Python Reference Guide for IBM SPSS Statistics

Note
Before using this information and the product it supports, read the information in "Notices" on page 261.

Product Information

This edition applies to version 25, release 0, modification 0 of IBM SPSS Statistics and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. Overview

The IBM® SPSS® Statistics - Integration Plug-in for Python provides two interfaces for programming with the Python language within IBM SPSS Statistics on Windows, Linux, Mac OS, and for IBM SPSS Statistics Server.

Python Integration Package
The Python Integration Package provides functions that operate on the IBM SPSS Statistics processor, extending IBM SPSS Statistics command syntax with the full capabilities of the Python programming language. With this interface, you can access IBM SPSS Statistics variable dictionary information, case data, and procedure output. You can submit command syntax to IBM SPSS Statistics for processing, create new variables and new cases in the active dataset, or create new datasets. You can also create output in the form of pivot tables and text blocks, all from within Python code.

Scripting Facility
The Scripting Facility provides Python functions that operate on user interface and output objects. With this interface, you can customize pivot tables, and export items such as charts and tables in various formats. You can also start IBM SPSS Statistics dialog boxes, and manage connections to instances of IBM SPSS Statistics Server, all from within Python code.

The IBM SPSS Statistics - Integration Plug-in for Python is included with IBM SPSS Statistics - Essentials for Python, which is installed by default with your IBM SPSS Statistics product. Essentials for Python also includes Python versions 2.7 and 3.4 on all supported operating systems (Windows, Linux, Mac OS, and UNIX for IBM SPSS Statistics Server) and a set of extension commands that are implemented in Python that provide capabilities beyond what is available with built-in SPSS Statistics procedures.

By default, the Integration Plug-in for Python uses the distributions of Python 2.7 and Python 3.4 that are installed with your IBM SPSS Statistics product (as part of Essentials for Python). They are in the `Python` (contains Python 2.7) and `Python3` directories under the directory where SPSS Statistics is installed. You can specify to use a different installation of Python 2.7 or Python 3.4 on the File Locations tab on the Options dialog (Edit>Options). In distributed analysis mode (requires IBM SPSS Statistics Server), the Python location on the remote server is set from the IBM SPSS Statistics Administration Console. Contact your system administrator for assistance.

Note: The locations for extension commands that are listed in the output from the `SHOW EXTPATHS` command are added to the Python search path when you are accessing Python from within IBM SPSS Statistics. If you develop your own Python modules for use with IBM SPSS Statistics on your computer then you can store your modules in one of those locations.

Related information:

Appendix F, “Python Extension Commands for SPSS Statistics,” on page 259

Running Python Code with Python 2 or Python 3

Chapter 2. Python Integration Package

Introduction to Python Programs

The Python® Integration Package for IBM SPSS Statistics allows you to create **Python programs** that control the flow of command syntax jobs, read and write data, and create custom procedures that generate their own pivot table output. This feature requires the IBM SPSS Statistics - Integration Plug-in for Python, which is installed by default with your IBM SPSS Statistics product.

A companion interface is available for creating **Python scripts** that operate on the IBM SPSS Statistics user interface and manipulate output objects. See the topic "Introduction to Python Scripts" on page 93 for more information.

Python programming features described here are available inside `BEGIN PROGRAM-END PROGRAM` program blocks in command syntax. A program block provides access to all the functionality of the Python programming language, including the functions specific to IBM SPSS Statistics and provided in the Python Integration Package for IBM SPSS Statistics. You can use program blocks to combine the programmability features of Python with all the capabilities of IBM SPSS Statistics by building strings of command syntax that are then executed by IBM SPSS Statistics.

You can also run IBM SPSS Statistics from an external Python process, such as a Python IDE or the Python interpreter. See the topic "Running IBM SPSS Statistics from an External Python Process" on page 9 for more information.

Within a program block, Python is in control, and it knows nothing about IBM SPSS Statistics commands. When the Python Integration Package for IBM SPSS Statistics is loaded, Python knows about the functions provided in the package, but standard IBM SPSS Statistics commands are basically invalid within a program block. For example:

```
BEGIN PROGRAM PYTHON.
FREQUENCIES VARIABLES=var1, var2, var3.
END PROGRAM.
```

will generate an error, because `FREQUENCIES` is not recognized by Python. But since the goal of a program block is typically to generate some command syntax that IBM SPSS Statistics can understand, there must be a way to specify command syntax within a program block. This is done by expressing syntax commands, or parts of commands, as character strings, as in:

```
spss.Submit("FREQUENCIES VARIABLES=var1, var2, var3.")
```

The real power of program blocks comes from the ability to dynamically build strings of command syntax, as in:

```
BEGIN PROGRAM PYTHON.
import spss
string1="DESCRIPTIVES VARIABLES="
N=spss.GetVariableCount()
scaleVarList=[]
for i in xrange(N):
  if spss.GetVariableMeasurementLevel(i)=='scale':
    scaleVarList.append(spss.GetVariableName(i))
string2="."
spss.Submit([string1, ' '.join(scaleVarList), string2])
END PROGRAM.
```

- `spss.GetVariableCount` returns the number of variables in the active dataset.
- `if spss.GetVariableMeasurementLevel(i)=="scale"` is true only for variables with a scale measurement level.
- `scaleVarList.append(spss.GetVariableName(i))` builds a list of variable names that includes only those variables with a scale measurement level.

- `spss.Submit` submits a `DESCRIPTIVES` command to IBM SPSS Statistics that looks something like this:

```
DESCRIPTIVES VARIABLES=
scalevar1 scalevar2 scalevar3...etc.
.
```

Working with Python Program Blocks

Use `SET PRINTBACK ON MPRINT ON` to display the syntax generated by program blocks.

Example

```
SET PRINTBACK ON MPRINT ON.
GET FILE='/examples/data/Employee data.sav'.
BEGIN PROGRAM PYTHON.
import spss
scaleVarList=[]
catVarList=[]
varcount=spss.GetVariableCount()
for i in xrange(varcount):
  if spss.GetVariableMeasurementLevel(i)=='scale':
    scaleVarList.append(spss.GetVariableName(i))
  else:
    catVarList.append(spss.GetVariableName(i))
spss.Submit("""
FREQUENCIES
 /VARIABLES=%s.
DESCRIPTIVES
   /VARIABLES=%s.
""" %(' '.join(catVarList), ' '.join(scaleVarList)))
END PROGRAM.
```

The generated command syntax is displayed in the log in the IBM SPSS Statistics Viewer:

```
225 M>  FREQUENCIES
226 M>    /VARIABLES=gender educ jobcat minority.
227 M>  DESCRIPTIVES
228 M>    /VARIABLES=id bdate salary salbegin jobtime prevexp.
```

Basic Specification for a Python Program Block

The basic specification for a Python program block is `BEGIN PROGRAM PYTHON` (the keyword `PYTHON` can be omitted) followed by one or more Python statements, followed by `END PROGRAM.`

Note: The Python function `sys.exit()` is not supported for use within a program block.

- The first program block in a session should start with the Python function `import spss`, which imports the `spss` module, providing access to the functions in the Python Integration Package for IBM SPSS Statistics. See the topic "Python Functions and Classes" on page 15 for more information.
- Subsequent program blocks in the same session do not require `import spss`, and it is silently ignored if the module has already been imported.

Example

```
DATA LIST FREE /var1.
BEGIN DATA
1
END DATA.
DATASET NAME File1.
BEGIN PROGRAM PYTHON.
import spss
File1N=spss.GetVariableCount()
END PROGRAM.
DATA LIST FREE /var1 var2 var3.
BEGIN DATA
1 2 3
END DATA.
DATASET NAME File2.
BEGIN PROGRAM PYTHON.
File2N=spss.GetVariableCount()
if File2N > File1N:
    message="File2 has more variables than File1."
elif File1N > File2N:
    message="File1 has more variables than File2."
```

```
else:
    message="Both files have the same number of variables."
print message
END PROGRAM.
```

- The first program block contains the `import spss` statement. This statement is not required in the second program block.
- The first program block defines a programmatic variable, *File1N*, with a value set to the number of variables in the active dataset.
- Prior to the second program block, a different dataset becomes the active dataset, and the second program block defines a programmatic variable, *File2N*, with a value set to the number of variables in that dataset.
- Since the value of *File1N* persists from the first program block, the two variable counts can be compared in the second program block.

Syntax Rules

- Within a program block, only statements recognized by the specified programming language are allowed.
- Command syntax generated within a program block must follow **interactive** syntax rules. See the topic for more information.
- Within a program block, each line should not exceed 251 bytes (although syntax generated by those lines can be longer).
- With the IBM SPSS Statistics Batch Facility (available only with IBM SPSS Statistics Server), use the -i switch when submitting command files that contain program blocks. All command syntax (not just the program blocks) in the file must adhere to interactive syntax rules.

Within a program block, the programming language is in control, and the syntax rules for that programming language apply. Command syntax generated from within program blocks must always follow interactive syntax rules. For most practical purposes this means command strings you build in a programming block must contain a period (.) at the end of each command.

Scope and Limitations

- Programmatic variables created in a program block cannot be used outside of program blocks.
- Program blocks cannot be contained within `DEFINE-!ENDDEFINE` macro definitions.
- Program blocks can be contained in command syntax files run via the `INSERT` command, with the default `SYNTAX=INTERACTIVE` setting.
- Program blocks cannot be contained within command syntax files run via the `INCLUDE` command.
- Python variables specified in a given program block persist to subsequent program blocks.
- Python programs (*.py*, *.pyc*) utilizing the `spss` module cannot be run as autoscripts, nor are they intended to be run from Utilities>Run Script.

More information about Python programs and Python scripts is available from the IBM SPSS Statistics Help system, and accessed from Core System>Scripting Facility.

Nested Program Blocks

From within Python, you can submit command syntax containing a `BEGIN PROGRAM` block, thus allowing you to nest program blocks. This can be done by including the nested program block in a separate command syntax file and submitting an `INSERT` command to read in the block. It can also be done by submitting the nested program block from within a user-defined Python function.

Example: Nesting Program Blocks Using the INSERT Command

```
import spss
spss.Submit("INSERT FILE='/myprograms/nested_block.sps'.")
```

The file */myprograms/nested_block.sps* would contain a `BEGIN PROGRAM` block, as in:

```
BEGIN PROGRAM PYTHON.
import spss
<Python code>
END PROGRAM.
```

Note: You cannot import a Python module containing code that nests a program block, such as the above code that uses the INSERT command to insert a file containing a program block. If you wish to encapsulate nested program blocks in a Python module that can be imported, then embed the nesting code in a user-defined function as shown in the following example.

Example: Nesting Program Blocks With a User-Defined Python Function

```
import spss, myfuncs
myfuncs.demo()
```

- myfuncs is a user-defined Python module containing the function (demo) that will submit the nested program block.

  A Python module is simply a text file containing Python definitions and statements. You can create a module with a Python IDE, or with any text editor, by saving a file with an extension of *.py*. The name of the file, without the *.py* extension, is then the name of the module.
- The import statement includes myfuncs so that it is loaded along with the spss module. To be sure that Python can find your module, you may want to save it to your Python "site-packages" directory, typically */Python27/Lib/site-packages*.
- The code myfuncs.demo() calls the function demo in the myfuncs module.

Following is a sample of the contents of myfuncs.

```
import spss
def demo():
   spss.Submit("""
BEGIN PROGRAM PYTHON.
<Python code>
END PROGRAM.""")
```

- The sample myfuncs module includes an import spss statement. This is necessary since a function in the module makes use of a function from the spss module--specifically, the Submit function.
- The nested program block is contained within a Python triple-quoted string. Triple-quoted strings allow you to specify a block of commands on multiple lines, resembling the way you might normally write command syntax.
- Notice that spss.Submit is indented but the BEGIN PROGRAM block is not. Python statements that form the body of a user-defined Python function must be indented. The level of indentation is arbitrary but must be the same for all statements in the function body. The BEGIN PROGRAM block is passed as a string argument to the Submit function and is processed by IBM SPSS Statistics as a block of Python statements. Python statements are not indented unless they are part of a group of statements, as in a function or class definition, a conditional expression, or a looping structure.

Notes

- You can have up to five levels of nesting.
- Python variables specified in a nested program block are local to that block unless they are specified as global variables. In addition, Python variables specified in a program block that invokes a nested block can be read, but not modified, in the nested block.
- Nested program blocks can be Python program blocks or R program blocks.
- If a Submit function containing a triple quoted string nests a Python program block containing another triple quoted string, use a different type of triple quotes in the nested block. For example, if the outer block uses triple double quotes, then use triple single quotes in the nested block.

Unicode Mode

When IBM SPSS Statistics is in Unicode mode (controlled by the UNICODE subcommand of the SET command) the following conversions are automatically done when passing and receiving strings through the functions available with the spss module:

- Strings received by Python from IBM SPSS Statistics are converted from UTF-8 to Python Unicode, which is UTF-16.
- Strings passed from Python to IBM SPSS Statistics are converted from UTF-16 to UTF-8.

Note: Changing the locale and/or the unicode setting during an `OMS` request may result in incorrectly transcoded text.

Command Syntax Files

Special care must be taken when working in Unicode mode with command syntax files. Specifically, Python string literals used in command syntax files need to be explicitly expressed as UTF-16 strings. This is best done by using the `u()` function from the `spssaux` module (installed with IBM SPSS Statistics - Essentials for Python). The function has the following behavior:

- If IBM SPSS Statistics is in Unicode mode, the input string is converted to UTF-16.
- If IBM SPSS Statistics is not in Unicode mode, the input string is returned unchanged.

Note: If the string literals in a command syntax file only consist of plain roman characters (7-bit ascii), the `u()` function is not needed.

The following example demonstrates some of this behavior and the usage of the `u()` function.

```
set unicode on locale=english.
BEGIN PROGRAM.
import spss, spssaux
from spssaux import u
literal = "âbc"
try:
   print "literal without conversion:", literal
except:
   print "can't print literal"
try:
   print "literal converted to utf-16:", u(literal)
except:
   print "can't print literal"
END PROGRAM.
```

Following are the results:

```
literal without conversion: can't print literal
literal converted to utf-16: âbc
```

Truncating Unicode Strings

When working in Unicode mode, use the `truncatestring` function from the `spssaux` module (installed with IBM SPSS Statistics - Essentials for Python) to correctly truncate a string to a specified maximum length in bytes. This is especially useful for truncating strings to be used as IBM SPSS Statistics variable names, which have a maximum allowed length of 64 bytes.

The `truncatestring` function takes two arguments--the string to truncate, and the maximum number of bytes, which is optional and defaults to 64. For example:

```
import spss, spssaux
newstring = spssaux.truncatestring(string,8)
```

Python Syntax Rules

Within a Python program block, only statements and functions recognized by Python are allowed. Python syntax rules differ from IBM SPSS Statistics command syntax rules in a number of ways:

Python is case-sensitive. This includes variable names, function names, and pretty much anything else you can think of. A variable name of *myvariable* is not the same as *MyVariable*, and the function `spss.GetVariableCount` cannot be written as `SPSS.getvariablecount`.

Python uses UNIX-style path specifications, with forward slashes. This applies even for IBM SPSS Statistics command syntax generated within a Python program block. For example:

```
spss.Submit("GET FILE '/data/somedata.sav'.")
```

Alternatively, you can escape each backslash with another backslash, as in:

```
spss.Submit("GET FILE '\\data\\somedata.sav'.")
```

There is no command terminator in Python, and continuation lines come in two flavors:

- **Implicit.** Expressions enclosed in parentheses, square brackets, or curly braces can continue across multiple lines without any continuation character. The expression continues implicitly until the closing character for the expression.
- **Explicit.** All other expression require a backslash at the end of each line to explicitly denote continuation.

