
<i>numVars</i>	Number of variables
<i>values</i>	Array of pointers to values
<i>labels</i>	Array of pointers to labels
<i>numLabels</i>	Number of labels or values)

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error.
SPSS_INVALID_HANDLE	The file handle is not valid.
SPSS_OPEN_RDONLY	File is open for reading, not writing.
SPSS_DICT_COMMIT	Dictionary has already been written with <code>spssCommitHeader</code> .
SPSS_NO_VARIABLES	Number of variables (<i>numVars</i>) is zero or negative.
SPSS_NO_LABELS	Number of labels (<i>numLabels</i>) is zero or negative.
SPSS_INVALID_VARNAME	At least one variable name is invalid.

SPSS_VAR_NOTFOUND	At least one of the variables does not exist.
SPSS_STR_EXP	At least one of the variables is numeric.
SPSS_SHORTSTR_EXP	At least one of the variables is a long string (length < 8).
SPSS_EXC_STRVALUE	At least one value is longer than the length of the variable.
SPSS_DUP_VALUE	The list of values contains duplicates.
SPSS_NO_MEMORY	Insufficient memory.
SPSS_INTERNAL_VLABS	Internal data structures of the DLL are invalid. This signals an error in the DLL.

Example

```
#include "spssdio.h"
void func()
{
    int    fH;                                /* file handle          */
    int    error;                             /* error code           */
    static char *vNames[2]=                   /* variable names       */
    { "TITLE", "OLDTITLE" };
    static char *vValues[3] =                 /* values to be labeled */
    { " ", "techst", "consul" };
    static char *vLabels[3] =                /* corresponding labels */
    { "Unknown", "Member of tech. staff", "Outside consultant" };
    ...
    error = spssOpenWrite("data.sav", &fH);
    ...
    /* Define two short string variables TITLE & OLDTITLE and a
    ** set of shared value labels
    */
    error = spssSetVarName(fH, vNames[0], SPSS_STRING(6));
    if (error == SPSS_OK)
        error = spssSetVarName(fH, vNames[1], SPSS_STRING(6));
    if (error == SPSS_OK)
    {
        error =
            spssSetVarCValueLabels(fH, vNames, 2, vValues, vLabels, 3);
        ...
    }
}
```

See also **spssSetVarCValueLabel**.

spssSetVarLabel

int spssSetVarLabel (int *handle*, const char **varName*, const char **varLabel*)

Description

This function sets the label of a variable.

Parameter	Description
<i>handle</i>	Handle to the data file.
<i>varName</i>	Variable name.
<i>varLabel</i>	Variable label. The length of the string should not exceed 120 characters. If <i>varLabel</i> is the empty string, the existing label, if any, is deleted.

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_EXC_LEN120	Variable label's length exceeds 120; truncated and used (warning)
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	File is open for reading, not writing
SPSS_DICT_COMMIT	Dictionary has already been written with <code>spssCommitHeader</code>
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist
SPSS_NO_MEMORY	Insufficient memory

Example

```
#include "spssdio.h"
void func()
{
    int  fH;                /* file handle */
    int  error;             /* error code */
    ...
    error = spssOpenWrite("data.sav", &fH);
    /* Do the file operations here */
    ...
    /* Define string variable NAME of length 8 */
    error = spssSetVarName(fH, "NAME", SPSS_STRING(8));
    ...
    /* Label the variable */
    error =
        spssSetVarLabel(fH, "NAME", "Name of respondent");
    ...
}
```

spssSetVarMeasureLevel

int spssSetVarMeasureLevel (int *handle*, const char **varName*, int *measureLevel*)

Description

This function sets the value of the measurement level attribute of a variable.

Parameter	Description
<i>handle</i>	Handle to the data file.
<i>varName</i>	Variable name.
<i>measureLevel</i>	Measurement level. Must be one of SPSS_MLVL_NOM, SPSS_MLVL_ORD, SPSS_MLVL_RAT, or SPSS_MLVL_UNK for nominal, ordinal, scale (ratio), and unknown, respectively. If SPSS_MLVL_UNK, measurement level is set to a type-appropriate default.

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error

SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	The file is open for input or append
SPSS_DICT_COMMIT	spssCommitHeader has already been called
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist
SPSS_INVALID_MEASURELEVEL	measureLevel is not in the legal range, or it is SPSS_MLVL_RAT and the variable is a string variable

spssSetVarNMissingValues

```
int spssSetVarNMissingValues
(int handle, const char *varName, int missingFormat,
double missingVal1, double missingVal2, double missingVal3)
```

Description

This function sets missing values for a numeric variable. The interpretation of the arguments *missingVal1*, *missingVal2*, and *missingVal3* depends on the value of *missingFormat*. If *missingFormat* is set to SPSS_MISS_RANGE, *missingVal1* and *missingVal2* are taken as the upper and lower limits, respectively, of the range, and *missingVal3* is ignored. If *missingFormat* is SPSS_MISS_RANGEANDVAL, *missingVal1* and *missingVal2* are taken as limits of the range and *missingVal3* is taken as the discrete missing value. If *missingFormat* is neither of the above, it must be in the range 0–3, indicating the number of discrete missing values present. For example, if *missingFormat* is 2, *missingVal1* and *missingVal2* are taken as two discrete missing values and *missingVal3* is ignored. (The macros SPSS_NO_MISSVAL, SPSS_ONE_MISSVAL, SPSS_TWO_MISSVAL, and SPSS_THREE_MISSVAL may be used as synonyms for 0–3.)