Line indentation indicates grouping of statements. Groups of statements contained in conditional processing and looping structures are identified by indentation, as is the body of a user-defined Python function. There is no statement or character that indicates the end of the structure. Instead, the indentation level of the statements defines the structure, as in:

```
for i in xrange(varcount):
  if spss.GetVariableMeasurementLevel(i)=="scale":
      ScaleVarList=ScaleVarList + " " + spss.GetVariableName(i)
  else:
     CatVarList=CatVarList + " " + spss.GetVariableName(i)
print CatVarList
```

Note: You should avoid the use of tab characters in Python code within `BEGIN PROGRAM-END PROGRAM` blocks. For line indentation, use spaces.

Working with Multiple Versions of IBM SPSS Statistics

For versions 16.0 to 21.0, special considerations apply when multiple versions of the IBM SPSS Statistics - Integration Plug-in for Python (each associated with a major version of IBM SPSS Statistics, such as 20 or 21) are installed on your computer.

Running Python Programs from Within IBM SPSS Statistics

Important: This section only applies to versions 16.0 to 21.0.

By default, Python programs run from within the last installed version of IBM SPSS Statistics will automatically use the appropriate version of the plug-in. To run Python programs from within a different version of IBM SPSS Statistics, use the spss.SetDefaultPlugInVersion function to set the default to a different version (the setting persists across sessions). You can then run Python programs from within the other version. If you are attempting to change the default version from 16.0 to 17.0, additional configuration is required; please see the Notes below.

Running Python Programs from an External Python Process

Important: This section only applies to versions 16.0 to 21.0. For version 22 and higher, see "Running IBM SPSS Statistics from an External Python Process" on page 9.

When you are driving the IBM SPSS Statistics backend from a separate Python process, such as the Python interpreter or a Python IDE, the plug-in will drive the version of the IBM SPSS Statistics backend that matches the default plug-in version specified for that version of Python. Unless you change it, the default plug-in version for a given version of Python (such as Python 2.6) is the last one installed. You can view the default version using the spss.GetDefaultPlugInVersion function and you can change the default version using the spss.SetDefaultPlugInVersion function. The setting persists across sessions. If you are attempting to change the default version from 16.0 to 17.0 please see the Notes below.

Note:

- If you are using the `spss.SetDefaultPlugInVersion` function to change the default from version 16.0 to version 17.0, you should also manually modify the file *SpssClient.pth* located in the Python 2.5 *site-packages* directory. Change the order of entries in the file so that the first line is `SpssClient170`.

  Windows. The *site-packages* directory is located in the *Lib* directory under the Python 2.5 installation directory—for example, *C:\Python25\Lib\site-packages*.

  Mac OS X 10.4 (Tiger). The *site-packages* directory is located at */Library/Frameworks/Python.framework/Versions/2.5/lib/python2.5/site-packages*.

  Mac OS X 10.5 (Leopard). The *site-packages* directory is typically located at */Library/Python/2.5/site-packages*.

  Linux and UNIX Server. The *site-packages* directory is located in the */lib/python2.5/* directory under the Python 2.5 installation directory—for example, */usr/local/python25/lib/python2.5/site-packages*.
- Beginning with version 15.0, a restructuring of the IBM SPSS Statistics - Integration Plug-in for Python installation directory and changes to some class structures may affect Python code written for an earlier version and used with a 15.0 or higher version. Specifically, the type of an object, as given by the Python `type` function, may return a different result. For example:

  ```
  cur=spss.Cursor()
  print type(cur)
  ```

  will return `spss.cursors.Cursor` when run with version 14.0, `spss.spss150.cursors.ReadCursor` when run with version 15.0, and `spss.cursors.ReadCursor` when run with a version higher than 15.0.

Python and IBM SPSS Statistics Working Directories

When running Python code that is within a `BEGIN PROGRAM-END PROGRAM` block and that contains relative paths in file specifications, you will need to understand the notions of working directories, both for Python and IBM SPSS Statistics. You may want to avoid the subtleties involved with working directories by avoiding the use of relative paths and using full paths for file specifications.

- Relative paths used for file specifications in command syntax submitted from Python (with `spss.Submit`) are relative to the IBM SPSS Statistics backend working directory. The IBM SPSS Statistics backend working directory determines the full path used for file specifications in command syntax in the case where only a relative path is provided. It can be changed with the `CD` command, but is not affected by actions involving the file open dialogs, and it is private to the IBM SPSS Statistics backend.
- Relative paths used when reading and writing files with built-in Python functions--such as `open`--are relative to the Python current working directory. You can get the Python current working directory from the `getcwd` function in the `os` module.

Running IBM SPSS Statistics from an External Python Process

You can run Python programs utilizing the `spss` module from any external Python process, such as a Python IDE or the Python interpreter. In this mode, the Python program starts up a new instance of the IBM SPSS Statistics processor without an associated instance of the IBM SPSS Statistics client. You can use this mode to debug your Python programs using the Python IDE of your choice.

To drive the IBM SPSS Statistics processor from a Python IDE, simply include an `import spss` statement in the IDE's code window, followed by a call to the `spss.StartSPSS` function. You can then call any of the functions in the `spss` module, just like with program blocks in command syntax jobs, but you do not need to wrap your Python code in `BEGIN PROGRAM-END PROGRAM` statements. Some initial configuration may be required as described in the following sections.

Windows Users

You can start IDLE (the default IDE provided with Python) for Python 2.7 or Python 3.4 from All Programs > IBM SPSS Statistics > Python 2.7 for IBM SPSS Statistics 25 > Python 2.7 IDLE(PythonGUI) or All Programs > IBM SPSS Statistics > Python 3.4 for IBM SPSS Statistics 25 > Python 3.4

IDLE(PythonGUI). This action starts IDLE from the Python location that is specified on the File Locations tab in the Options dialog. You can then use `import spss` to start driving the IBM SPSS Statistics processor.

If you do not use IDLE to drive IBM SPSS Statistics, then you must modify the Python search path to add the path to the `spss` module. You can add to the Python search path by modifying (or creating) the `sitecustomize.py` module for the installation of Python 2.7 or Python 3.4 that you want to use. The `sitecustomize.py` module, if it exists, is in the `<PYTHON_HOME>\Lib\site-packages` directory, where `<PYTHON_HOME>` is the installation location of Python 2.7 or Python 3.4; for example, `C:\Python27\Lib\site-packages`. If it does not exist then create it in that location.

For Python 2.7, add the following lines to `sitecustomize.py`:

```
import sys
sys.path.insert(0,r'<SPSS_HOME>\Python\Lib\site-packages')
```

For Python 3.4, add the following lines to `sitecustomize.py`:

```
import sys
sys.path.insert(0,r'<SPSS_HOME>\Python3\Lib\site-packages')
```

In the preceding expressions, `<SPSS_HOME>` is the installation location of IBM SPSS Statistics; for example, `C:\Program Files\IBM\SPSS\Statistics\25`.

Linux Users

The `statisticspython` script (for Python 2) or the `statisticspython3` script (for Python 3), in the `bin` directory under the location where IBM SPSS Statistics is installed, starts the Python interpreter from the Python location that is specified on the File Locations tab in the Options dialog. You can then use `import spss` to start driving the IBM SPSS Statistics processor.

If you choose not to use this script, then you must modify the Python search path to add the path to the `spss` module. You can add to the Python search path by modifying (or creating) the `sitecustomize.py` module for the installation of Python 2.7 or Python 3.4 that you want to use. The `sitecustomize.py` module, if it exists, is in the `<PYTHON_HOME>/lib/python2.7/site-packages` or `<PYTHON_HOME>/lib/python3.4/site-packages` directory, where `<PYTHON_HOME>` is the installation location of Python 2.7 or Python 3.4. If it does not exist then create it in that location.

For Python 2.7, add the following lines to `sitecustomize.py`:

```
import sys
sys.path.insert(0,'<SPSS_HOME>/Python/lib/python2.7/site-packages')
```

For Python 3.4, add the following lines to `sitecustomize.py`:

```
import sys
sys.path.insert(0,'<SPSS_HOME>/Python3/lib/python3.4/site-packages')
```

In the preceding expression, `<SPSS_HOME>` is the installation location of IBM SPSS Statistics; for example, `/opt/IBM/SPSS/Statistics/25`.

You must also modify the LD_LIBRARY_PATH environment variable as follows:

```
export LD_LIBRARY_PATH=<PYTHON_HOME>/lib:<SPSS_HOME>/lib:$LD_LIBRARY_PATH
```

In the preceding expression, `<PYTHON_HOME>` is the location of the installation of Python 2.7 or Python 3.4 that you want to use. For reference, for the version of Python 2.7 that is installed with IBM SPSS Statistics, `<PYTHON_HOME>` is `<SPSS_HOME>/Python`. And, for Python 3.4, `<PYTHON_HOME>` is `<SPSS_HOME>/Python3`.

Mac Users

To drive the IBM SPSS Statistics processor from an external Python 2 or Python 3 process on Mac, start the *Python2 for SPSS Statistics* or *Python3 for SPSS Statistics* application, which are in the directory where IBM SPSS Statistics is installed. The applications start IDLE from the Python location that is specified on the File Locations tab in the Options dialog. You can then use `import spss` to start driving the IBM SPSS Statistics processor.

If you choose not to use the *Python2 for SPSS Statistics* or *Python3 for SPSS Statistics* application, then you must modify the Python search path to add the path to the `spss` module. You can add to the Python search path by modifying (or creating) the `sitecustomize.py` module for the installation of Python 2.7 or Python 3.4 that you want to use. The `sitecustomize.py` module, if it exists, is in the `<PYTHON_HOME>/lib/python2.7/site-packages` or `<PYTHON_HOME>/lib/python3.4/site-packages` directory, where `<PYTHON_HOME>` is the installation location of Python 2.7 or Python 3.4. If it does not exist then create it in that location.

For Python 2.7, add the following lines to `sitecustomize.py`:

```
import sys
sys.path.insert(0,'<INSTALLDIR>/Python/lib/python2.7/site-packages')
```

For Python 3.4, add the following lines to `sitecustomize.py`:

```
import sys
sys.path.insert(0,'<INSTALLDIR>/Python3/lib/python3.4/site-packages')
```

In the preceding expression, `<INSTALLDIR>` is the location of the IBM SPSS Statistics application bundle; for example, `/Applications/IBM/SPSS/Statistics/25`.

You must also modify environment variables as follows:

```
export
DYLD_LIBRARY_PATH=<SPSS_HOME>/lib:<SPSS_HOME>/Library/Frameworks/Sentinel.framework/Versions/A:
<SPSS_HOME>/Library/Frameworks/SuperPro.framework/Versions/A
export PYTHONHOME=<PYTHON_HOME>
```

In the preceding expression, `<SPSS_HOME>` is the location of the Contents folder in the IBM SPSS Statistics application bundle, and is given by `<INSTALLDIR>/SPSSStatistics.app/Contents`. `<PYTHON_HOME>` is the location of the installation of Python 2.7 or Python 3.4 that you want to use. For reference, for the version of Python 2.7 that is installed with IBM SPSS Statistics, `<PYTHON_HOME>` is `<INSTALLDIR>/Python`. And, for Python 3.4, `<PYTHON_HOME>` is `<INSTALLDIR>/Python3`.

Related information:

"spss.StartSPSS Function" on page 88

Localizing Output from Python Programs

You can localize output, such as messages and pivot table strings, from extension commands implemented in Python. The localization process consists of the following steps:

1. Modifying the Python implementation code to identify translatable strings
2. Extracting translatable text from the implementation code using standard Python tools
3. Preparing a translated file of strings for each target language
4. Installing the translation files along with the extension command

The process described here assumes use of the Python `extension` module, which is installed with IBM SPSS Statistics - Essentials for Python.

Notes

- When running an extension command from within IBM SPSS Statistics, the language for extension command output will be automatically synchronized with the IBM SPSS Statistics output language

(OLANG). When running an extension command from an external Python process, such as a Python IDE, you can set the output language by submitting a SET OLANG command when IBM SPSS Statistics is started. If no translation for an item is available for the output language, the untranslated string will be used.

- Messages produced by the extension module, such as error messages for violation of the specifications in the Syntax definition, are automatically produced in the current output language. Exceptions raised in the extension command implementation code are automatically converted to a Warnings pivot table.
- Translation of dialog boxes built with the Custom Dialog Builder is a separate process, but translators should ensure that the dialog and extension command translations are consistent.

Additional Resources

Examples of extension commands implemented in Python with localized output are included with IBM SPSS Statistics - Essentials for Python. The Python modules for these examples can be found in the location where extension commands are installed on your computer. To view the location, run the SHOW EXTPATHS syntax command. The output displays a list of locations under the heading "Locations for extension commands". The files are installed to the first writable location in the list.

Information on creating extension commands is also available from the following sources:

- The article *"Writing IBM SPSS Statistics Extension Commands"*, available from the IBM SPSS Predictive Analytics community at https://developer.ibm.com/predictiveanalytics/.
- The chapter on Extension Commands in *Programming and Data Management for IBM SPSS Statistics*, which is also available from the IBM SPSS Predictive Analytics community.

Modifying the Python code

First, ensure that the text to be translated is in a reasonable form for translation.

- Do not build up text by combining fragments of text in code. This makes it impossible to rearrange the text according to the grammar of the target languages and makes it difficult for translators to understand the context of the strings.
- Avoid using multiple parameters in a string. Translators may need to change the parameter order.
- Avoid the use of abbreviations and colloquialisms that are difficult to translate.

Enclose each translatable string in a call to the underscore function "_". For example:

```
_("File not found: %s") % filespec
```

The _ function will fetch the translation, if available, when the statement containing the string is executed. The following limitations apply:

- Never pass an empty string as the argument to _, i.e., _(""). This will damage the translation mechanism.
- Do not use the underscore function in static text such as class variables. The _ function is defined dynamically.
- The _ function, as defined in the extension module, always returns Unicode text even if IBM SPSS Statistics is running in code page mode. If there are text parameters in the string as in the example above, the parameter should be in Unicode. The automatic conversion used in the parameter substitution logic will fail if the parameter text contains any extended characters. One way to resolve this is as follows, assuming that the locale module has been imported.

```
if not isinstance(filespec, unicode):
   filespec = unicode(filespec, locale.getlocale()[1])
   _("File not found: %s") % filespec
```

Note: There is a conflict between the definition of the _ function as used by the Python modules (`pygettext` and `gettext`) that handle translations, and the automatic assignment of interactively generated expression values to the variable `_`. In order to resolve this, the translation initialization code in the `extension` module disables this assignment.

Calls to the `spss.StartProcedure` function (or the `spss.Procedure` class) should use the form `spss.StartProcedure(procedureName,omsIdentifier)` where *procedureName* is the translatable name associated with output from the procedure and *omsIdentifier* is the language invariant `OMS` command identifier associated with the procedure. For example:

```
spss.StartProcedure(_("Demo"),"demoId")
```

Extracting translatable text

The Python implementation code is never modified by the translators. Translation is accomplished by extracting the translatable text from the code files and then creating separate files containing the translated text, one file for each language. The _ function uses compiled versions of these files.

The standard Python distribution includes `pygettext.py`, which is a command line script that extracts strings marked as translatable (i.e., strings wrapped in the _ function) and saves them to a *.pot* file. Run `pygettext.py` on the implementation code, and specify the name of the implementing Python module (the module containing the `Run` function) as the name of the output file, but with the extension *.pot*. If the implementation uses multiple Python files, the *.pot* files for each should be combined into one under the name of the main implementing module (the module containing the `Run` function).

- Change the *charset* value, in the `msgstr` field corresponding to `msgid ""`, to utf-8.
- A *pot* file includes one `msgid` field with the value `""`, with an associated `msgstr` field containing metadata. There must be only one of these.
- Optionally, update the generated title and organization comments.