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varName</i>	Variable name
<i>missingFormat</i>	Missing values format code
<i>missingVal1</i>	First missing value

<i>missingVal2</i>	Second missing value
<i>missingVal3</i>	Third missing value

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	File is open for reading, not writing
SPSS_DICT_COMMIT	Dictionary has already been written with <code>spssCommitHeader</code>
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist
SPSS_NUME_EXP	The variable is not numeric
SPSS_INVALID_MISSFOR	Invalid missing values specification (<i>missingFormat</i> is invalid or the lower limit of range is greater than the upper limit)
SPSS_NO_MEMORY	Insufficient memory

Example

```

#include "spssdio.h"
void func()
{
    int    fH;                      /* file handle    */
    int    error;                   /* error code     */
    ...
    error = spssOpenWrite("data.sav", &fH);
    ...
    /* Create numeric variable BUYCODE and set range 1-9 as
    ** missing
    */
    error = spssSetVarName(fH, "BUYCODE", SPSS_NUMERIC);
    if (error == SPSS_OK)
    {
        /* Last arg. is a placeholder since we are defining a range
        ** only
        */
        error =
            spssSetVarNMissingValues(fH, "BUYCODE", SPSS_MISS_RANGE,
                                     1.0, 9.0, 0.0);
        ...
    }
}

```

See also **spssSetVarCMissingValues**.

spssSetVarNValueLabel

```
int spssSetVarNValueLabel
```

```
(int handle, const char *varName, double value, const char *label)
```

Description

This function changes or adds a value label for the specified value of a numeric variable. The label should be a null-terminated string not exceeding 60 characters in length.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varName</i>	Variable name
<i>value</i>	Value to be labeled
<i>label</i>	Label

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error.
SPSS_INVALID_HANDLE	File handle not valid.
SPSS_OPEN_RDMODE	File is open for reading, not writing.
SPSS_DICT_COMMIT	Dictionary has already been written with <code>spssCommitHeader</code> .
SPSS_INVALID_VARNAME	Variable name is invalid.
SPSS_VAR_NOTFOUND	A variable with the given name does not exist.
SPSS_NUME_EXP	The variable is not numeric.
SPSS_NO_MEMORY	Insufficient memory.
SPSS_INTERNAL_VLABS	Internal data structures of the DLL are invalid. This signals an error in the DLL.

Example

```
#include "spssdio.h"
void func()
{
    int    fH;                      /* file handle    */
    int    error;                   /* error code     */
    ...
    error = spssOpenWrite("data.sav", &fH);
    ...
    /* Create numeric variable BUYCODE and label value 0.0 as
    ** "Unknown"
    */
    error = spssSetVarName(fH, "BUYCODE", SPSS_NUMERIC);
    if (error == SPSS_OK)
    {
        error =
            spssSetVarNValueLabel(fH, "BUYCODE", 0.0, "Unknown");
        ...
    }
}
```

See also `spssSetVarNValueLabels`.

spssSetVarNValueLabels

```
int spssSetVarNValueLabels
(int handle, const char **varNames, int numVars,
const double *values, const char **labels, int numLabels)
```

Description

This function defines a set of value labels for one or more numeric variables. Value labels already defined for any of the given variable(s), if any, are discarded (if the labels are shared with other variables, they remain associated with those variables).

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varNames</i>	Array of pointers to variable names
<i>numVars</i>	Number of variables
<i>values</i>	Array of values
<i>labels</i>	Array of pointers to labels
<i>numLabels</i>	Number of labels or values

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error.
SPSS_INVALID_HANDLE	The file handle is not valid.
SPSS_OPEN_RDONLY	File is open for reading, not writing.
SPSS_DICT_COMMIT	Dictionary has already been written with <code>spssCommitHeader</code> .
SPSS_NO_VARIABLES	Number of variables (<i>numVars</i>) is zero or negative.
SPSS_NO_LABELS	Number of labels (<i>numLabels</i>) is zero or negative.
SPSS_INVALID_VARNAME	At least one variable name is invalid.

SPSS_VAR_NOTFOUND	At least one of the variables does not exist.
SPSS_NUME_EXP	At least one of the variables is not numeric.
SPSS_DUP_VALUE	The list of values contains duplicates.
SPSS_NO_MEMORY	Insufficient memory.
SPSS_INTERNAL_VLABS	Internal data structures of the DLL are invalid. This signals an error in the DLL.

Example

```
#include "spssdio.h"
void func()
{
    int    fH;                /* file handle          */
    int    error;             /* error code           */
    static char *vNames[2]=   /* variable names       */
    { "AGE", "AGECHILD" };
    static double vValues[3] = /* values to be labeled */
    { -2.0, -1.0, 0.0 };
    static char *vLabels[3] = /* corresponding labels */
    { "Unknown", "Not applicable", "Under 1" };
    ...
    error = spssOpenWrite("data.sav", &fH);
    ...
    /* Define two numeric variables AGE & AGECHILD and a set of
    ** shared value labels
    */
    error = spssSetVarName(fH, vNames[0], SPSS_NUMERIC);
    if (error == SPSS_OK)
        error = spssSetVarName(fH, vNames[1], SPSS_NUMERIC);
    if (error == SPSS_OK)
    {
        error =
            spssSetVarNValueLabels(fH, vNames, 2, vValues, vLabels, 3);
        ...
    }
}
```

See also **spssSetVarNValueLabel**.

spssSetVarName

int spssSetVarName (int *handle*, const char **varName*, int *varLength*)

Description

This function creates a new variable named *varName*, which will be either numeric or string based on *varLength*. If the latter is zero, a numeric variable with a default format of F8.2 will be created; if it is greater than 0 and less than or equal to 32767, a string variable with length *varLength* will be created; any other value will be rejected as invalid. For better readability, the macros SPSS_NUMERIC and SPSS_STRING(*length*) may be used as values for *varLength*.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varName</i>	Variable name
<i>varLength</i>	Type and size of the variable

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	File is open for reading, not writing
SPSS_DICT_COMMIT	Dictionary has already been written with <code>spssCommitHeader</code>
SPSS_INVALID_VARTYPE	Invalid length code (<i>varLength</i> is negative or exceeds 32767)
SPSS_INVALID_VARNAME	Variable name is invalid
SPSS_DUP_VAR	There is already a variable with the same name
SPSS_NO_MEMORY	Insufficient memory

Example

```
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;             /* error code */
    ...
    error = spssOpenWrite("data.sav", &fH);
    ...
    /* Create numeric variable AGE and string variable NAME */
    error = spssSetVarName(fH, "AGE", SPSS_NUMERIC);
    if (error == SPSS_OK)
        error = spssSetVarName(fH, "NAME", SPSS_STRING(20));
    ...
}
```

spssSetVarPrintFormat

int spssSetVarPrintFormat
 (int *handle*, const char **varName*, int *printType*, int *printDec*, int *printWid*)