Documentation for `pygettext.py` is available from the topic on the `gettext` module in the Python help system.

Translating the pot file

Translators enter the translation of each `msgid` into the corresponding `msgstr` field and save the result as a file with the same name as the *pot* file but with the extension *.po*. There will be one *po* file for each target language.

- *po* files should be saved in Unicode utf-8 encoding.
- *po* files should not have a BOM (Byte Order Mark) at the start of the file.
- If a `msgstr` contains an embedded double quote character (x22), precede it with a backslash (\). as in:
  `msgstr "He said, \"Wow\", when he saw the R-squared"`
- `msgid` and `msgstr` entries can have multiple lines. Enclose each line in double quotes.

Each translated *po* file is compiled into a binary format by running `msgfmt.py` from the standard Python distribution, giving the output the same name as the *po* file but with an extension of *.mo*.

Installing the mo files

When installed, the `mo` files should reside in the following directory structure:

lang/<language-identifier>/LC_MESSAGES/<command name>.mo

- <*command name*> is the name of the extension command in upper case with any spaces replaced with underscores, and is the same as the name of the Python implementation module. Note that the `mo` files have the same name for all languages.
- <*language-identifier*> is the identifier for a particular language. Identifiers for the languages supported by IBM SPSS Statistics are shown in the section on Language Identifiers at the end of this topic.

For example, if the extension command is named *MYORG MYSTAT* then an *mo* file for French should be stored in *lang/fr/LC_MESSAGES/MYORG_MYSTAT.mo*.

Manually installing translation files

If you are manually installing an extension command and associated translation files, then the *lang* directory containing the translation files should be installed in the <*command name*> directory under the directory where the Python implementation module is installed.

For example, if the extension command is named *MYORG MYSTAT* and the associated Python implementation module (*MYORG_MYSTAT.py*) is located in the *extensions* directory (under the location where IBM SPSS Statistics is installed), then the *lang* directory should reside under *extensions/MYORG_MYSTAT*.

Using the example of a French translation discussed above, an *mo* file for French would be stored in *extensions/MYORG_MYSTAT/lang/fr/LC_MESSAGES/MYORG_MYSTAT.mo*.

Deploying translation files to other users

If you are localizing output for a custom dialog or extension command that you intend to distribute to other users, then you should create an extension bundle (requires IBM SPSS Statistics version 18 or higher) to package your translation files with your custom components. Specifically, you add the *lang* directory containing your compiled translation files (*mo* files) to the extension bundle during the creation of the bundle (from the Translation Catalogues Folder field on the Optional tab of the Create Extension Bundle dialog). When an end user installs the extension bundle, the directory containing the translation files is installed in the *extensions/<extension bundle name>* directory under the IBM SPSS Statistics installation location, and where <*extension bundle name*> is the name of the extension bundle with spaces replaced by underscores. *Note*: An extension bundle that includes translation files for an extension command should have the same name as the extension command.

- If the *SPSS_EXTENSIONS_PATH* environment variable has been set, then the *extensions* directory (in *extensions/<extension bundle name>*) is replaced by the first writable directory in the environment variable.
- Information on creating extension bundles is available from the Help system, under Core System>Utilities>Working with Extension Bundles.

Language Identifiers

de. German

en. English

es. Spanish

fr. French

it. Italian

ja. Japanese

ko. Korean

pl. Polish

pt_BR. Brazilian Portuguese

ru. Russian

zh_CN. Simplified Chinese

zh_TW. Traditional Chinese

Python Functions and Classes

The Python Integration Package for IBM SPSS Statistics contains functions and classes that facilitate the process of using Python programming features with IBM SPSS Statistics, including those that:

Build and run command syntax
- spss.Submit

Get information about data files in the current IBM SPSS Statistics session
- spss.GetCaseCount
- spss.GetDataFileAttributes
- spss.GetFileHandles
- spss.GetMultiResponseSet
- spss.GetSplitVariableNames
- spss.GetVarAttributes
- spss.GetVariableCount
- spss.GetVariableFormat
- spss.GetVariableLabel
- spss.GetVariableMeasurementLevel
- spss.GetVariableName
- spss.GetVariableType
- spss.GetVarMissingValues
- spss.GetWeightVar

Get data, add new variables, and append cases to the active dataset
- spss.Cursor

Access and manage multiple datasets
- spss.ActiveDataset
- spss.Dataset
- spss.GetDatasets
- spss.GetFileHandles
- spss.IsActive
- spss.SetActive

Get output results
- spss.EvaluateXPath
- spss.GetXmlUtf16

Create custom pivot tables and text blocks
- spss.BasePivotTable
- spss.TextBlock

Create macro variables

- spss.SetMacroValue

Get error information

- spss.GetLastErrorLevel
- spss.GetLastErrorMessage

Manage multiple versions of the IBM SPSS Statistics - Integration Plug-in for Python

- spss.GetDefaultPlugInVersion
- spss.SetDefaultPlugInVersion
- spss.ShowInstalledPlugInVersions

Locale and Output Language Settings

- spss.GetSPSSLocale
- spss.SetOutputLanguage

Brief descriptions of each function are available using the Python `help` function, as in:

```
BEGIN PROGRAM.
import spss
help(spss.Submit)
END PROGRAM.
```

spss.ActiveDataset Function

spss.ActiveDataset(). *Returns the name of the active dataset.*

- If the active dataset is unnamed, '*' is returned.

Example

```
import spss
name = spss.ActiveDataset()
```

spss.AddProcedureFootnotes Function

spss.AddProcedureFootnotes(footnote). *Adds a footnote to all tables generated by a procedure.* The argument *footnote* is a string specifying the footnote.

- The `AddProcedureFootnotes` function can only be used within a `StartProcedure-EndProcedure` block or within a custom procedure class based on the `spss.BaseProcedure` class.

Example

```
import spss
spss.StartProcedure("mycompany.com.demoProc")
spss.AddProcedureFootnotes("A footnote")
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")
table.SimplePivotTable(cells = [1,2,3,4])
spss.EndProcedure()
```

spss.BasePivotTable Class

spss.BasePivotTable(title,templateName,outline,isSplit,caption). *Provides the ability to create custom pivot tables that can be displayed in the IBM SPSS Statistics Viewer or written to an external file using the IBM SPSS Statistics Output Management System.*

- The argument *title* is a string that specifies the title that appears with the table. Each table associated with a set of output (as specified in a `StartProcedure-EndProcedure` block) should have a unique *title*. Multiple tables within a given procedure can, however, have the same value of the *title* argument as long as they have different values of the *outline* argument.
- The argument *templateName* is a string that specifies the OMS (Output Management System) table subtype for this table. It must begin with a letter and have a maximum of 64 characters. Unless you are

routing this pivot table with OMS, you will not need to keep track of this value, although you do have to provide a value that meets the stated requirements.

Note: Specifying "Warnings" for *templateName* will generate an IBM SPSS Statistics Warnings table. Unless you want to generate an IBM SPSS Statistics Warnings table, you should avoid specifying "Warnings" for *templateName*. See the topic "Creating a Warnings Table" on page 36 for more information.

- The optional argument *outline* is a string that specifies a title, for the pivot table, that appears in the outline pane of the Viewer. The item for the table itself will be placed one level deeper than the item for the *outline* title. If omitted, the Viewer item for the table will be placed one level deeper than the root item for the output containing the table.
- The optional Boolean argument *isSplit* specifies whether to enable split processing when creating pivot tables from data that have splits. By default, split processing is enabled. To disable split processing for pivot tables, specify `isSplit=False`. If you are creating a pivot table from data that has splits and you want separate results displayed for each split group, you will want to make use of the spss.SplitChange function. In the absence of calls to `spss.SplitChange`, *isSplit* has no effect.
- The optional argument *caption* is a string that specifies a table caption.

An instance of the `BasePivotTable` class can only be used within a `StartProcedure-EndProcedure` block or within a custom procedure class based on the `spss.BaseProcedure` class. For an example of creating a pivot table using `spss.StartProcedure-spss.EndProcedure`, see "Creating Pivot Tables with the SimplePivotTable Method" on page 18. For an example of creating a pivot table using a class based on the `spss.BaseProcedure` class, see "spss.BaseProcedure Class" on page 37.

The figure below shows the basic structural components of a pivot table. Pivot tables consists of one or more dimensions, each of which can be of the type row, column, or layer. In this example, there is one dimension of each type. Each dimension contains a set of categories that label the elements of the dimension--for instance, row labels for a row dimension. A layer dimension allows you to display a separate two-dimensional table for each category in the layered dimension--for example, a separate table for each value of minority classification, as shown here. When layers are present, the pivot table can be thought of as stacked in layers, with only the top layer visible.

Each cell in the table can be specified by a combination of category values. In the example shown here, the indicated cell is specified by a category value of *Male* for the *Gender* dimension, *Custodial* for the *Employment Category* dimension, and *No* for the *Minority Classification* dimension.

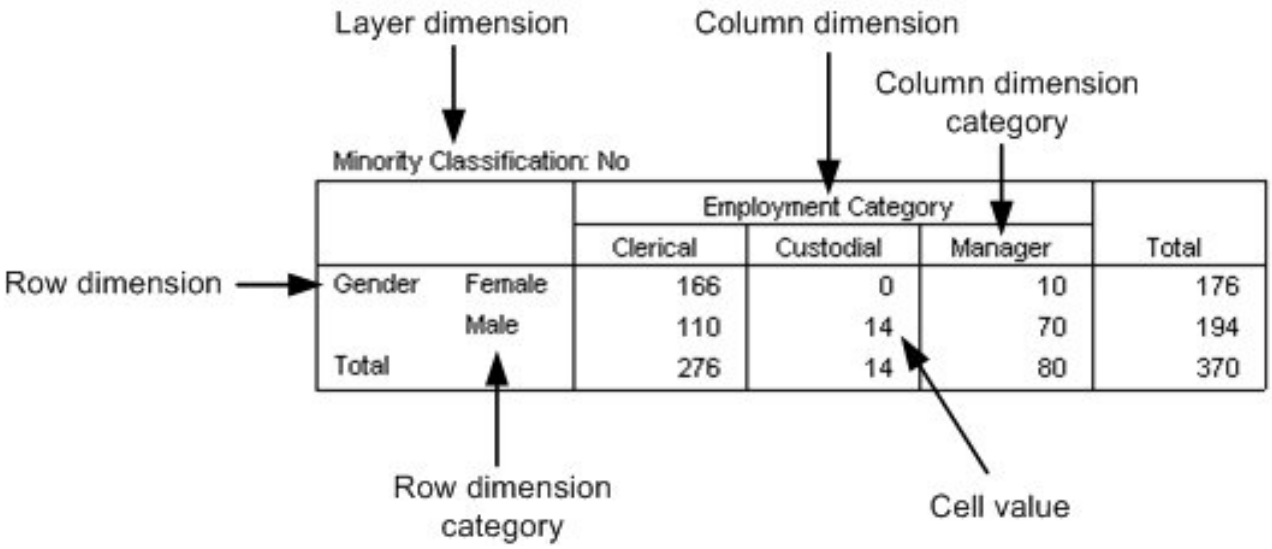


Figure 1. Pivot table structure

Creating Pivot Tables with the SimplePivotTable Method

For creating a pivot table with a single row dimension and a single column dimension, the `BasePivotTable` class provides the SimplePivotTable method. The arguments to the method provide the dimensions, categories, and cell values. No other methods are necessary in order to create the table structure and populate the cells. If you require more functionality than the `SimplePivotTable` method provides, there are a variety of methods to create the table structure and populate the cells. See the topic "General Approach to Creating Pivot Tables" on page 19 for more information.

Example

```
import spss
spss.StartProcedure("mycompany.com.demoProc")
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")

table.SimplePivotTable(rowdim = "row dimension",
                       rowlabels = ["first row","second row"],
                       coldim = "column dimension",
                       collabels = ["first column","second column"],
                       cells = [11,12,21,22])
spss.EndProcedure()
```

Result

Table Title

	column dimension	
row dimension	first column	second column
first row	11	12
second row	21	22

Figure 2. Simple pivot table

- This example shows how to generate a pivot table within a `spss.StartProcedure-spss.EndProcedure` block. The argument to the `StartProcedure` function specifies a name to associate with the output. This is the name that appears in the outline pane of the Viewer associated with the output--in this case, *mycompany.com.demoProc*. It is also the command name associated with this output when routing output with OMS.

  Note: In order that names associated with output do not conflict with names of existing IBM SPSS Statistics commands (when working with OMS), it is recommended that they have the form *yourcompanyname.com.procedurename*. See the topic "spss.StartProcedure Function" on page 85 for more information.
- You create a pivot table by first creating an instance of the `BasePivotTable` class and storing it to a variable--in this case, the variable *table*.
- The `SimplePivotTable` method of the `BasePivotTable` instance is called to create the structure of the table and populate its cells. Row and column labels and cell values can be specified as character strings or numeric values. They can also be specified as a `CellText` object. `CellText` objects allow you to specify that category labels be treated as variable names or variable values, or that cell values be displayed in one of the numeric formats used in IBM SPSS Statistics pivot tables, such as the format for a mean. When you specify a category as a variable name or variable value, pivot table display options such as display variable labels or display value labels are honored.
- Numeric values specified for cell values, row labels, or column labels, are displayed using the default format for the pivot table. Instances of the `BasePivotTable` class have an implicit default format of `GeneralStat`. You can change the default format using the SetDefaultFormatSpec method.
- `spss.EndProcedure` marks the end of output creation.

General Approach to Creating Pivot Tables

The BasePivotTable class provides a variety of methods for creating pivot tables that cannot be created with the SimplePivotTable method. The basic steps for creating a pivot table are:

1. Create an instance of the BasePivotTable class.
2. Add dimensions.
3. Define categories.
4. Set cell values.

Once a cell value has been set, you can access its value. This is convenient for cell values that depend on the value of another cell. See the topic "Using Cell Values in Expressions" on page 23 for more information.

Related information:

"Step 1: Adding Dimensions"

"Step 2: Defining Categories" on page 20

"Step 3: Setting Cell Values" on page 21

Step 1: Adding Dimensions: You add dimensions to a pivot table with the Append or Insert method.

Example: Using the Append Method

```
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")
coldim=table.Append(spss.Dimension.Place.column,"coldim")
rowdim1=table.Append(spss.Dimension.Place.row,"rowdim-1")
rowdim2=table.Append(spss.Dimension.Place.row,"rowdim-2")
```

- The first argument to the Append method specifies the type of dimension, using one member from a set of built-in object properties: spss.Dimension.Place.row for a row dimension, spss.Dimension.Place.column for a column dimension, and spss.Dimension.Place.layer for a layer dimension.
- The second argument to Append is a string that specifies the name used to label this dimension in the displayed table.
- Although not required to append a dimension, it's good practice to store a reference to the newly created dimension object in a variable. For instance, the variable *rowdim1* holds a reference to the object for the row dimension named *rowdim-1*. Depending on which approach you use for setting categories, you may need this object reference.

Figure 3. Resulting table structure

The order in which the dimensions are appended determines how they are displayed in the table. Each newly appended dimension of a particular type (row, column, or layer) becomes the current innermost dimension in the displayed table. In the example above, *rowdim-2* is the innermost row dimension since it is the last one to be appended. Had *rowdim-2* been appended first, followed by *rowdim-1*, *rowdim-1* would be the innermost dimension.

Note: Generation of the resulting table requires more code than is shown here.

Example: Using the Insert Method

```
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")
rowdim1=table.Append(spss.Dimension.Place.row,"rowdim-1")
rowdim2=table.Append(spss.Dimension.Place.row,"rowdim-2")
rowdim3=table.Insert(2,spss.Dimension.Place.row,"rowdim-3")
coldim=table.Append(spss.Dimension.Place.column,"coldim")
```

- The first argument to the `Insert` method specifies the position within the dimensions of that type (row, column, or layer). The first position has index 1 (unlike typical Python indexing that starts with 0) and defines the innermost dimension of that type in the displayed table. Successive integers specify the next innermost dimension and so on. In the current example, *rowdim-3* is inserted at position 2 and *rowdim-1* is moved from position 2 to position 3.
- The second argument to `Insert` specifies the type of dimension, using one member from a set of built-in object properties: `spss.Dimension.Place.row` for a row dimension, `spss.Dimension.Place.column` for a column dimension, and `spss.Dimension.Place.layer` for a layer dimension.
- The third argument to `Insert` is a string that specifies the name used to label this dimension in the displayed table.
- Although not required to insert a dimension, it is good practice to store a reference to the newly created dimension object to a variable. For instance, the variable *rowdim3* holds a reference to the object for the row dimension named *rowdim-3*. Depending on which approach you use for setting categories, you may need this object reference.

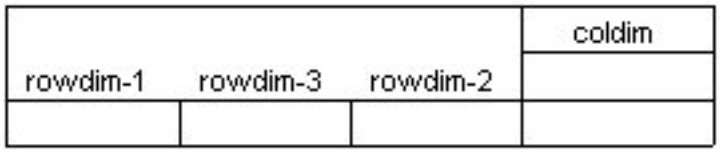


Figure 4. Resulting table structure

Note: Generation of the resulting table requires more code than is shown here.

Step 2: Defining Categories: There are two ways to define categories for each dimension: explicitly, using the SetCategories method, or implicitly when setting values. The explicit method is shown here. The implicit method is shown in "Step 3: Setting Cell Values" on page 21.

Example

```
from spss import CellText
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")

coldim=table.Append(spss.Dimension.Place.column,"coldim")
rowdim1=table.Append(spss.Dimension.Place.row,"rowdim-1")
rowdim2=table.Append(spss.Dimension.Place.row,"rowdim-2")

cat1=CellText.String("A1")
cat2=CellText.String("B1")
cat3=CellText.String("A2")
cat4=CellText.String("B2")
cat5=CellText.String("C")
cat6=CellText.String("D")
cat7=CellText.String("E")

table.SetCategories(rowdim1,[cat1,cat2])
table.SetCategories(rowdim2,[cat3,cat4])
table.SetCategories(coldim,[cat5,cat6,cat7])
```

- The statement `from spss import CellText` allows you to omit the `spss` prefix when specifying `CellText` objects (discussed below), once you have imported the `spss` module.
- You set categories after you add dimensions, so the `SetCategories` method calls follow the `Append` or `Insert` method calls.

- The first argument to `SetCategories` is an object reference to the dimension for which the categories are being defined. This underscores the need to save references to the dimensions you create with `Append` or `Insert`, as discussed in the previous topic.
- The second argument to `SetCategories` is a single category or a sequence of unique category values, each expressed as a CellText object (one of `CellText.Number`, `CellText.String`, `CellText.VarName`, or `CellText.VarValue`). When you specify a category as a variable name or variable value, pivot table display options such as display variable labels or display value labels are honored. In the present example, we use string objects whose single argument is the string specifying the category.
- It is a good practice to assign variables to the `CellText` objects representing the category names, since each category will often need to be referenced more than once when setting cell values.

		coldim		
rowdim-1	rowdim-2	C	D	E
A1	A2			
	B2			
B1	A2			
	B2			

Figure 5. Resulting table structure

Note: Generation of the resulting table requires more code than is shown here.

Step 3: Setting Cell Values: There are two primary methods for setting cell values: setting values one cell at a time by specifying the categories that define the cell, or using the SetCellsByRow or SetCellsByColumn method.

Example: Specifying Cells by Their Category Values

This example reproduces the table created in the SimplePivotTable example.

```
from spss import CellText
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")

table.Append(spss.Dimension.Place.row,"row dimension")
table.Append(spss.Dimension.Place.column,"column dimension")

row_cat1 = CellText.String("first row")
row_cat2 = CellText.String("second row")
col_cat1 = CellText.String("first column")
col_cat2 = CellText.String("second column")

table[(row_cat1,col_cat1)] = CellText.Number(11)
table[(row_cat1,col_cat2)] = CellText.Number(12)
table[(row_cat2,col_cat1)] = CellText.Number(21)
table[(row_cat2,col_cat2)] = CellText.Number(22)
```

- The Append method is used to add a row dimension and then a column dimension to the structure of the table. The table specified in this example has one row dimension and one column dimension. Notice that references to the dimension objects created by the `Append` method are not saved to variables, contrary to the recommendations in the topic on adding dimensions. When setting cells using the current approach, these object references are not needed.
- For convenience, variables consisting of `CellText` objects are created for each of the categories in the two dimensions.
- Cells are specified by their category values in each dimension. In the tuple (or list) that specifies the category values--for example, `(row_cat1,col_cat1)`--the first element corresponds to the first appended dimension (what we have named "row dimension") and the second element to the second appended

dimension (what we have named "column dimension"). The tuple (row_cat1,col_cat1) then specifies the cell whose "row dimension" category is "first row" and "column dimension" category is "first column."

- You may notice that the example does not make use of the SetCategories method to define the row and column dimension category values. When you assign cell values in the manner done here--table[(category1,category2)]--the values provided to specify the categories for a given cell are used by the BasePivotTable object to build the set of categories for the table. Values provided in the first element of the tuple (or list) become the categories in the dimension created by the first method call to Append or Insert. Values in the second element become the categories in the dimension created by the second method call to Append or Insert, and so on. Within a given dimension, the specified category values must be unique. The order of the categories, as displayed in the table, is the order in which they are created from table[(category1,category2)]. In the example shown above, the row categories will be displayed in the order *"first row," "second row."*
- Cell values must be specified as CellText objects (one of CellText.Number, CellText.String, CellText.VarName, or CellText.VarValue).
- In this example, Number objects are used to specify numeric values for the cells. Values will be formatted using the table's default format. Instances of the BasePivotTable class have an implicit default format of GeneralStat. You can change the default format using the SetDefaultFormatSpec method, or you can override the default by explicitly specifying the format, as in: CellText.Number(22,spss.FormatSpec.Correlation). See the topic "Number Class" on page 33 for more information.

Example: Setting Cell Values by Row or Column

The SetCellsByRow and SetCellsByColumn methods allow you to set cell values for entire rows or columns with one method call. To illustrate the approach, we will use the SetCellsByRow method to reproduce the table created in the SimplePivotTable example. It is a simple matter to rewrite the example to set cells by column.

Note: You can only use the SetCellsByRow method with pivot tables that have one column dimension and you can only use the SetCellsByColumn method with pivot tables that have one row dimension.

```
from spss import CellText
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")

rowdim = table.Append(spss.Dimension.Place.row,"row dimension")
coldim = table.Append(spss.Dimension.Place.column,"column dimension")

row_cat1 = CellText.String("first row")
row_cat2 = CellText.String("second row")
col_cat1 = CellText.String("first column")
col_cat2 = CellText.String("second column")

table.SetCategories(rowdim,[row_cat1,row_cat2])
table.SetCategories(coldim,[col_cat1,col_cat2])

table.SetCellsByRow(row_cat1,[CellText.Number(11),
                              CellText.Number(12)])
table.SetCellsByRow(row_cat2,[CellText.Number(21),
                              CellText.Number(22)])
```

- The SetCellsByRow method is called for each of the two categories in the row dimension.
- The first argument to the SetCellsByRow method is the row category for which values are to be set. The argument must be specified as a CellText object (one of CellText.Number, CellText.String, CellText.VarName, or CellText.VarValue). When setting row values for a pivot table with multiple row dimensions, you specify a list of category values for the first argument to SetCellsByRow, where each element in the list is a category value for a different row dimension.
- The second argument to the SetCellsByRow method is a list or tuple of CellText objects (one of CellText.Number, CellText.String, CellText.VarName, or CellText.VarValue) that specify the elements

of the row, one element for each column category in the single column dimension. The first element in the list or tuple will populate the first column category (in this case, *col_cat1*), the second will populate the second column category, and so on.

- In this example, Number objects are used to specify numeric values for the cells. Values will be formatted using the table's default format. Instances of the `BasePivotTable` class have an implicit default format of `GeneralStat`. You can change the default format using the SetDefaultFormatSpec method, or you can override the default by explicitly specifying the format, as in: `CellText.Number(22,spss.FormatSpec.Correlation)`. See the topic "Number Class" on page 33 for more information.

Using Cell Values in Expressions: Once a cell's value has been set, it can be accessed and used to specify the value for another cell. Cell values are stored as `CellText.Number` or `CellText.String` objects. To use a cell value in an expression, you obtain a string or numeric representation of the value using the toString or toNumber method.

Example: Numeric Representations of Cell Values

```
from spss import CellText
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")

table.Append(spss.Dimension.Place.row,"row dimension")
table.Append(spss.Dimension.Place.column,"column dimension")

row_cat1 = CellText.String("first row")
row_cat2 = CellText.String("second row")
col_cat1 = CellText.String("first column")
col_cat2 = CellText.String("second column")

table[(row_cat1,col_cat1)] = CellText.Number(11)
cellValue = table[(row_cat1,col_cat1)].toNumber()
table[(row_cat2,col_cat2)] = CellText.Number(2*cellValue)
```

- The `toNumber` method is used to obtain a numeric representation of the cell with category values `("first row","first column")`. The numeric value is stored in the variable *cellValue* and used to specify the value of another cell.
- Character representations of numeric values stored as `CellText.String` objects, such as `CellText.String("11")`, are converted to a numeric value by the `toNumber` method.

Example: String Representations of Cell Values

```
from spss import CellText
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")

table.Append(spss.Dimension.Place.row,"row dimension")
table.Append(spss.Dimension.Place.column,"column dimension")

row_cat1 = CellText.String("first row")
row_cat2 = CellText.String("second row")
col_cat1 = CellText.String("first column")
col_cat2 = CellText.String("second column")

table[(row_cat1,col_cat1)] = CellText.String("abc")
cellValue = table[(row_cat1,col_cat1)].toString()
table[(row_cat2,col_cat2)] = CellText.String(cellValue + "d")
```

- The `toString` method is used to obtain a string representation of the cell with category values `("first row","first column")`. The string value is stored in the variable *cellValue* and used to specify the value of another cell.
- Numeric values stored as `CellText.Number` objects are converted to a string value by the `toString` method.

spss.BasePivotTable Methods

The `BasePivotTable` class has methods that allow you to build complex pivot tables. If you only need to create a pivot table with a single row and a single column dimension then consider using the `SimplePivotTable` method.

Append Method: .Append(place,dimName,hideName, hideLabels). *Appends row, column, and layer dimensions to a pivot table.* You use this method, or the Insert method, to create the dimensions associated with a custom pivot table. The argument *place* specifies the type of dimension: `spss.Dimension.Place.row` for a row dimension, `spss.Dimension.Place.column` for a column dimension, and `spss.Dimension.Place.layer` for a layer dimension. The argument *dimName* is a string that specifies the name used to label this dimension in the displayed table. Each dimension must have a unique name. The argument *hideName* specifies whether the dimension name is hidden--by default, it is displayed. Use `hideName=True` to hide the name. The argument *hideLabels* specifies whether category labels for this dimension are hidden--by default, they are displayed. Use `hideLabels=True` to hide category labels.

- The order in which dimensions are appended affects how they are displayed in the resulting table. Each newly appended dimension of a particular type (row, column, or layer) becomes the current innermost dimension in the displayed table, as shown in the example below.
- The order in which dimensions are created (with the `Append` or `Insert` method) determines the order in which categories should be specified when providing the dimension coordinates for a particular cell (used when Setting Cell Values or adding Footnotes). For example, when specifying coordinates using an expression such as `(category1,category2)`, *category1* refers to the dimension created by the first call to `Append` or `Insert`, and *category2* refers to the dimension created by the second call to `Append` or `Insert`.

Example

```
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")
coldim=table.Append(spss.Dimension.Place.column,"coldim")
rowdim1=table.Append(spss.Dimension.Place.row,"rowdim-1")
rowdim2=table.Append(spss.Dimension.Place.row,"rowdim-2")
```

Figure 6. Resulting table structure

Examples of using the `Append` method are most easily understood in the context of going through the steps to create a pivot table. See the topic "General Approach to Creating Pivot Tables" on page 19 for more information.

Caption Method: .Caption(caption). *Adds a caption to the pivot table.* The argument *caption* is a string specifying the caption.

Example

```
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")
table.Caption("A sample caption")
```

CategoryFootnotes Method: .CategoryFootnotes(dimPlace,dimName,category,footnote). *Used to add a footnote to a specified category.*

- The argument *dimPlace* specifies the dimension type associated with the category, using one member from a set of built-in object properties: `spss.Dimension.Place.row` for a row dimension, `spss.Dimension.Place.column` for a column dimension, and `spss.Dimension.Place.layer` for a layer dimension.
- The argument *dimName* is the string that specifies the dimension name associated with the category. This is the name specified when the dimension was created by the `Append` or `Insert` method.
- The argument *category* specifies the category and must be a CellText object (one of `CellText.Number`, `CellText.String`, `CellText.VarName`, or `CellText.VarValue`).

- The argument *footnote* is a string specifying the footnote.

Example

```
from spss import CellText
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")

table.Append(spss.Dimension.Place.row,"row dimension")
table.Append(spss.Dimension.Place.column,"column dimension")

row_cat1 = CellText.String("first row")
row_cat2 = CellText.String("second row")
col_cat1 = CellText.String("first column")
col_cat2 = CellText.String("second column")

table.CategoryFootnotes(spss.Dimension.Place.row,"row dimension",
                        row_cat1,"A category footnote")
```

DimensionFootnotes Method: .DimensionFootnotes(dimPlace,dimName,footnote). *Used to add a footnote to a dimension.*

- The argument *dimPlace* specifies the type of dimension, using one member from a set of built-in object properties: `spss.Dimension.Place.row` for a row dimension, `spss.Dimension.Place.column` for a column dimension, and `spss.Dimension.Place.layer` for a layer dimension.
- The argument *dimName* is the string that specifies the name given to this dimension when it was created by the `Append` or `Insert` method.
- The argument *footnote* is a string specifying the footnote.

Example

```
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")

table.Append(spss.Dimension.Place.row,"row dimension")
table.Append(spss.Dimension.Place.column,"column dimension")
table.DimensionFootnotes(spss.Dimension.Place.column,
                         "column dimension","A dimension footnote")
```

Footnotes Method: .Footnotes(categories,footnote). *Used to add a footnote to a table cell.* The argument *categories* is a list or tuple of categories specifying the cell for which a footnote is to be added. Each element in the list or tuple must be a CellText object (one of `CellText.Number`, `CellText.String`, `CellText.VarName`, or `CellText.VarValue`). The argument *footnote* is a string specifying the footnote.

Example

```
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")

rowdim=table.Append(spss.Dimension.Place.row,"rowdim")
coldim=table.Append(spss.Dimension.Place.column,"coldim")

table.SetCategories(rowdim,spss.CellText.String("row1"))
table.SetCategories(coldim,spss.CellText.String("column1"))

table[(spss.CellText.String("row1"),spss.CellText.String("column1"))] = \
       spss.CellText.String("cell value")
table.Footnotes((spss.CellText.String("row1"),
                 spss.CellText.String("column1")),
     "Footnote for the cell specified by the categories row1 and column1")
```

- The order in which dimensions are added to the table, either through a call to `Append` or to `Insert`, determines the order in which categories should be specified when providing the dimension coordinates for a particular cell. In the present example, the dimension *rowdim* is added first and *coldim* second, so the first element of `(spss.CellText.String("row1"),spss.CellText.String("column1"))` specifies a category of *rowdim* and the second element specifies a category of *coldim*.