Description

This function sets the print format of a variable.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varName</i>	Variable name
<i>printType</i>	Print format type code (file <i>spssdio.h</i> defines macros of the form SPSS_FMT_... for all valid format type codes)
<i>printDec</i>	Number of digits after the decimal
<i>printWid</i>	Print format width

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error

SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	File is open for reading, not writing
SPSS_DICT_COMMIT	Dictionary has already been written with <code>spssCommitHeader</code>
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist
SPSS_INVALID_PRFOR	The print format specification is invalid or is incompatible with the variable type
SPSS_NO_MEMORY	Insufficient memory

Example

```
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;             /* error code */
    ...
    error = spssOpenWrite("data.sav", &fH);
    /* Define numeric variable TIMESTMP */
    error = spssSetVarName(fH, "TIMESTMP", SPSS_NUMERIC);
    ...
    /* Set the print format of TIMESTMP to DATETIME28.4 */
    error = spssSetVarPrintFormat(fH, "TIMESTMP",
        SPSS_FMT_DATE_TIME, 4, 28);
    ...
}
```

See also **`spssSetVarWriteFormat`**.

spssSetVarWriteFormat

```
int spssSetVarWriteFormat
(int handle, const char *varName, int writeType, int writeDec, int writeWid)
```

Description

This function sets the write format of a variable.

Parameter	Description
-----------	-------------

<i>handle</i>	Handle to the data file
<i>varName</i>	Variable name
<i>writeType</i>	Write format type code (file <i>spssdio.h</i> defines macros of the form SPSS_FMT_... for all valid format type codes)
<i>writeDec</i>	Number of digits after the decimal
<i>writeWid</i>	Write format width

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	File is open for reading, not writing
SPSS_DICT_COMMIT	Dictionary has already been written with spssCommitHeader
SPSS_INVALID_VARNAME	The variable name is not valid
SPSS_VAR_NOTFOUND	A variable with the given name does not exist
SPSS_INVALID_WRFOR	The write format specification is invalid or is incompatible with the variable type
SPSS_NO_MEMORY	Insufficient memory

Example

```
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;             /* error code */
    ...
    error = spssOpenWrite("data.sav", &fH);
    /* Define string variable ODDCHARS of length 7 */
    error = spssSetVarName(fH, "ODDCHARS", SPSS_STRING(7));
    ...
    /* Set the write format of ODDCHARS to AHX14 */
    error =
    spssSetVarWriteFormat(fH, "ODDCHARS", SPSS_FMT_AHEX, 0, 14);
    ...
}
```

spssSetVariableSets

```
int spssSetVariableSets (int handle, const char *varSets)
```

Description

This function sets the variable sets information in the data file. The information must be provided in the form of a null-terminated string. No validity checks are performed on the supplied string beyond ensuring that its length is not 0. Any existing variable sets information is discarded.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>varSets</i>	Variable sets information

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_EMPTY_VARSETS	The variable sets information is empty (warning)
SPSS_INVALID_HANDLE	The file handle is not valid

SPSS_OPEN_RDMODE	File is open for reading, not writing
SPSS_DICT_COMMIT	Dictionary has already been written with spssCommitHeader
SPSS_NO_MEMORY	Insufficient memory

Example

```
#include <stdlib.h>
#include "spssdio.h"
void func()
{
    int fHIn, fHOut;          /* input & output file handles */
    int error;                /* error code */
    char *vSets;              /* ptr to variable sets info. */
    ...
    /* Open one file for reading and one for writing. */
    error = spssOpenRead("bank.sav", &fHIn);
    ...
    error = spssOpenWrite("bankcopy.sav", &fHOut);
    ...
    /* Copy variable sets information from input file to output
    ** file
    */
    error = spssGetVariableSets(fHIn, &vSets);
    if (error == SPSS_OK)
    {
        error = spssSetVariableSets(fHOut, vSets);
        /* Handle errors and remember to free variable set string */
        ...
        free(vSets);
    }
    else if (error != SPSS_EMPTY_VARSETS)
    {
        /* Error getting variable sets information from input file */
        ...
    }
    ...
}
```

spssSysmisVal

double spssSysmisVal (void)

Description

This function returns the SPSS system-missing value for the host system. It may be called at any time.

Parameter	Description
-----------	-------------

None

Returns

The SPSS system-missing value for the host system.

Example

```
#include <stdio.h>
#include "spssdio.h"
void func()
{
    double sysmis;          /* system missing value */
    ...
    /* Get and print the system missing value */
    sysmis = spssSysmisVal();
    printf("System missing value: %e\n");
    ...
}
```

spssValidateVarname

int spssValidateVarname (const char* varName)

Description

This function allows the client to validate a potential variable name. The name is checked for lexical validity only; there is no check for whether it is a duplicate name. Note that the error code SPSS_NAME_BADFIRST indicates that the name is entirely composed of valid characters but that the first character is not valid in that position, for example the name begins with a period or digit. Note also that names ending with a period are technically valid but are to be discouraged because they cause difficulty if they appear at the end of a line of syntax.

Parameter	Description
<i>varName</i>	null-terminated variable name

Returns

Error Code	Description
SPSS_NAME_OK	The name is valid

SPSS_NAME_SCRATCH	The name is invalid because it begins with "#"
SPSS_NAME_SYSTEM	The name is invalid because it begins with "\$"
SPSS_NAME_BADLTH	The name is too long
SPSS_NAME_BADCHAR	The name contains an invalid character
SPSS_NAME_RESERVED	The name is an SPSS reserved word
SPSS_NAME_BADFIRST	The name begins with an invalid character

spssWholeCaseIn

int spssWholeCaseIn (int *handle*, char **caseRec*)

Description

This function reads a case from a data file into a case buffer provided by the user. The required size of the buffer may be obtained by calling spssGetCaseSize. This is a fairly low-level function whose use should not be mixed with calls to spssReadCaseRecord using the same file handle because both procedures read a new case from the data file.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>caseRec</i>	Buffer to contain the case

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_FILE_END	End of the file reached; no more cases (warning)
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_WRMODE	File is open for writing, not reading
SPSS_FILE_ERROR	Error reading file

Example

```

#include <stdlib.h>
#include "spssdio.h"
void func()
{
    int fH;                /* file handle */
    int error;             /* error code */
    int caseSize;          /* size of a case */
    char *cRec;            /* pointer to case record */
    ...
    error = spssOpenRead("bank.sav", &fH);
    ...
    /* Find out the size of the case and allocate memory for the
    ** case record.
    */
    error = spssGetCaseSize(fH, &caseSize);
    ...
    cRec = (char *) malloc(caseSize);
    ...
    error = spssWholeCaseIn(fH, cRec);
    ...
    /* Buffer cRec now contains the first case in the data file.
    ** It is up to us to make sense out of it.
    */
    ...
}

```

See also **spssGetCaseSize**, **spssWholeCaseOut**.

spssWholeCaseOut

```

int spssWholeCaseOut
(int handle, const char *caseRec)

```

Description

This function writes a case assembled by the caller to a data file. The case is assumed to have been constructed correctly in the buffer *caseRec*, and its validity is not checked. This is a fairly low-level function whose use should not be mixed with calls to **spssCommitCaseRecord** using the same file handle because both procedures write a new case to the data file.

Parameter	Description
<i>handle</i>	Handle to the data file
<i>caseRec</i>	Case record to be written to the data file

Returns

One of the following codes. Success is indicated by zero (SPSS_OK), errors by positive values, and warnings, if any, by negative values.

Error Code	Description
SPSS_OK	No error
SPSS_INVALID_HANDLE	The file handle is not valid
SPSS_OPEN_RDMODE	File is open for reading, not writing
SPSS_DICT_NOTCOMMIT	Dictionary of the output file has not yet been written with <code>spssCommitHeader</code>
SPSS_FILE_WERROR	File write error

Example

```

#include <string.h>
#include "spssdio.h"
void func()
{
    int fh;                /* file handle */
    int error;              /* error code */
    int caseSize;           /* size of a case */
    char caseRec[16];       /* case record */
    double age;             /* value of AGE */
    ...
    error = spssOpenWrite("data.sav", &fh);
    ...
    /* Define two variables */
    error = spssSetVarName(fh, "NAME", SPSS_STRING(7));
    ...
    error = spssSetVarName(fh, "AGE", SPSS_NUMERIC);
    ...
    /* Done with dictionary definition; commit dictionary */
    error = spssCommitHeader(fh);
    ...
    /* Please note that code beyond this requires knowledge of
    ** SPSS data file formats, and it very easy to produce
    ** garbage.
    */
    /* Find out the size of the case and make sure it is 16 as
    ** we assume it to be
    */
    error = spssGetCaseSize(fh, &caseSize);
    ...
    /* Construct one case with NAME "KNIEVEL" and AGE 50.
    ** Write out the case and close file.
    */
    memcpy(caseRec, "KNIEVEL ", 8); /* Padding to 8 */
    age = 50.0;
    memcpy(caseRec+8, &age, 8); /* Assuming sizeof double is 8 */
    error = spssWholeCaseOut(fh, caseRec);
    ...
    error = spssCloseWrite(fh);
    ...
}

```

See also **spssGetCaseSize**, **spssWholeCaseIn**.

SPSS Third-Party API for SPSS for Windows

Introduction

This document is intended for developers who are writing applications to be integrated with SPSS for Windows. It discusses the specific registry entries you'll need to use and strategies for modifying your setup program to automatically register add-ins to SPSS.

The SPSS third-party API permits applications, scripts, and syntax files (**add-ins**) to be added to the menu bar of the SPSS product. These add-ins are stored in the system registry and are persistent from session to session. Beginning with SPSS 7.5, users can add menu and toolbar items interactively. These entries are stored in the registry using the same technique. You'll need to familiarize yourself with the Windows registry in order to use the information below effectively. Good background articles are available with Microsoft Visual C++ (32-bit editions) and the Microsoft Developer Support Network.

Add-ins are registered on the following path:

HKEY_CURRENT_USER\Software\SPSS\SPSS for Windows[version]\OtherApps

where *[version]* is the version number of the installed version of SPSS (e.g., 12.0).

For localized versions in languages other than English, the language name is appended to *SPSS for Windows*. Depending on the release, localized versions may include:

- Japanese

- French
- Russian
- Korean
- Polish
- Simplified Chinese
- Traditional Chinese
- Spanish
- Italian
- German

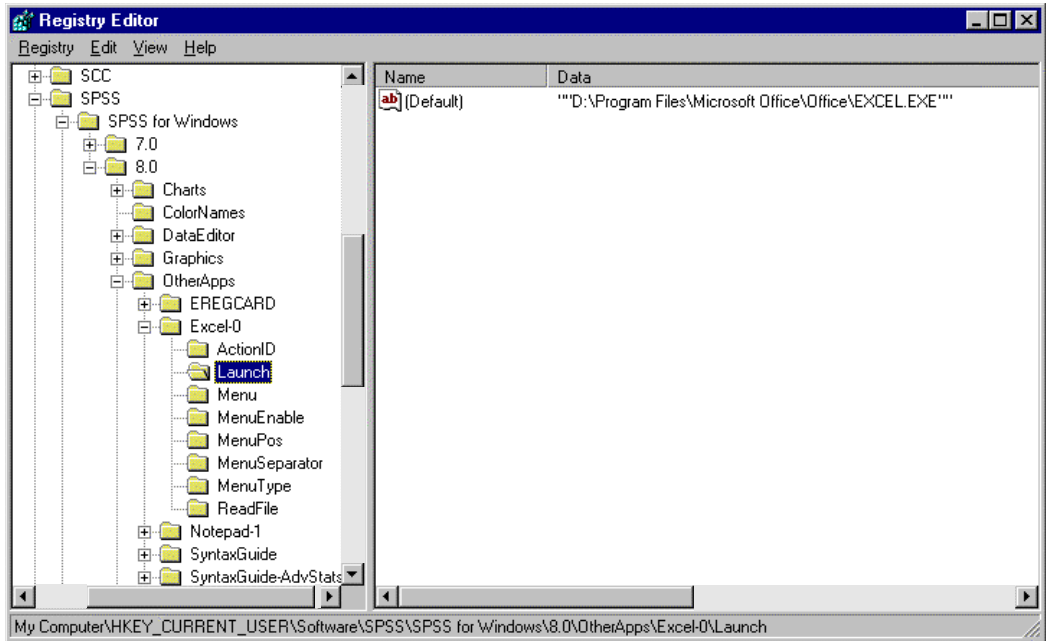
For example, the registry path for the add-ins for the German version of SPSS 12.0 is:

HKEY_CURRENT_USER\Software\SPSS\SPSS for Windows German \12.0\OtherApps

Contents of the Registry

When SPSS is launched, it will look for the presence of the SPSS *OtherApps* key. If found, SPSS will iterate over its subkeys, each of which represents a menu add-in. In Figure B-1 below, there are two add-in keys, Excel charting-0 and Notepad0. The names of these keys are not in themselves significant as long as they're unique within OtherApps.