GetDefaultFormatSpec Method: .GetDefaultFormatSpec(). *Returns the default format for CellText.Number objects.* The returned value is a list with two elements. The first element is the integer code associated with the format. Codes and associated formats are listed in Table 1 on page 34. For formats with codes `5` (`Mean`), `12` (`Variable`), `13` (`StdDev`), `14` (`Difference`), and `15` (`Sum`), the second element of the returned value

is the index of the variable in the active dataset whose format is used to determine details of the resulting format. For all other formats, the second element is the Python data type *None*. You can set the default format with the SetDefaultFormatSpec method.

- Instances of the `BasePivotTable` class have an implicit default format of `GeneralStat`.

Example

```
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")
print "Default format: ", table.GetDefaultFormatSpec()
```

HideTitle Method: .HideTitle(). *Used to hide the title of a pivot table.* By default, the title is shown.

Example

```
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")
table.HideTitle()
```

Insert Method: .Insert(i,place,dimName,hideName, hideLabels). *Inserts row, column, and layer dimensions into a pivot table.* You use this method, or the Append method, to create the dimensions associated with a custom pivot table. The argument *i* specifies the position within the dimensions of that type (row, column, or layer). The first position has index 1 and defines the innermost dimension of that type in the displayed table. Successive integers specify the next innermost dimension and so on. The argument *place* specifies the type of dimension: `spss.Dimension.Place.row` for a row dimension, `spss.Dimension.Place.column` for a column dimension, and `spss.Dimension.Place.layer` for a layer dimension. The argument *dimName* is a string that specifies the name used to label this dimension in the displayed table. Each dimension must have a unique name. The argument *hideName* specifies whether the dimension name is hidden--by default, it is displayed. Use `hideName=True` to hide the name. The argument *hideLabels* specifies whether category labels for this dimension are hidden--by default, they are displayed. Use `hideLabels=True` to hide category labels.

- The argument *i* can take on the values 1, 2, ... , *n*+1 where *n* is the position of the outermost dimension (of the type specified by *place*) created by any previous calls to `Append` or `Insert`. For example, after appending two row dimensions, you can insert a row dimension at positions 1, 2, or 3. You cannot, however, insert a row dimension at position 3 if only one row dimension has been created.
- The order in which dimensions are created (with the `Append` or `Insert` method) determines the order in which categories should be specified when providing the dimension coordinates for a particular cell (used when Setting Cell Values or adding Footnotes). For example, when specifying coordinates using an expression such as `(category1,category2)`, *category1* refers to the dimension created by the first call to `Append` or `Insert`, and *category2* refers to the dimension created by the second call to `Append` or `Insert`.

  Note: The order in which categories should be specified is not determined by dimension positions as specified by the argument *i*.

Example

```
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")
rowdim1=table.Append(spss.Dimension.Place.row,"rowdim-1")
rowdim2=table.Append(spss.Dimension.Place.row,"rowdim-2")
rowdim3=table.Insert(2,spss.Dimension.Place.row,"rowdim-3")
coldim=table.Append(spss.Dimension.Place.column,"coldim")
```

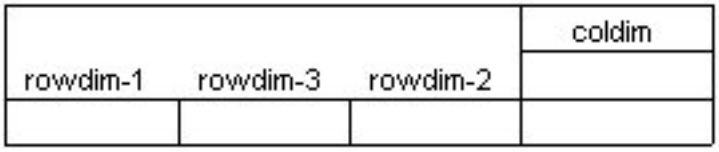


Figure 7. Resulting table structure

Examples of using the `Insert` method are most easily understood in the context of going through the steps to create a pivot table. See the topic "General Approach to Creating Pivot Tables" on page 19 for more information.

SetCategories Method: .SetCategories(dim,categories). *Sets categories for the specified dimension.* The argument *dim* is a reference to the dimension object for which categories are to be set. Dimensions are created with the Append or Insert method. The argument *categories* is a single value or a sequence of unique values, each of which is a CellText object (one of `CellText.Number`, `CellText.String`, `CellText.VarName`, or `CellText.VarValue`).

- In addition to defining category values for a specified dimension, `SetCategories` sets the pivot table object's value of the currently selected category for the specified dimension. In other words, calling `SetCategories` also sets a pointer to a category in the pivot table. When a sequence of values is provided, the currently selected category (for the specified dimension) is the last value in the sequence. For an example of using currently selected dimension categories to specify a cell, see the SetCell method.
- Once a category has been defined, a subsequent call to `SetCategories` (for that category) will set that category as the currently selected one for the specified dimension.

Example

```
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")

rowdim=table.Append(spss.Dimension.Place.row,"rowdim")
coldim=table.Append(spss.Dimension.Place.column,"coldim")

table.SetCategories(rowdim,[spss.CellText.String("row1"),
                            spss.CellText.String("row2")])
table.SetCategories(coldim,[spss.CellText.String("column1"),
                            spss.CellText.String("column2")])
```

Examples of using the `SetCategories` method are most easily understood in the context of going through the steps to create a pivot table. See the topic "General Approach to Creating Pivot Tables" on page 19 for more information.

SetCell Method: .SetCell(cell). *Sets the value for the cell specified by the currently selected set of category values.* The argument *cell* is the value, specified as a CellText object (one of `CellText.Number`, `CellText.String`, `CellText.VarName`, or `CellText.VarValue`). Category values are selected using the SetCategories method as shown in the following example.

Example

```
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")

rowdim = table.Append(spss.Dimension.Place.row,"rowdim")
coldim = table.Append(spss.Dimension.Place.column,"coldim")

# Define category values and set the currently selected set of
# category values to "row1" for rowdim and "column1" for coldim.
table.SetCategories(rowdim,spss.CellText.String("row1"))
table.SetCategories(coldim,spss.CellText.String("column1"))

# Set the value for the current cell specified by the currently
# selected set of category values.
table.SetCell(spss.CellText.Number(11))

table.SetCategories(rowdim,spss.CellText.String("row2"))
```

```
table.SetCategories(coldim,spss.CellText.String("column2"))

# Set the value for the current cell. Its category values are "row2"
# for rowdim and "column2" for coldim.
table.SetCell(spss.CellText.Number(22))

# Set the currently selected category to "row1" for rowdim.
table.SetCategories(rowdim,spss.CellText.String("row1"))

# Set the value for the current cell. Its category values are "row1"
# for rowdim and "column2" for coldim.

table.SetCell(spss.CellText.Number(12))
```

- In this example, Number objects are used to specify numeric values for the cells. Values will be formatted using the table's default format. Instances of the `BasePivotTable` class have an implicit default format of `GeneralStat`. You can change the default format using the SetDefaultFormatSpec method, or you can override the default by explicitly specifying the format, as in: `CellText.Number(22,spss.FormatSpec.Correlation)`. See the topic "Number Class" on page 33 for more information.

	coldim	
rowdim	column1	column2
row1	11	12
row2		22

Figure 8. Resulting table

SetCellsByColumn Method: .SetCellsByColumn(collabels,cells). *Sets cell values for the column specified by a set of categories, one for each column dimension.* The argument *collabels* specifies the set of categories that defines the column--a single value, or a list or tuple. The argument *cells* is a tuple or list of cell values. Column categories and cell values must be specified as CellText objects (one of `CellText.Number`, `CellText.String`, `CellText.VarName`, or `CellText.VarValue`).

- For tables with multiple column dimensions, the order of categories in the *collabels* argument is the order in which their respective dimensions were added (appended or inserted) to the table. For example, given two column dimensions *coldim1* and *coldim2* added in the order *coldim1* and *coldim2*, the first element in *collabels* should be the category for *coldim1* and the second the category for *coldim2*.
- You can only use the `SetCellsByColumn` method with pivot tables that have one row dimension.

Example

```
from spss import CellText
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")
rowdim=table.Append(spss.Dimension.Place.row,"rowdim")
coldim1=table.Append(spss.Dimension.Place.column,"coldim-1")
coldim2=table.Append(spss.Dimension.Place.column,"coldim-2")

cat1=CellText.String("coldim1:A")
cat2=CellText.String("coldim1:B")
cat3=CellText.String("coldim2:A")
cat4=CellText.String("coldim2:B")
cat5=CellText.String("C")
cat6=CellText.String("D")

table.SetCategories(coldim1,[cat1,cat2])
table.SetCategories(coldim2,[cat3,cat4])
table.SetCategories(rowdim,[cat5,cat6])

table.SetCellsByColumn((cat1,cat3),
                    [CellText.Number(11),
                     CellText.Number(21)])
table.SetCellsByColumn((cat1,cat4),
                    [CellText.Number(12),
                     CellText.Number(22)])
table.SetCellsByColumn((cat2,cat3),
                    [CellText.Number(13),
```

```
                    CellText.Number(23)])
table.SetCellsByColumn((cat2,cat4),
                   [CellText.Number(14),
                    CellText.Number(24)])
```

- In this example, Number objects are used to specify numeric values for the cells. Values will be formatted using the table's default format. Instances of the `BasePivotTable` class have an implicit default format of `GeneralStat`. You can change the default format using the SetDefaultFormatSpec method, or you can override the default by explicitly specifying the format, as in: `CellText.Number(22,spss.FormatSpec.Correlation)`. See the topic "Number Class" on page 33 for more information.

	coldim-1			
	coldim1:A		coldim1:B	
	coldim-2		coldim-2	
rowdim	coldim2:A	coldim2:B	coldim2:A	coldim2:B
C	11	12	13	14
D	21	22	23	24

Figure 9. Resulting table structure

Examples of using the `SetCellsByColumn` method are most easily understood in the context of going through the steps to create a pivot table. See the topic "General Approach to Creating Pivot Tables" on page 19 for more information.

SetCellsByRow Method: .SetCellsByRow(rowlabels,cells). *Sets cell values for the row specified by a set of categories, one for each row dimension.* The argument *rowlabels* specifies the set of categories that defines the row--a single value, or a list or tuple. The argument *cells* is a tuple or list of cell values. Row categories and cell values must be specified as CellText objects (one of `CellText.Number`, `CellText.String`, `CellText.VarName`, or `CellText.VarValue`).

- For tables with multiple row dimensions, the order of categories in the *rowlabels* argument is the order in which their respective dimensions were added (appended or inserted) to the table. For example, given two row dimensions *rowdim1* and *rowdim2* added in the order *rowdim1* and *rowdim2*, the first element in *rowlabels* should be the category for *rowdim1* and the second the category for *rowdim2*.
- You can only use the `SetCellsByRow` method with pivot tables that have one column dimension.

Example

```
from spss import CellText

table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")

coldim=table.Append(spss.Dimension.Place.column,"coldim")
rowdim1=table.Append(spss.Dimension.Place.row,"rowdim-1")
rowdim2=table.Append(spss.Dimension.Place.row,"rowdim-2")

cat1=CellText.String("rowdim1:A")
cat2=CellText.String("rowdim1:B")
cat3=CellText.String("rowdim2:A")
cat4=CellText.String("rowdim2:B")
cat5=CellText.String("C")
cat6=CellText.String("D")

table.SetCategories(rowdim1,[cat1,cat2])
table.SetCategories(rowdim2,[cat3,cat4])
table.SetCategories(coldim,[cat5,cat6])

table.SetCellsByRow((cat1,cat3),
                    [CellText.Number(11),
                     CellText.Number(12)])
table.SetCellsByRow((cat1,cat4),
                    [CellText.Number(21),
```

```
                    CellText.Number(22)])
table.SetCellsByRow((cat2,cat3),
                    [CellText.Number(31),
                     CellText.Number(32)])
table.SetCellsByRow((cat2,cat4),
                    [CellText.Number(41),
                     CellText.Number(42)])
```

- In this example, Number objects are used to specify numeric values for the cells. Values will be formatted using the table's default format. Instances of the `BasePivotTable` class have an implicit default format of `GeneralStat`. You can change the default format using the SetDefaultFormatSpec method, or you can override the default by explicitly specifying the format, as in: `CellText.Number(22,spss.FormatSpec.Correlation)`. See the topic "Number Class" on page 33 for more information.

		coldim	
rowdim-1	rowdim-2	C	D
rowdim1:A	rowdim2:A	11	12
	rowdim2:B	21	22
rowdim1:B	rowdim2:A	31	32
	rowdim2:B	41	42

Figure 10. Resulting table

Examples of using the `SetCellsByRow` method are most easily understood in the context of going through the steps to create a pivot table. See the topic "General Approach to Creating Pivot Tables" on page 19 for more information.

SetDefaultFormatSpec Method: .SetDefaultFormatSpec(formatSpec,varIndex). *Sets the default format for CellText.Number objects.* The argument *formatspec* is of the form `spss.FormatSpec.format` where `format` is one of those listed in Table 1 on page 34--for example, `spss.FormatSpec.Mean`. The argument *varIndex* is the index of a variable in the active dataset whose format is used to determine details of the resulting format. *varIndex* is only used for, and required by, the following subset of formats: `Mean`, `Variable`, `StdDev`, `Difference`, and `Sum`. Index values represent position in the active dataset, starting with 0 for the first variable in file order. The default format can be retrieved with the GetDefaultFormatSpec method.

- Instances of the `BasePivotTable` class have an implicit default format of `GeneralStat`.

Example

```
from spss import CellText
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")
table.SetDefaultFormatSpec(spss.FormatSpec.Mean,2)
table.Append(spss.Dimension.Place.row,"rowdim")
table.Append(spss.Dimension.Place.column,"coldim")

table[(CellText.String("row1"),CellText.String("col1"))] = \
       CellText.Number(2.37)
table[(CellText.String("row2"),CellText.String("col1"))] = \
       CellText.Number(4.34)
```

- The call to `SetDefaultFormatSpec` specifies that the format for mean values is to be used as the default, and that it will be based on the format for the variable with index value 2 in the active dataset. Subsequent instances of `CellText.Number` will use this default, so the cell values `2.37` and `4.34` will be formatted as mean values.

SimplePivotTable Method: .SimplePivotTable(rowdim,rowlabels,coldim,collabels,cells). *Creates a pivot table with one row dimension and one column dimension.*

- **rowdim.** An optional label for the row dimension, given as a string. If empty, the row dimension label is hidden. If specified, it must be distinct from the value, if any, of the *coldim* argument.

- **rowlabels.** An optional list of items to label the rows. Each item must be unique and can be a character string, a numeric value, or a CellText object (one of `CellText.Number`, `CellText.String`, `CellText.VarName`, or `CellText.VarValue`). If provided, the length of this list determines the number of rows in the table. If omitted, the number of rows is equal to the number of elements in the argument *cells*.
- **coldim.** An optional label for the column dimension, given as a string. If empty, the column dimension label is hidden. If specified, it must be distinct from the value, if any, of the *rowdim* argument.
- **collabels.** An optional list of items to label the columns. Each item must be unique and can be a character string, a numeric value, or a CellText object (one of `CellText.Number`, `CellText.String`, `CellText.VarName`, or `CellText.VarValue`). If provided, the length of this list determines the number of columns in the table. If omitted, the number of columns is equal to the length of the first element of *cells*. If *cells* is one-dimensional, this implies a table with one column and as many rows as there are elements in *cells*. See the examples below for the case where *cells* is two-dimensional and *collabels* is omitted.
- **cells.** This argument specifies the values for the cells of the pivot table. It consists of a one- or two-dimensional sequence of items that can be indexed as `cells[i]` or `cells[i][j]`. For example, `[1,2,3,4]` is a one-dimensional sequence, and `[[1,2],[3,4]]` is a two-dimensional sequence. Each element in *cells* can be a character string, a numeric value, a CellText object (one of `CellText.Number`, `CellText.String`, `CellText.VarName`, or `CellText.VarValue`), a Python `time.struct_time` object, or a Python `datetime.datetime` object. Examples showing how the rows and columns of the pivot table are populated from *cells* are provided below.
- The number of elements in *cells* must equal the product of the number of rows and the number of columns.
- Elements in the pivot table are populated in row-wise fashion from the elements of *cells*. For example, if you specify a table with two rows and two columns and provide `cells=[1,2,3,4]`, the first row will consist of the first two elements and the second row will consist of the last two elements.
- Numeric values specified in *cells*, *rowlabels*, or *collabels* will be converted to CellText.Number objects with a format given by the default. The default format can be set with the SetDefaultFormatSpec method and retrieved with the GetDefaultFormatSpec method. Instances of the `BasePivotTable` class have an implicit default format of `GeneralStat`.
- String values specified in *cells*, *rowlabels*, or *collabels* will be converted to CellText.String objects.
- When specifying cell values with Python `time.struct_time` or `datetime.datetime` objects, the value will be displayed in seconds--specifically, the number of seconds from October 14, 1582. You can change the format of a cell to a datetime format using the `SetNumericFormatAt` Python Scripting method. This requires embedding Python Scripting code within your Python program. For more information, see the Scripting Guide under Integration Plug-in for Python in the Help system.