Figure B-1
Windows Registry Editor with Notepad and Excel added to SPSS menus



The specific details of how the add-ins are configured are located under the add-in keys and are keys, not values. Table B-1 summarizes the available keys. All keys use only the default name, and their values are stored in the registry as strings. Notice the right pane of Figure B-1. The ab icon indicates a string value, and (Default) indicates the default (null) name. Make sure that when you've populated the registry, the values all have this representation in the registry.


Table B-1
Add-in configuration details

Key	Description
ActionID	How the application is to be launched
DeleteFile	Whether the data file is to be deleted when SPSS terminates
Launch	The command line to launch the program
Menu	Name of menu and its place in the menu hierarchy
MenuEnable	Whether always enabled or only if data are loaded into SPSS
MenuPos	The position of the menu item on the containing pop-up menu
MenuSeparator	Whether a menu separator is placed above menu item
MenuType	Which SPSS windows are affected
Minimize	Whether or not the application is launched minimized and whether or not SPSS is minimized when the application is launched
ReadFile	The type of data file passed to the application, if any
SingleSeat	Whether the application is disabled in client/server mode.
TBOnly	Whether the application is to be accessible only from the toolbar

A description of each configuration detail follows.

Launch

The application name and verb within the registration database. The name and verb are validated when SPSS attempts to launch the third-party application. The maximum length of the value is 255 characters. SPSS will not update the menus if it is longer. For example:

 (Default)

"C:\MSOFFICE\EXCEL\Excel.exe"

MenuType

Indicates which SPSS window(s) the menu item will be added to. The specification is a list of values separated by commas. Valid entries are 0, 2, 3, or 4. Zero is the Data Editor, 2 is the Viewer, 3 is the Syntax Editor, and 4 is the Script menu. The example below would place the menu item in the Data Editor and syntax windows.

For example:



"0,3"

MenuPos

Indicates the position of the menu item within the individual menu popups. This can be a series of numbers if the user has created new menus as well as menu items. In the example below, a new menu, My Applications, was added to the Utilities menu, and three submenus, Corel tools, Productivity tools, and Games, were added under it. The Games menu has four menu items under it. The MenuPos for the Whizzy Whirly game is as follows:

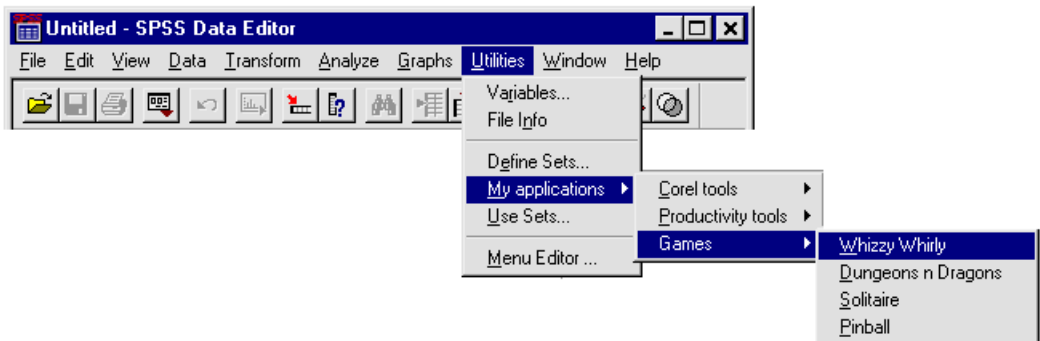


"0, 2, 6"

Zero indicates that Whizzy Whirly is the first item on its menu; 2 indicates that Games is the third item on its menu, and 6 indicates that My Applications is the seventh menu item on its menu (counting the separator).

Figure B-2

SPSS menus displaying Whizzy Whirly menu item



ActionID

Indicates the type of add-in. SPSS supports the running of external applications, internal scripts (typically, *.sbs files), or SPSS syntax files (typically, *.sps files). External applications are usually *.exe files but can be invoked from any valid command line. Therefore, you can use `c:\MSOffice\Excel\Excel.exe`, with your sales data loaded from `c:\mydata\sales1.xls`; or you can simply use `c:\mydata\sales1.xls` to launch Excel with your sales data loaded. A good rule of thumb is that if it works on the Start > Run menu, it will work here. You may need embedded quotes around long filenames.

This key is required by SPSS to identify the type of application to which the LAUNCH keyword applies. Valid values are 0 (application), 1 (script), and 2 (syntax). For example:



(Default)

"1"

SingleSeat

This value is 1 if the add-in works only in single-seat (non-client/server) mode. A value of 0 or a missing key indicates that the application will be available in both single-seat and client/server modes. For example:



(Default)

"0"

TBOnly

This value is 1 if the add-in is to be accessible via the toolbar only (and not via the menu bar). A value of 0 indicates that the application can appear on SPSS menus as well as on toolbars. For example:



(Default)

"0"

Menu

Describes the menu text and the location of the add-in on the menu hierarchy. This entry is in the format MenuName > Menuitem where MenuName is one of the symbols defined below:

Table B-2
Symbols and menu names

Symbol	Menu
\$FILE	File
\$NEW	File > New
\$DATABASE	File > Open Database
\$EDIT	Edit
\$VIEW	View
\$DATA	Data
\$MERGE	Data > Merge Files
\$ORTHO	Data > Orthogonal Design
\$TRANS	Transform
\$RECODE	Transform > Recode
\$ANALYZE	Analyze
\$REPORTS	Analyze > Reports
\$DESCSTATS	Analyze > Descriptive Statistics
\$TABLE	Analyze > Custom Tables
\$MEAN	Analyze > Compare Means
\$ANOVA	Analyze > General Linear Model
\$CORR	Analyze > Correlate
\$REGR	Analyze > Regression
\$LOGLIN	Analyze > Loglinear
\$CLASS	Analyze > Classify
\$REDUCT	Analyze > Data Reduction
\$SCALE	Analyze > Scale
\$NPAR	Analyze > Nonparametric Tests
\$TIMESERIES	Analyze > Time Series
\$SURV	Analyze > Survival
\$MULTRESP	Analyze > Multiple Response
\$GRAPH	Graphs
\$GRAFTIME	Graphs > Time Series

Table B-2 (Continued)
Symbols and menu names

\$UTIL	Utilities
\$HELP	Help
\$CHART	Graphics Editor Chart
\$SERIES	Graphics Editor Series
\$ATTRIBUTES	Graphics Editor Attributes
\$SELECT	Edit > Select
\$OUTLINE	Edit > Outline
\$INSERT	Insert
\$FORMAT	Format

Defining these symbolically makes for easier localization of programs that add menu items to the SPSS menus. For example, a value of \$ANALYZE > My Statistic places a menu item called My statistic on the Analyze menu. Add an ampersand (&) to indicate the accelerator; \$ANALYZE > My Sta&tistic would make Alt-t the accelerator for the add-in.

If you're defining your own menus, you'll need to use actual strings rather than the symbols defined above. For example, &My applications>My Sta&tistic would create a menu, My Application (with "m" as the accelerator) and a menu item of My Statistic below it. It is the responsibility of the third-party application to specify a unique menu item and mnemonic when the registry is updated. (To do so, the vendor must check all other third-party sections within the registration database.) The menu and its items are validated when SPSS is launched.

If a single token is specified, it is assumed to be a menu item at the bottom of the Utilities menu.

If the entire tree already exists and a MenuPos entry does not exist, the item will be placed at the bottom of the last node.

MenuSeparator

Whether or not a separator line is inserted before the menu item. Valid entries are 0 and 1. Zero does not insert a separator; 1 does insert a separator. Defaults to 0. For example:

 (Default) "1"

Minimize

Whether or not the application is launched minimized and whether or not SPSS is minimized when the application is launched. Valid entries are 0, 1, and 2. Zero minimizes neither; 1 minimizes SPSS; 2 minimizes the third-party application. Defaults to 0. For example:

 (Default) "0"

MenuEnable

When the menu item is enabled. Valid entries are 0 and 1. Zero enables the menu item at all times; 1 enables the menu item only while data is present in SPSS. Defaults to 0. For example:

 (Default)

ReadFile

The type of data file passed to the application. The following table describes the types. For example:

 (Default) "1"

Table B-3
File types

Spec	Type	ReadFile
0	DDE	Third-party application will initiate a DDE conversation to read the working data file.
1	No file	No file is written to disk when the third-party application is launched.
2	SPSS data file	SPSS writes the working data file to disk (in the Windows temporary directory). If the menu item is enabled but no working data file exists, the third-party application is launched without a filename.
3	Excel version 2	Spreadsheet type files are written with field names. If the menu item is enabled but no working data file exists, the third-party application is launched without a filename.
4	SYLK	
5	123 Release 3	
6	Tab-Delimited	
7	dBase IV	

This value defaults to 1.

DeleteFile

Whether the file used to pass data to the application is to be deleted when SPSS terminates. If this key has the value 0, the file will not be deleted. If it has the value 1 (the default), the file will be deleted. For example:

 (Default) "1"

Status of Files

When SPSS creates a file, it stores it in the Windows temporary directory. SPSS will then delete that file when it exits unless the DeleteFile key has a value of 0. When the third-party application creates a file for SPSS to read, it should follow the SPSS “temporary” naming conventions so that SPSS will delete the file when it exits.

Coding Conventions

This appendix describes the coding conventions used by SPSS in developing the examples included with the developer's tools. Following a consistent set of guidelines makes your code easier to manage, particularly if more than one person will be working on it. These conventions are included here to aid to interpreting the examples and to use as guidelines for developing your own code.

Following are some general guidelines:

- Declare all variables before using them. (In Visual Basic, the Option Explicit statement can be used to force explicit declaration of all variables.)
- Be consistent in naming variables and procedures. Use standard prefixes to indicate the data type and scope of variables.
- Be generous with comments.
- Indent nested blocks of code to show logic and increase readability.

Variable and Procedure Names

Be consistent when naming variables and procedures. Names should be written in mixed case and should be descriptive. Variable names should use standard prefixes to indicate data type and scope, as shown in Table C-1 and Table C-2. Procedure names should begin with verbs, such as InitNameArray or CloseDialog.

Table C-1

Variable subtypes and suggested prefixes

Subtype	Prefix	Example
Boolean	bln	blnFound
Byte	byt	bytRasterData
Date(Time)	dtm	dtmStart
Double	dbl	dblTolerance
Error	err	errOrderNum
Integer	int	intQuantity
Long	lng	lngDistance
Object	obj	objCurrent
Single	sng	sngAverage
String	str	strFirstName

For frequently used or long terms, use abbreviations to help keep name length reasonable, but be consistent. For example, randomly switching between “Cnt” and “Count” can lead to confusion.

Variable Scope

The scope of variables varies depending on where they are declared. For example, variables declared within a procedure are available only within that procedure. Variables declared at the beginning of a module, above any procedure, are available to all procedures in the module.

Always define variables with the smallest scope possible. However, in cases in which it is necessary to give variables scope beyond a single procedure, you can add a one-letter scope prefix to the variable name, as shown in Table C-2.

Table C-2

Variable scope and suggested prefixes.

Scope	Prefix	Example
Procedure-level	None	dblVelocity
Module-level	m	mdblVelocity

Object Variables

When referencing SPSS objects, use the variable names shown in Table C-3.

Table C-3
SPSS objects and suggested variable names.