Example: Creating a Table with One Column

```
import spss
spss.StartProcedure("mycompany.com.demoProc")

table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")
table.SimplePivotTable(rowdim="row dimension",
                       rowlabels=["row 1","row 2","row 3","row 4"],
                       collabels=["column 1"],
                       cells = [1,2,3,4])
spss.EndProcedure()
```

Result

row dimension	column 1
row 1	1
row 2	2
row 3	3
row 4	4

Figure 11. Pivot table with a single column

Example: Using a Two-Dimensional Sequence for Cells

```
import spss
spss.StartProcedure("mycompany.com.demoProc")

table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")
table.SimplePivotTable(rowdim="row dimension",
                       coldim="column dimension",
                       rowlabels=["row 1","row 2","row 3","row 4"],
                       collabels=["column 1","column 2"],
                       cells = [[1,2],[3,4],[5,6],[7,8]])
spss.EndProcedure()
```

Result

	column dimension	
row dimension	column 1	column 2
row 1	1	2
row 2	3	4
row 3	5	6
row 4	7	8

Figure 12. Table populated from two-dimensional sequence

Example: Using a Two-Dimensional Sequence for Cells and Omitting Column Labels

```
import spss
spss.StartProcedure("mycompany.com.demoProc")

table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")
table.SimplePivotTable(rowdim="row dimension",
                       coldim="column dimension",
                       rowlabels=["row 1","row 2","row 3","row 4"],
                       cells = [[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
spss.EndProcedure()
```

Result

	column dimension		
row dimension	col0	col1	col2
row 1	1	2	3
row 2	4	5	6
row 3	7	8	9
row 4	10	11	12

Figure 13. Table populated from two-dimensional sequence without specifying column labels

TitleFootnotes Method: .TitleFootnotes(footnote). *Used to add a footnote to the table title.* The argument *footnote* is a string specifying the footnote.

Example

```
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")

table.TitleFootnotes("A title footnote")
```

spss.CellText Class

spss.CellText. *A class of objects used to create a dimension category or a cell in a pivot table.* This class is only for use with the `spss.BasePivotTable` class. The `CellText` class is used to create the following object types:

- CellText.Number: Used to specify a numeric value.
- CellText.String: Used to specify a string value.
- CellText.VarName: Used to specify a variable name. Use of this object means that settings for the display of variable names in pivot tables (names, labels, or both) are honored.
- CellText.VarValue: Used to specify a variable value. Use of this object means that settings for the display of variable values in pivot tables (values, labels, or both) are honored.

Number Class: spss.CellText.Number(value,formatspec,varIndex). *Used to specify a numeric value for a category or a cell in a pivot table.* The argument *value* specifies the numeric value. You can also pass a string representation of a numeric value, a Python `time.struct_time` object, or a Python `datetime.datetime` object to this argument. The optional argument *formatspec* is of the form `spss.FormatSpec.format` where `format` is one of those listed in the table below--for example, `spss.FormatSpec.Mean`. You can also specify an integer code for *formatspec*, as in the value `5` for `Mean`. The argument *varIndex* is the index of a variable in the active dataset whose format is used to determine details of the resulting format. *varIndex* is only used in conjunction with *formatspec* and is required when specifying one of the following formats: `Mean`, `Variable`, `StdDev`, `Difference`, and `Sum`. Index values represent position in the active dataset, starting with 0 for the first variable in file order.

- When *formatspec* is omitted, the default format is used. You can set the default format with the SetDefaultFormatSpec method and retrieve the default with the GetDefaultFormatSpec method. Instances of the `BasePivotTable` class have an implicit default format of `GeneralStat`.
- You can obtain a numeric representation of a `CellText.Number` object using the toNumber method, and you can use the toString method to obtain a string representation.
- When specifying cell values with Python `time.struct_time` or `datetime.datetime` objects, the value will be displayed in seconds--specifically, the number of seconds from October 14, 1582. You can change the format of a cell to a datetime format using the `SetNumericFormatAt` Python Scripting method. This requires embedding Python Scripting code within your Python program. For more information, see the Scripting Guide under Integration Plug-in for Python in the Help system.

Example

```
from spss import CellText
from spss import FormatSpec
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")

table.Append(spss.Dimension.Place.row,"rowdim")
table.Append(spss.Dimension.Place.column,"coldim")

table[(CellText.String("row1"),CellText.String("col1"))] = \
       CellText.Number(25.632,FormatSpec.Mean,2)
table[(CellText.String("row2"),CellText.String("col1"))] = \
       CellText.Number(23.785,FormatSpec.Mean,2)
```

In this example, cell values are displayed in the format used for mean values. The format of the variable with index 2 in the active dataset is used to determine the details of the resulting format.

Table 1. Numeric formats for use with FormatSpec.

Format name	**Code**
Coefficient	0
CoefficientSE	1
CoefficientVar	2
Correlation	3
GeneralStat	4
Mean	5
Count	6
Percent	7
PercentNoSign	8
Proportion	9
Significance	10
Residual	11
Variable	12
StdDev	13
Difference	14
Sum	15

Suggestions for Choosing a Format

- Consider using `Coefficient` for unbounded, unstandardized statistics; for instance, beta coefficients in regression.
- `Correlation` is appropriate for statistics bounded by –1 and 1 (typically correlations or measures of association).
- Consider using `GeneralStat` for unbounded, scale-free statistics; for instance, beta coefficients in regression.
- `Mean` is appropriate for the mean of a single variable, or the mean across multiple variables.
- `Count` is appropriate for counts and other integers such as integer degrees of freedom.
- `Percent` and `PercentNoSign` are both appropriate for percentages. `PercentNoSign` results in a value without a percentage symbol (%).
- `Significance` is appropriate for statistics bounded by 0 and 1 (for example, significance levels).
- Consider using `Residual` for residuals from cell counts.
- `Variable` refers to a variable's print format as given in the data dictionary and is appropriate for statistics whose values are taken directly from the observed data (for instance, minimum, maximum, and mode).

- `StdDev` is appropriate for the standard deviation of a single variable, or the standard deviation across multiple variables.
- `Sum` is appropriate for sums of single variables. Results are displayed using the specified variable's print format.

String Class: spss.CellText.String(value). *Used to specify a string value for a category or a cell in a pivot table.* The argument is the string value. You can also pass a numeric value, and it will be converted to a string.

- You can obtain a string representation of a `CellText.String` object using the toString method. For character representations of numeric values stored as `CellText.String` objects, such as `CellText.String("11")`, you can obtain the numeric value using the toNumber method.

Example

```
from spss import CellText
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")

table.Append(spss.Dimension.Place.row,"rowdim")
table.Append(spss.Dimension.Place.column,"coldim")

table[(CellText.String("row1"),CellText.String("col1"))] = \
       CellText.String("1")
table[(CellText.String("row2"),CellText.String("col1"))] = \
       CellText.String("2")
```

VarName Class: spss.CellText.VarName(index). *Used to specify that a category or cell in a pivot table is to be treated as a variable name.* `CellText.VarName` objects honor display settings for variable names in pivot tables (names, labels, or both). The argument is the index value of the variable. Index values represent position in the active dataset, starting with 0 for the first variable in file order.

Example

```
from spss import CellText
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")
coldim=table.Append(spss.Dimension.Place.column,"coldim")
rowdim=table.Append(spss.Dimension.Place.row,"rowdim")
table.SetCategories(rowdim,[CellText.VarName(0),CellText.VarName(1)])
table.SetCategories(coldim,CellText.String("Column Heading"))
```

In this example, row categories are specified as the names of the variables with index values 0 and 1 in the active dataset. Depending on the setting of pivot table labeling for variables in labels, the variable names, labels, or both will be displayed.

VarValue Class: spss.CellText.VarValue(index,value). *Used to specify that a category or cell in a pivot table is to be treated as a variable value.* `CellText.VarValue` objects honor display settings for variable values in pivot tables (values, labels, or both). The argument *index* is the index value of the variable. Index values represent position in the active dataset, starting with 0 for the first variable in file order. The argument *value* is a number (for a numeric variable) or string (for a string variable) representing the value of the `CellText` object.

Example

```
from spss import CellText
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")
coldim=table.Append(spss.Dimension.Place.column,"coldim")
rowdim=table.Append(spss.Dimension.Place.row,"rowdim")
table.SetCategories(rowdim,[CellText.VarValue(0,1),CellText.VarValue(0,2)])
table.SetCategories(coldim,CellText.String("Column Heading"))
```

In this example, row categories are specified as the values 1 and 2 of the variable with index value 0 in the active dataset. Depending on the setting of pivot table labeling for variable values in labels, the values, value labels, or both will be displayed.

toNumber Method: This method is used to obtain a numeric representation of a `CellText.Number` object or a `CellText.String` object that stores a character representation of a numeric value, as in `CellText.String("123")`. Values obtained from this method can be used in arithmetic expressions. You call this method on a `CellText.Number` or `CellText.String` object.

Example

```
from spss import CellText
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")

table.Append(spss.Dimension.Place.row,"row dimension")
table.Append(spss.Dimension.Place.column,"column dimension")

row_cat1 = CellText.String("first row")
row_cat2 = CellText.String("second row")
col_cat1 = CellText.String("first column")
col_cat2 = CellText.String("second column")

table[(row_cat1,col_cat1)] = CellText.Number(11)
cellValue = table[(row_cat1,col_cat1)].toNumber()
table[(row_cat2,col_cat2)] = CellText.Number(2*cellValue)
```

- `table[(row_cat1,col_cat1)].toNumber()` returns a numeric representation of the `CellText` object (recall that table cells are stored as `CellText` objects) for the cell with category values `("first row","first column")`.

toString Method: This method is used to obtain a string representation of a `CellText.String` or `CellText.Number` object. Values obtained from this method can be used in string expressions. You call this method on a `CellText.String` or `CellText.Number` object.

Example

```
from spss import CellText
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")

table.Append(spss.Dimension.Place.row,"row dimension")
table.Append(spss.Dimension.Place.column,"column dimension")

row_cat1 = CellText.String("first row")
row_cat2 = CellText.String("second row")
col_cat1 = CellText.String("first column")
col_cat2 = CellText.String("second column")

table[(row_cat1,col_cat1)] = CellText.String("abc")
cellValue = table[(row_cat1,col_cat1)].toString()
table[(row_cat2,col_cat2)] = CellText.String(cellValue + "d")
```

- `table[(row_cat1,col_cat1)].toString()` returns a string representation of the `CellText` object (recall that table cells are stored as `CellText` objects) for the cell with category values `("first row","first column")`.

Creating a Warnings Table

You can create an IBM SPSS Statistics Warnings table using the `BasePivotTable` class by specifying `"Warnings"` for the *templateName* argument. Note that an IBM SPSS Statistics Warnings table has a very specific structure, so unless you actually want a Warnings table you should avoid using `"Warnings"` for *templateName*.

Example

```
import spss
spss.StartProcedure("demo")
table = spss.BasePivotTable("Warnings ","Warnings")
table.Append(spss.Dimension.Place.row,"rowdim",hideLabels=True)
rowLabel = spss.CellText.String("1")
table[(rowLabel,)] = spss.CellText.String("""First line of Warnings table content
Second line of Warnings table content""")
```

- The *title* argument is set to the string "Warnings ", It can be set to an arbitrary value but it cannot be identical to the *templateName* value, hence the space at the end of the string.

- The *templateName* argument must be set to the string "Warnings", independent of the IBM SPSS Statistics output language.
- A Warnings table has a single row dimension with all labels hidden and can consist of one or more rows. In this example, the table has a single multi-line row, formatted with a Python triple-quoted string.

Result

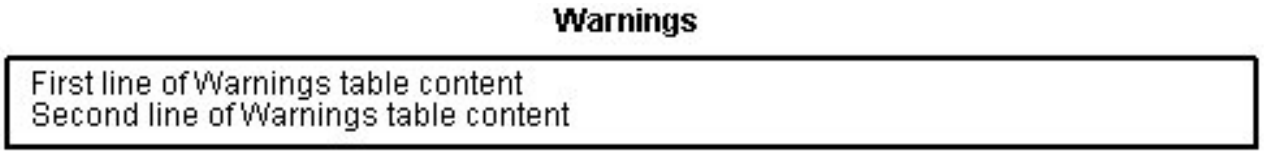


Figure 14. Warnings table

spss.BaseProcedure Class

The `spss.BaseProcedure` class is used to create classes that encapsulate procedures. Procedures can read the data, perform computations, add new variables and/or new cases to the active dataset, and produce pivot table output and text blocks in the IBM SPSS Statistics Viewer. Procedures have almost the same capabilities as built-in IBM SPSS Statistics procedures, such as `DESCRIPTIVES` and `REGRESSION`, but they are written in Python by users. Use of the `spss.BaseProcedure` class provides an alternative to encapsulating procedure code within a Python function and explicitly using an `spss.StartProcedure-spss.EndProcedure` block for the procedure output. All classes that encapsulate procedures must inherit from the `BaseProcedure` class.

The `spss.BaseProcedure` class has three methods: `__init__`, `execProcedure`, and `execUserProcedure`. When creating procedure classes you always override the `execUserProcedure` method, replacing it with the body of your procedure. You override the `__init__` method if you need to provide arguments other than the procedure name and the optional OMS identifier. You never override the `execProcedure` method. It is responsible for calling `execUserProcedure` to run your procedure as well as automatically making the necessary calls to `spss.StartProcedure` and `spss.EndProcedure`.

The rules governing procedure code contained within the `execUserProcedure` method are the same as those for `StartProcedure-EndProcedure` blocks. See the topic “spss.StartProcedure Function” on page 85 for more information.

Example

As an example, we will create a procedure class that calculates group means for a selected variable using a specified categorical variable to define the groups. The output of the procedure is a pivot table displaying the group means. For an alternative approach to creating the same procedure, but making explicit use of `spss.StartProcedure-spss.EndProcedure` and without the use of the `BaseProcedure` class, see the example for the spss.StartProcedure function.

```
class groupMeans(spss.BaseProcedure):

    #Overrides __init__ method to pass arguments
    def __init__(self, procName, groupVar, sumVar):
        self.procName = procName
        self.omsIdentifier = ""
        self.groupVar = groupVar
        self.sumVar = sumVar

    #Overrides execUserProcedure method of BaseProcedure
    def execUserProcedure(self):
        #Determine variable indexes from variable names
        varCount = spss.GetVariableCount()
        groupIndex = 0
        sumIndex = 0
        for i in range(varCount):
```

```
            varName = spss.GetVariableName(i)
            if varName == self.groupVar:
                groupIndex = i
                continue
            elif varName == self.sumVar:
                sumIndex = i
                continue

        varIndex = [groupIndex,sumIndex]
        cur = spss.Cursor(varIndex)
        Counts={};Totals={}

        #Calculate group sums
        for i in range(cur.GetCaseCount()):
            row = cur.fetchone()
            cat=int(row[0])
            Counts[cat]=Counts.get(cat,0) + 1
            Totals[cat]=Totals.get(cat,0) + row[1]

        cur.close()

        #Create a pivot table
        table = spss.BasePivotTable("Group Means",
                                    "OMS table subtype")
        table.Append(spss.Dimension.Place.row,
                     spss.GetVariableLabel(groupIndex))
        table.Append(spss.Dimension.Place.column,
                     spss.GetVariableLabel(sumIndex))

        category2 = spss.CellText.String("Mean")
        for cat in sorted(Counts):
            category1 = spss.CellText.Number(cat)
            table[(category1,category2)] = \
                   spss.CellText.Number(Totals[cat]/Counts[cat])
```

- `groupMeans` is a class based on the `spss.BaseProcedure` class.
- The procedure defined by the class requires two arguments, the name of the grouping variable (*groupVar*) and the name of the variable for which group means are desired (*sumVar*). Passing these values requires overriding the `__init__` method of `spss.BaseProcedure`. The values of the parameters are stored to the properties *groupVar* and *sumVar* of the class instance.
- The value passed in as the procedure name is stored to the *procName* property. The `spss.BaseProcedure` class allows for an optional *omsIdentifier* property that specifies the command name associated with output from this procedure when routing the output with OMS (Output Management System), as used in the `COMMANDS` keyword of the `OMS` command. If *omsIdentifier* is an empty string then the value of *procName* is used as the OMS identifier. Although specifying a non-blank value of the *omsIdentifier* property is optional, the property itself must be included.

Note:

- The body of the procedure is contained within the `execUserProcedure` method, which overrides that method in `spss.BaseProcedure`. The procedure reads the data to calculate group sums and group case counts and creates a pivot table populated with the group means.
- The necessary calls to `spss.StartProcedure` and `spss.EndProcedure` are handled by `spss.BaseProcedure`.

Saving and Running Procedure Classes

Once you have written a procedure class, you will want to save it in a Python module on the Python search path so that you can call it. A Python module is simply a text file containing Python definitions and statements. You can create a module with a Python IDE, or with any text editor, by saving a file with an extension of *.py*. The name of the file, without the *.py* extension, is then the name of the module. You can have many classes in a single module. To be sure that Python can find your new module, you may want to save it to your Python "site-packages" directory, typically */Python27/Lib/site-packages*.

For the example procedure class described above, you might choose to save the class definition to a Python module named *myprocs.py*. And be sure to include an `import spss` statement in the module. Sample command syntax to instantiate this class and run the procedure is:

```
import spss, myprocs
spss.Submit("GET FILE='/examples/data/Employee data.sav'.")
proc = myprocs.groupMeans("mycompany.com.groupMeans","educ","salary")
proc.execProcedure()
```

- The import statement containing `myprocs` makes the contents of the Python module *myprocs.py* available to the current session (assuming that the module is on the Python search path).
- The call to `myprocs.groupMeans` creates an instance of the `groupMeans` class. The variables *educ* and *salary* in */examples/data/Employee data.sav* are used as the grouping variable and the variable for which means are calculated.
- Output from the procedure is associated with the name *mycompany.com.groupMeans*. This is the name that appears in the outline pane of the Viewer associated with output produced by the procedure. It is also the command name associated with this procedure when routing output from this procedure with OMS (Output Management System). In order that names associated with procedure output not conflict with names of existing IBM SPSS Statistics commands (when working with OMS), it is recommended that they have the form *yourcompanyname.com.procedurename*. See the topic "spss.StartProcedure Function" on page 85 for more information.

Result

	Current Salary
Educational Level (years)	Mean
8	24399
12	25887
14	31625
15	31685
16	48226
17	59527
18	65128
19	72520
20	64313
21	65000

Figure 15. Output from the groupMeans procedure

spss.CreateXPathDictionary Function

spss.CreateXPathDictionary(handle). *Creates an XPath dictionary DOM for the active dataset that can be accessed with XPath expressions.* The argument is a handle name, used to identify this DOM in subsequent `spss.EvaluateXPath` and `spss.DeleteXPathHandle` functions.

Example

```
handle='demo'
spss.CreateXPathDictionary(handle)
```

- The XPath dictionary DOM for the current active dataset is assigned the handle name *demo*. Any subsequent `spss.EvaluateXPath` or `spss.DeleteXPathHandle` functions that reference this dictionary DOM must use this handle name.

spss.Cursor Class

spss.Cursor(var, accessType, cvtDates, isBinary). *Provides the ability to read cases, append cases, and add new variables to the active dataset.*

- The optional argument *var* specifies a tuple or a list of variable index values representing position in the active dataset, starting with 0 for the first variable in file order. This argument is used in read or write mode to specify a subset of variables to return when reading case data from the active dataset. If the argument is omitted, all variables are returned. The argument has no effect if used in append mode.
- The optional argument *accessType* specifies one of three usage modes: read ('r'), write ('w'), and append ('a'). The default is read mode.
- The optional argument *cvtDates* specifies a set of IBM SPSS Statistics variables with date or datetime formats to convert to Python `datetime.datetime` objects when reading data from IBM SPSS Statistics. The argument is a sequence of variable index values representing position in the active dataset, starting with 0 for the first variable in file order. If the optional argument *var* is specified, then *cvtDates* must be a subset of the index values specified for *var*. You can specify to convert all date or datetime format variables with `cvtDates='ALL'`, or by setting *cvtDates* to a list or tuple with the single element `'ALL'`, as in `cvtDates=['ALL']`. When `'ALL'` is specified in conjunction with *var*, it refers to all variables specified in *var*. If *cvtDates* is omitted, then no conversions are performed. Variables included in *cvtDates* that do not have a date or datetime format are ignored in terms of the conversion. *cvtDates* applies to read and write mode and cannot be used in append mode.

  Note: Values of variables with date or datetime formats that are not converted with *cvtDates* are returned as integers representing the number of seconds from October 14, 1582.
- The optional Boolean argument *isBinary* (introduced in version 22) specifies the method that is used by the `Cursor` class to work with the data in the active dataset. It has no effect on `Cursor` functionality. By default *isBinary* is set to `True`, which typically provides the best performance but might require more temporary disk space. When *isBinary* is set to `False`, the `Cursor` class uses the same method for working with the data as in versions before version 22.
- You cannot use the `spss.Submit` function while a data cursor is open. You must close or delete the cursor first.
- Only one data cursor can be open at any point in the program block. To define a new data cursor, you must first close or delete the previous one. If you need to concurrently work with the data from multiple datasets, consider using the `Dataset` class.
- Instances of the `Cursor` class are implicitly deleted at the end of a `BEGIN PROGRAM-END PROGRAM` block, and therefore they do not persist across `BEGIN PROGRAM-END PROGRAM` blocks.
- The `Cursor` class honors case filters specified with the `FILTER` or `USE` commands.

Read Mode

This is the default for the `Cursor` class and provides the ability to read case data from the active dataset. Read mode is specified with `spss.Cursor(accessType='r')` or simply `spss.Cursor()`.

Note: For users of a 14.0.x version of the plug-in who are upgrading to a newer version, this mode is equivalent to `spss.Cursor(n)` in 14.0.x versions. No changes to your 14.0.x code for the `Cursor` class are required to run the code with a newer version.

The `Cursor` methods fetchone, fetchmany, and fetchall are used to retrieve cases from the active dataset.

```
DATA LIST FREE /var1 (F) var2 (A2)  var3 (F).
BEGIN DATA
11 ab 13
21 cd 23
31 ef  33
END DATA.
BEGIN PROGRAM.
import spss
dataCursor=spss.Cursor()
oneRow=dataCursor.fetchone()
dataCursor.close()
i=[0]
dataCursor=spss.Cursor(i)
oneVar=dataCursor.fetchall()
dataCursor.close()
print "One row (case): ", oneRow
print "One column (variable): ", oneVar
END PROGRAM.
```

Result

```
One row (case):  (11.0, 'ab', 13.0)
One column (variable):  ((11.0,), (21.0,), (31.0,))
```

- Cases from the active dataset are returned as a single tuple for `fetchone` and a list of tuples for `fetchall`.
- Each tuple represents the data for one case. For `fetchall` the tuples are arranged in the same order as the cases in the active dataset.
- Each element in a tuple contains the data value for a specific variable. The order of variable values within a tuple is the order specified by the optional argument *var* to the `Cursor` class, or file order if *var* is omittted.

```
*System- and user-missing values.
DATA LIST LIST (',') /numVar (f) stringVar (a4).
BEGIN DATA
1,a
,b
3,,
4,d
END DATA.
MISSING VALUES stringVar (' ').
BEGIN PROGRAM.
import spss
dataCursor=spss.Cursor()
print dataCursor.fetchall()
dataCursor.close()
END PROGRAM.
```

Result

```
((1.0, 'a   '), (None, 'b   '), (3.0, None), (4.0, 'd   '))
```

- String values are right-padded to the defined width of the string variable.
- System-missing values are always converted to the Python data type *None*.
- By default, user-missing values are converted to the Python data type *None*. You can use the SetUserMissingInclude method to specify that user-missing values be treated as valid.

Write Mode

This mode is used to add new variables, along with their case values, to an existing dataset. It cannot be used to append cases to the active dataset. Write mode is specified with `spss.Cursor(accessType='w')`.

- All of the methods available in read mode are also available in write mode.
- When adding new variables, the `CommitDictionary` method must be called after the statements defining the new variables and prior to setting case values for those variables. You cannot add new variables to an empty dataset.
- When setting case values for new variables, the `CommitCase` method must be called for each case that is modified. The `fetchone` method is used to advance the record pointer by one case, or you can use the `fetchmany` method to advance the record pointer by a specified number of cases.

  Note: If a case filter (specified with the `FILTER` or `USE` command) is in effect,`fetchone` and `fetchmany` advance the record pointer through the set of cases that have not been filtered out.
- Changes to the active dataset do not take effect until the cursor is closed.
- Write mode supports multiple data passes and allows you to add new variables on each pass. In the case of multiple data passes where you need to add variables on a data pass other than the first, you must call the `AllocNewVarsBuffer` method to allocate the buffer size for the new variables. When used, `AllocNewVarsBuffer` must be called before reading any data with `fetchone`, `fetchmany`, or `fetchall`.
- The `Cursor` methods SetVarNameAndType and SetOneVarNameAndType are used to add new variables to the active dataset, and the methods SetValueChar and SetValueNumeric are used to set case values.

Example

In this example, we create a new numeric variable and a new string variable and set their values for all cases.

```
DATA LIST FREE /var1 (F) var2 (A2)  var3 (F).
BEGIN DATA
11 ab 13
21 cd 23
31 ef  33
END DATA.
BEGIN PROGRAM.
import spss
cur=spss.Cursor(accessType='w')
cur.SetVarNameAndType(['var4','strvar'],[0,8])
cur.SetVarFormat('var4',5,2,0)
cur.CommitDictionary()
for i in range(cur.GetCaseCount()):
   cur.fetchone()
   cur.SetValueNumeric('var4',4+10*(i+1))
   cur.SetValueChar('strvar','row' + str(i+1))
   cur.CommitCase()
cur.close()
END PROGRAM.
```

- An instance of the `Cursor` class in write mode is created and assigned to the variable *cur*.
- The SetVarNameAndType method is used to add two new variables to the active dataset. *var4* is a numeric variable and *strvar* is a string variable of width 8.
- SetVarFormat sets the display format for *var4*. The integers 5, 2, and 0 specify the format type (5 is standard numeric), the defined width, and the number of decimal digits respectively.
- The `CommitDictionary` method is called to commit the new variables to the cursor before populating their case values.
- The `SetValueNumeric` and `SetValueChar` methods are used to set the case values of the new variables. The `CommitCase` method is called to commit the changes for each modified case.
- `fetchone` advances the record pointer to the next case.

Example: Setting Values for Specific Cases

In this example, we create new variables and set their values for specific cases. The `fetchone` method is used to advance the record pointer to the desired cases.

```
DATA LIST FREE /code (A1) loc (A3) emp (F) dtop (F) ltop (F).
BEGIN DATA
H NY 151 127 24
W CHI 17 4 0
S CHI 9 3 6
W ATL 12 3 0
W SDG 13 4 0
S ATL 10 3 7
S SDG 11 3 8
END DATA.
BEGIN PROGRAM.
import spss
cur=spss.Cursor(accessType='w')
cur.SetVarNameAndType(['emp_est','dtop_est','ltop_est'],[0,0,0])
cur.SetVarFormat('emp_est',5,2,0)
cur.SetVarFormat('dtop_est',5,2,0)
cur.SetVarFormat('ltop_est',5,2,0)
cur.CommitDictionary()
for i in range(cur.GetCaseCount()):
   row=cur.fetchone()
   if (row[0].lower()=='s'):
      cur.SetValueNumeric('emp_est',1.2*row[2])
      cur.SetValueNumeric('dtop_est',1.2*row[3])
      cur.SetValueNumeric('ltop_est',1.2*row[4])
      cur.CommitCase()
cur.close()
END PROGRAM.
```

Example: Multiple Data Passes

In this example, we read the data, calculate a summary statistic, and use a second data pass to add a summary variable to the active dataset.

```
DATA LIST FREE /var (F).
BEGIN DATA
57000
40200
21450
21900
END DATA.
BEGIN PROGRAM.
import spss
cur=spss.Cursor(accessType='w')
cur.AllocNewVarsBuffer(8)
total=0
for i in range(spss.GetCaseCount()):
   total+=cur.fetchone()[0]
meanVal=total/spss.GetCaseCount()
cur.reset()
cur.SetOneVarNameAndType('mean',0)
cur.CommitDictionary()
for i in range(spss.GetCaseCount()):
   row=cur.fetchone()
   cur.SetValueNumeric('mean',meanVal)
   cur.CommitCase()
cur.close()
END PROGRAM.
```

- Because we will be adding a new variable on the second data pass, the `AllocNewVarsBuffer` method is called to allocate the required space. In the current example, we are creating a single numeric variable, which requires eight bytes.
- The first `for` loop is used to read the data and total the case values.
- After the data pass, the `reset` method must be called prior to defining new variables.
- The second data pass (second `for` loop) is used to add the mean value of the data as a new variable.

Append Mode

This mode is used to append new cases to the active dataset. It cannot be used to add new variables or read case data from the active dataset. A dataset must contain at least one variable in order to append cases to it, but it need not contain any cases. Append mode is specified with `spss.Cursor(accessType='a')`.

- The `CommitCase` method must be called for each case that is added.
- The `EndChanges` method must be called before the cursor is closed.
- Changes to the active dataset do not take effect until the cursor is closed.
- A numeric variable whose value is not specified (for a new case) is set to the system-missing value.
- A string variable whose value is not specified (for a new case) will have a blank value. The value will be valid unless you explicitly define the blank value to be missing for that variable.
- The `Cursor` methods SetValueChar and SetValueNumeric are used to set variable values for new cases.