Object	Type	Variable Name
SPSS Application	ISpssApp	objSpssApp
SPSS Options	ISpssOptions	objSpssOptions
Documents	ISpssDocuments	objDocuments
Data Document	ISpssDataDoc	objDataDoc
Syntax Document	ISpssSyntaxDoc	objSyntaxDoc
Viewer Document	ISpssOutputDoc	objOutputDoc
Output Items Collection	ISpssItems	objOutputItems
Output Item	ISpssItem	objOutputItem
Chart	ISpssChart	objSpssChart
Text	ISpssRtf	objSpssText
Print Options	ISpssPrintOptions	objPrintOptions
PivotTable	PivotTable	objPivotTable
Footnotes	ISpssFootnotes	objFootnotes
DataCells	ISpssDataCells	objDataCells
LayerLabels	ISpssLayerLabels	objLayerLabels
(Column)Labels	ISpssLabels	objColumnLabels
(Row)Labels	ISpssLabels	objRowLabels
PivotMgr	ISpssPivotMgr	objPivotMgr
Dimension	ISpssDimension	objDimension

Naming Constants

Constant names should be in upper case, with underscores (_) between words. For example:

```
USER_LIST_MAX
NEW_LINE
```

Commenting Code

It is a good idea to begin each procedure with a comment that describes what the procedure does. This description should not provide the implementation details because these often change over time, resulting in unnecessary comment maintenance or, worse yet, erroneous comments. (Use the code itself and any necessary inline comments to describe the implementation.)

- Arguments passed to a procedure should be described when their purpose is not obvious and when the procedure expects the arguments to be within a specific range. Function return values and other variables that are changed by the procedure, especially through reference arguments, should also be described at the beginning of each procedure.
- It may be helpful to provide an overview of each step in a procedure. You need not comment every line; rather, summarize blocks of code that accomplish meaningful steps in the overall procedure.
- Comments should be contained within the procedure to which they pertain.

Code Structure

Organizing code into procedures makes it easier to manage and reuse pieces of code. As a general rule, procedures are organized by task in the sample programs and can be broken up into those that get the objects to be manipulated (for example, a pivot table that is selected in the Viewer) and those that actually perform the manipulations on the objects of interest (for example, making the *Totals* bold).

This structure is intended to make it easy to understand each procedure and to reuse pieces of code. For example, a procedure that gets the first selected pivot table in the Viewer could be used by a number of programs or scripts that manipulate the table in different ways.

Index

- autoscripts, 75
- command syntax
 - creating by copying, 18
 - creating by pasting, 16
 - running procedures with, 14
- description in a script, 74
- developer's tools
 - automation overview, 5
 - compatibility with future versions of SPSS, 6
 - customization overview, 5
 - distributing your application, 6
 - integration overview, 5
 - technical support, 6
- distributed mode
 - overview, 25
- example programs
 - additional sources for, 87
 - analyze data in Excel, 101
 - application object, 40
 - chart object, 60
 - correlation matrix diagonal, 94
 - data document object, 49
 - display reports in Word, 100
 - display, print, and export reports, 98
 - file information object, 46
 - getting versus creating the application object, 41
 - I/O DLL, 107
 - interactive graph object, 63
 - make wide pivot tables narrow, 97
 - manipulate output items, 91
 - manipulate pivot tables, 93
 - multiple instances of SPSS, 89
 - options object, 42
 - output document object, 53
 - output item index, 90
 - output items collection object, 55
 - pivot table object, 58
 - Production Facility code, 103
 - run syntax code, 105
 - scripting, 88
 - shorten percentage labels, 95
 - syntax document object, 51
 - text object, 66
 - Visual Basic version, 87
- example scripts
 - adding an autoscript, 79
 - additional sources for, 87
 - edit all pivot tables, 88
 - modifying a starter script, 77
 - writing an original script, 82
- Excel
 - example program to analyze data, 101
- global scripts, 75
- I/O DLL
 - 16-bit versus 32-bit, 118
 - Borland C++, 119
 - coding with, 118
 - direct access input, 114
 - example programs, 107, 120
 - introduction to, 2
 - Visual Basic, 118
- I/O DLL Procedures

- spssAddVarAttribute, 127
- spssFreeAttributes, 139
- spssGetFileAttributes, 154
- spssGetVarAttributes, 169
- spssSetFileAttributes, 212
- spssSetLocale, 214
- I/O DLL procedures
 - spssAddFileAttribute, 121
 - spssAddMultRespDefC, 122
 - spssAddMultRespDefN, 125
 - spssCloseAppend, 128
 - spssCloseRead, 129
 - spssCloseWrite, 130
 - spssCommitCaseRecord, 131
 - spssCommitHeader, 132
 - spssConvertDate, 133
 - spssConvertSPSSDate, 135
 - spssConvertSPSSTime, 137
 - spssConvertTime, 137
 - spssCopyDocuments, 138
 - spssFreeDateVariables, 140
 - spssFreeMultRespDefs, 140
 - spssFreeVarCValueLabels, 141
 - spssFreeVariableSets, 142
 - spssFreeVarNames, 144
 - spssFreeVarNValueLabels, 143
 - spssGetCaseSize, 144
 - spssGetCaseWeightVar, 145
 - spssGetCompression, 147
 - spssGetDateVariables, 148
 - spssGetDEWFirst, 150
 - spssGetDEWGUID, 150
 - spssGetDewInfo, 151
 - spssGetDEWNext, 152
 - spssGetEstimatedNofCases, 153
 - spssGetIdString, 155
 - spssGetMultRespDefs, 157
 - spssGetNumberOfCases, 158
 - spssGetNumberOfVariables, 159
 - spssGetReleaseInfo, 160
 - spssGetSystemString, 162
 - spssGetTextInfo, 163
 - spssGetTimeStamp, 164
 - spssGetValueChar, 165
 - spssGetValueNumeric, 168
 - spssGetVarAlignment, 169
 - spssGetVarCMissingValues, 170
 - spssGetVarColumnWidth, 173
 - spssGetVarCompatName, 173
 - spssGetVarCValueLabel, 174
 - spssGetVarCValueLabelLong, 175
 - spssGetVarCValueLabels, 177
 - spssGetVarHandle, 178
 - spssGetVariableSets, 179
 - spssGetVarInfo, 181
 - spssGetVarLabel, 182
 - spssGetVarLabelLong, 183
 - spssGetVarMeasureLevel, 184
 - spssGetVarNames, 192
 - spssGetVarNMissingValues, 185
 - spssGetVarNValueLabel, 188
 - spssGetVarNValueLabelLong, 189
 - spssGetVarNValueLabels, 190
 - spssGetVarPrintFormat, 193
 - spssGetVarWriteFormat, 194
 - spssHostSysmisVal, 196
 - spssLowHighVal, 196
 - spssOpenAppend, 197
 - spssOpenRead, 199
 - spssOpenWrite, 200
 - spssOpenWriteCopy, 201
 - spssQueryType7, 202
 - spssReadCaseRecord, 203
 - spssSeekNextCase, 204
 - spssSetCaseWeightVar, 205
 - spssSetCompression, 206
 - spssSetDateVariables, 208
 - spssSetDEWFirst, 209
 - spssSetDEWGUID, 210
 - spssSetDEWNext, 211
 - spssSetIdString, 213
 - spssSetMultRespDefs, 215
 - spssSetTempDir, 215
 - spssSetTextInfo, 216
 - spssSetValueChar, 217
 - spssSetValueNumeric, 218
 - spssSetVarAlignment, 219
 - spssSetVarAttributes, 220
 - spssSetVarCMissingValues, 221
 - spssSetVarColumnWidth, 223
 - spssSetVarCValueLabel, 224
 - spssSetVarCValueLabels, 226
 - spssSetVariableSets, 240
 - spssSetVarLabel, 227

- spssSetVarMeasureLevel, 229
- spssSetVarName, 235
- spssSetVarNMissingValues, 230
- spssSetVarNValueLabel, 232
- spssSetVarNValueLabels, 234
- spssSetVarPrintFormat, 237
- spssSetVarWriteFormat, 238
- spssSysmisVal, 241
- spssValidateVarname, 242
- spssWholeCaseIn, 243
- spssWholeCaseOut, 244
- I/O DLL, SPSS data files and DOCUMENT command, 117
 - string variables, 115
 - system-missing values, 116
 - value labels, 116
 - variable alignment, 116
 - variable column widths, 116
 - variable labels, 116
 - variable measurement levels, 116
 - variable naming conventions, 115
- I/O DLL, using to
 - append cases to an SPSS data file, 112
 - copy a dictionary, 112
 - read an SPSS data file, 113
 - write an SPSS data file, 110
- introduction to
 - I/O DLL, 2
 - MACRO and MATRIX procedures, 3
 - OLE Automation, 2
 - Production Facility, 3
 - scripting facility, 3
 - third-party API, 2
- MACRO and MATRIX procedures
 - documentation, 4
 - introduction to, 3
- Microsoft Excel
 - example program to analyze data, 101
- Microsoft Word
 - example program to display reports, 100
- object browsers
 - accessing online Help from, 71
 - object model, 32
- OLE Automation, 34
 - application object code example, 40
 - application object, getting versus creating, 41
 - chart object code example, 60
 - compared to scripting, 73
 - data document object code example, 49
 - defined, 28
 - example application to display, print, and export reports, 98
 - example of deciding what objects to use in an application, 32
 - example program to analyze Excel data, 101
 - example program to display reports in Word, 100
 - example program to get output item index, 90
 - example program to manage multiple instances of SPSS, 89
 - example program to manipulate correlation matrix diagonal, 94
 - example program to manipulate output items, 91
 - example program to manipulate pivot tables, 93
 - example program to narrow pivot tables, 97
 - example program to shorten percentage labels, 95
 - file information object code example, 46
 - high-level properties and methods, 69
 - interactive graph object code example, 63
 - introduction to, 2
 - object model hierarchy, 32
 - objects and corresponding user interfaces, 31
 - options object code example, 42
 - output document object code example, 53
 - output items collection object code example, 55
 - pivot table object code example, 58
 - syntax document object code example, 51
 - tasks that can be automated, 30
 - terminology, 29
 - text object code example, 66
- pasting syntax to a script window, 74
- Production Facility
 - additional documentation, 4
 - code, 103
 - introduction to, 3
 - overview, 24

- running procedures, 16
- run syntax
 - code, 105
- running procedures
 - with command syntax, 14
 - with dialog boxes, 14
 - with Production Facility, 16
- running scripts, 77
- SCRIPT command syntax, 74
- script window, 74
- scripting
 - adding a description to a script, 74
 - additional examples, 87
 - autoscript, step-by-step example, 79
 - compared to OLE Automation, 73
 - defined, 73
 - example to edit pivot tables, 88
 - modifying a starter script, step-by-step example, 77
 - overview, 22
 - pasting syntax to a script window, 74
 - running scripts, 77
 - SCRIPT command syntax, 74
 - script window features, 74
 - steps to use, 76
 - types of scripts, 75
 - writing a script, step-by-step example, 82
- scripting facility
 - additional documentation, 4
 - introduction to, 3
- scripts
 - autoscripts, defined, 75
 - global, defined, 75
 - starter, defined, 75
- SPSS
 - creating command syntax by copying, 18
 - creating command syntax by pasting, 16
 - distributed mode, 25
 - getting data, 11
 - launching, 10
 - MACRO and MATRIX procedures, 3
 - output item types, 9
 - overview of running an analysis, 9
 - Production Facility, 24
 - running procedures from dialog boxes, 14
 - running procedures with command syntax, 14
 - scripting facility, overview, 22
 - selecting and running a procedure, 12
 - type libraries, 70
 - viewing and manipulating results, 19
 - window types, 8
- SPSS object model, 32
- spssAddMultRespDefExt, 123
- spssFreeMultRespDefStruct, 141
- spssGetMultRespDefByIndex, 157
- starter scripts, 75
- steps to write application code, 34
- syntax
 - creating by copying, 18
 - creating by pasting, 16
 - running procedures with, 14
- technical support, 6
- third-party API
 - introduction to, 2
 - menu names, 253
 - Windows registry, 248
- tutorials, 4
- Visual Basic
 - example application to display, print, and export reports, 98
 - example of detecting that the application object is already running, 41
 - example program to get output item index, 90
 - example program to manage multiple instances of SPSS, 89
 - example program to manipulate correlation matrix diagonal, 94
 - example program to manipulate output items, 91
 - example program to manipulate pivot tables, 93
 - example program to narrow pivot tables, 97

example program to shorten percentage labels, 95
Production Facility code, 103
run syntax code, 105

Word

example program to display reports, 100