Example

```
DATA LIST FREE /var1 (F) var2 (A2)  var3 (F).
BEGIN DATA
11 ab 13
21 cd 23
31 ef 33
END DATA.
BEGIN PROGRAM.
import spss
cur=spss.Cursor(accessType='a')
ncases=cur.GetCaseCount()
newcases=2
for i in range(newcases):
   cur.SetValueNumeric('var1',1+10*(ncases+i+1))
   cur.SetValueNumeric('var3',3+10*(ncases+i+1))
   cur.CommitCase()
cur.EndChanges()
cur.close()
END PROGRAM.
```

- An instance of the `Cursor` class in append mode is created and assigned to the variable *cur*.
- The `SetValueNumeric` method is used to set the case values of *var1* and *var3* for two new cases. No value is specified for *var2*. The `CommitCase` method is called to commit the values for each case.

- The `EndChanges` method is called to commit the new cases to the cursor.

spss.Cursor Methods

Each usage mode of the `Cursor` class supports its own set of methods, as shown in the table below. Descriptions of each method follow.

Table 2. Usage modes for Cursor class methods

Method	Read mode	Write mode	Append mode
AllocNewVarsBuffer		X	
close	X	X	X
CommitCase		X	X
CommitDictionary		X	
EndChanges			X
fetchall	X	X**	
fetchmany	X	X**	
fetchone	X	X	
GetCaseCount	X	X	X
GetDataFileAttributeNames	X	X	X
GetDataFileAttributes	X	X	X
GetMultiResponseSetNames	X	X	X
GetMultiResponseSet	X	X	X
GetVarAttributeNames	X	X	X
GetVarAttributes	X	X	X
GetVariableCount	X	X	X
GetVariableFormat	X	X	X
GetVariableLabel	X	X	X
GetVariableMeasurementLevel	X	X	X
GetVariableName	X	X	X
GetVariableRole	X	X	X
GetVariableType	X	X	X
GetVarMissingValues	X	X	X
IsEndSplit	X	X	
reset	X	X	X
SetFetchVarList	X	X	
SetOneVarNameAndType		X	
SetUserMissingInclude	X	X	
SetValueChar		X	X
SetValueNumeric		X	X
SetVarAlignment		X	
SetVarAttributes		X	
SetVarCMissingValues		X	
SetVarCValueLabel		X	
SetVarFormat		X	
SetVarLabel		X	

Table 2. Usage modes for Cursor class methods (continued)

Method	Read mode	Write mode	Append mode
SetVarMeasureLevel		X	
SetVarNameAndType		X	
SetVarNMissingValues		X	
SetVarNValueLabel		X	
SetVarRole		X	

** This method is primarily for use in read mode.

Note

The `Cursor` class `Get` methods (for instance, `GetCaseCount`, `GetVariableCount`, and so on) listed above have the same specifications as the functions in the `spss` module of the same name. For example, the specifications for the `Cursor` method `GetCaseCount` are the same as those for the `spss.GetCaseCount` function. While a cursor is open, both sets of functions return information about the current cursor and give identical results. In the absence of a cursor, the `spss` module functions retrieve information about the active dataset. Refer to the entries for the corresponding `spss` module functions for specifications of these `Cursor` methods.

AllocNewVarsBuffer Method: .AllocNewVarsBuffer(bufSize). *Specifies the buffer size, in bytes, to use when adding new variables in the context of multiple data passes.* The argument *bufSize* is a positive integer large enough to accommodate all new variables to be created by a given write cursor. Each numeric variable requires eight bytes. For each string variable, you should allocate a size that is an integer multiple of eight bytes, and large enough to store the defined length of the string (one byte per character). For example, you would allocate eight bytes for strings of length 1–8 and 16 bytes for strings of length 9–16.

- This method is only available in write mode.
- `AllocNewVarsBuffer` is required in the case of multiple data passes when you need to add variables on a data pass other than the first. When used, it must be called before reading any data with `fetchone`, `fetchmany`, or `fetchall`.
- `AllocNewVarsBuffer` can only be called once for a given write cursor instance.
- Specifying a larger buffer size than is required has no effect on the result.

For an example of the `AllocNewVarsBuffer` method, see the example on multiple data passes in the topic on "Write Mode" on page 41.

close Method: .close(). *Closes the cursor.* You cannot use the `spss.Submit` function while a data cursor is open. You must close or delete the cursor first.

- This method is available in read, write, or append mode.
- When appending cases, you must call the EndChanges method before the `close` method.
- Cursors are implicitly closed at the end of a `BEGIN PROGRAM-END PROGRAM` block.

Example

```
cur=spss.Cursor()
data=cur.fetchall()
cur.close()
```

CommitCase Method: .CommitCase(). *Commits changes to the current case in the current cursor.* This method must be called for each case that is modified, including existing cases modified in write mode and new cases created in append mode.

- This method is available in write or append mode.

- When working in write mode, you advance the record pointer by calling the `fetchone` method. To modify the first case, you must first call `fetchone`.
- When working in append mode, the cursor is ready to accept values for a new record (using `SetValueNumeric` and `SetValueChar`) once `CommitCase` has been called for the previous record.
- Changes to the active dataset take effect when the cursor is closed.

For an example of using `CommitCase` in write mode, see the topic on write mode "Write Mode" on page 41. For an example of using `CommitCase` in append mode, see the topic on append mode "Append Mode" on page 43.

CommitDictionary Method: .CommitDictionary(). *Commits new variables to the current cursor.*

- This method is only available in write mode.
- When adding new variables, you must call this method before setting case values for the new variables.
- Changes to the active dataset take effect when the cursor is closed.

Example

```
DATA LIST FREE /var1 (F) var2 (A2)  var3 (F).
BEGIN DATA
11 ab 13
21 cd 23
31 ef  33
END DATA.
BEGIN PROGRAM.
import spss
cur=spss.Cursor(accessType='w')
cur.SetVarNameAndType(['numvar'],[0])
cur.SetVarLabel('numvar','New numeric variable')
cur.SetVarFormat('numvar',5,2,0)
cur.CommitDictionary()
for i in range(cur.GetCaseCount()):
   cur.fetchone()
   cur.SetValueNumeric('numvar',4+10*(i+1))
   cur.CommitCase()
cur.close()
END PROGRAM.
```

EndChanges Method: .EndChanges(). *Specifies the end of appending new cases.* This method must be called before the cursor is closed.

- This method can only be called once for a given `Cursor` instance and is only available in append mode.
- Changes to the active dataset take effect when the cursor is closed.

For an example of using `EndChanges`, see the topic on append mode "Append Mode" on page 43.

fetchall Method: .fetchall(). *Fetches all (remaining) cases from the active dataset, or if there are splits, the remaining cases in the current split.* If there are no remaining rows, the result is an empty tuple.

- This method is available in read or write mode.
- When used in write mode, calling `fetchall` will position the record pointer at the last case of the active dataset, or if there are splits, the last case of the current split.
- Cases from the active dataset are returned as a list of tuples. Each tuple represents the data for one case, and the tuples are arranged in the same order as the cases in the active dataset. Each element in a tuple contains the data value for a specific variable. The order of variable values within a tuple is the order specified by the variable index values in the optional argument *n* to the `Cursor` class, or file order if *n* is omitted. For example, if *n*=[5,2,7] the first tuple element is the value of the variable with index value 5, the second is the variable with index value 2, and the third is the variable with index value 7.
- String values are right-padded to the defined width of the string variable.
- System-missing values are always converted to the Python data type *None*.
- By default, user-missing values are converted to the Python data type *None*. You can use the SetUserMissingInclude method to specify that user-missing values be treated as valid.

- Values of variables with time formats are returned as integers representing the number of seconds from midnight.
- By default, values of variables with date or datetime formats are returned as integers representing the number of seconds from October 14, 1582. You can specify to convert values of those variables to Python `datetime.datetime` objects with the *cvtDates* argument to the `spss.Cursor` function. See the topic "spss.Cursor Class" on page 39 for more information.
- If a weight variable has been defined for the active dataset, then cases with zero, negative, or missing values for the weighting variable are skipped when fetching data with `fetchone`, `fetchall`, or `fetchmany`. If you need to retrieve all cases when weighting is in effect, then you can use the Dataset class.
- The `fetchone`, `fetchall`, and `fetchmany` methods honor case filters specified with the `FILTER` or `USE` commands.

```
DATA LIST FREE /var1 (F) var2 (A2)  var3 (F).
BEGIN DATA
11 ab 13
21 cd 23
31 ef  33
END DATA.
BEGIN PROGRAM.
import spss
dataCursor=spss.Cursor()
dataFile=dataCursor.fetchall()
for i in enumerate(dataFile):
  print i
print dataCursor.fetchall()
dataCursor.close()
END PROGRAM.
```

Result

```
(0, (11.0, 'ab', 13.0))
(1, (21.0, 'cd', 23.0))
(2, (31.0, 'ef', 33.0))
()
```

fetchall with Variable Index

```
DATA LIST FREE /var1 var2 var3.
BEGIN DATA
1 2 3
1 4 5
2 5 7
END DATA.
BEGIN PROGRAM.
import spss
i=[0]
dataCursor=spss.Cursor(i)
oneVar=dataCursor.fetchall()
uniqueCount=len(set(oneVar))
print oneVar
print spss.GetVariableName(0), " has ", uniqueCount, " unique values."
dataCursor.close()
END PROGRAM.
```

Result

```
((1.0,), (1.0,), (2.0,))
var1  has  2  unique values.
```

fetchmany Method: .fetchmany(n). *Fetches the next n cases from the active dataset, where n is a positive integer.* If the value of *n* is greater than the number of remaining cases (and the dataset does not contain splits), it returns the value of all the remaining cases. In the case that the active dataset has splits, if *n* is greater than the number of remaining cases in the current split, it returns the value of the remaining cases in the split. If there are no remaining cases, the result is an empty tuple.

- This method is available in read or write mode.
- When used in write mode, calling `fetchmany(n)` will position the record pointer at case `n` of the active dataset. In the case that the dataset has splits and `n` is greater than the number of remaining cases in the current split, `fetchmany(n)` will position the record pointer at the end of the current split.

- Cases from the active dataset are returned as a list of tuples. Each tuple represents the data for one case, and the tuples are arranged in the same order as the cases in the active dataset. Each element in a tuple contains the data value for a specific variable. The order of variable values within a tuple is the order specified by the variable index values in the optional argument *n* to the `Cursor` class, or file order if *n* is omitted. For example, if *n*=[5,2,7] the first tuple element is the value of the variable with index value 5, the second is the variable with index value 2, and the third is the variable with index value 7.
- String values are right-padded to the defined width of the string variable.
- System-missing values are always converted to the Python data type *None*.
- By default, user-missing values are converted to the Python data type *None*. You can use the SetUserMissingInclude method to specify that user-missing values be treated as valid.
- Values of variables with time formats are returned as integers representing the number of seconds from midnight.
- By default, values of variables with date or datetime formats are returned as integers representing the number of seconds from October 14, 1582. You can specify to convert values of those variables to Python `datetime.datetime` objects with the *cvtDates* argument to the `spss.Cursor` function. See the topic "spss.Cursor Class" on page 39 for more information.
- If a weight variable has been defined for the active dataset, then cases with zero, negative, or missing values for the weighting variable are skipped when fetching data with `fetchone`, `fetchall`, or `fetchmany`. If you need to retrieve all cases when weighting is in effect, then you can use the Dataset class.
- The `fetchone`, `fetchall`, and `fetchmany` methods honor case filters specified with the `FILTER` or `USE` commands.

```
DATA LIST FREE /var1 (F) var2 (A2)  var3 (F).
BEGIN DATA
11 ab 13
21 cd 23
31 ef  33
END DATA.
BEGIN PROGRAM.
import spss
dataCursor=spss.Cursor()
n=2
print dataCursor.fetchmany(n)
print dataCursor.fetchmany(n)
print dataCursor.fetchmany(n)
dataCursor.close()
END PROGRAM.
```

Result

```
((11.0, 'ab', 13.0), (21.0, 'cd', 23.0))
((31.0, 'ef', 33.0),)
()
```

fetchone Method: .fetchone(). *Fetches the next row (case) from the active dataset.* The result is a single tuple or the Python data type *None* after the last row has been read. A value of *None* is also returned at a split boundary. In this case, a subsequent call to `fetchone` will retrieve the first case of the next split group.

- This method is available in read or write mode.
- Each element in the returned tuple contains the data value for a specific variable. The order of variable values in the tuple is the order specified by the variable index values in the optional argument *n* to the `Cursor` class, or file order if *n* is omitted. For example, if *n*=[5,2,7] the first tuple element is the value of the variable with index value 5, the second is the variable with index value 2, and the third is the variable with index value 7.
- String values are right-padded to the defined width of the string variable.
- System-missing values are always converted to the Python data type *None*.
- By default, user-missing values are converted to the Python data type *None*. You can use the SetUserMissingInclude method to specify that user-missing values be treated as valid.
- Values of variables with time formats are returned as integers representing the number of seconds from midnight.

- By default, values of variables with date or datetime formats are returned as integers representing the number of seconds from October 14, 1582. You can specify to convert values of those variables to Python `datetime.datetime` objects with the *cvtDates* argument to the `spss.Cursor` function. See the topic "spss.Cursor Class" on page 39 for more information.
- If a weight variable has been defined for the active dataset, then cases with zero, negative, or missing values for the weighting variable are skipped when fetching data with `fetchone`, `fetchall`, or `fetchmany`. If you need to retrieve all cases when weighting is in effect, then you can use the Dataset class.
- The `fetchone`, `fetchall`, and `fetchmany` methods honor case filters specified with the `FILTER` or `USE` commands.

```
DATA LIST FREE /var1 var2 var3.
BEGIN DATA
1 2 3
4 5 6
END DATA.
BEGIN PROGRAM.
import spss
dataCursor=spss.Cursor()
firstRow=dataCursor.fetchone()
secondRow=dataCursor.fetchone()
thirdRow=dataCursor.fetchone()
print "First row: ",firstRow
print "Second row ",secondRow
print "Third row...there is NO third row: ",thirdRow
dataCursor.close()
END PROGRAM.
```

Result

```
First row:  (1.0, 2.0, 3.0)
Second row  (4.0, 5.0, 6.0)
Third row...there is NO third row:  None
```

IsEndSplit Method: .IsEndSplit(). *Indicates if the cursor position has crossed a split boundary.* The result is Boolean—*True* if a split boundary has been crossed, otherwise *False*. This method is used in conjunction with the SplitChange function when creating custom pivot tables from data with splits.

- This method is available in read or write mode.
- The value returned from the `fetchone` method is *None* at a split boundary. Once a split has been detected, you will need to call `fetchone` again to retrieve the first case of the next split group.
- `IsEndSplit` returns *True* when the end of the dataset has been reached. Although a split boundary and the end of the dataset both result in a return value of *True* from `IsEndSplit`, the end of the dataset is identified by a return value of *None* from a subsequent call to `fetchone`, as shown in the following example.

Example

```
GET FILE='/examples/data/employee data.sav'.
SORT CASES BY GENDER.
SPLIT FILE LAYERED BY GENDER.

BEGIN PROGRAM.
import spss
i=0
cur=spss.Cursor()
while True:
   cur.fetchone()
   i+=1
   if cur.IsEndSplit():
      # Try to fetch the first case of the next split group
      if not None==cur.fetchone():
         print "Found split end. New split begins at case: ", i
      else:
         #There are no more cases, so quit
         break
cur.close()
END PROGRAM.
```

reset Method: .reset(). *Resets the cursor.*

- This method is available in read, write, or append mode.
- In read and write modes, `reset` moves the record pointer to the first case, allowing multiple data passes. In append mode, it deletes the current cursor instance and creates a new one.
- When executing multiple data passes, the `reset` method must be called prior to defining new variables on subsequent passes. For an example, see the topic on write mode.

Example

```
import spss
cur=spss.Cursor()
data=cur.fetchall()
cur.reset()
data10=cur.fetchmany(10)
cur.close()
```

SetFetchVarList: .SetFetchVarList(var). *Resets the list of variables to return when reading case data from the active dataset.* The argument *var* is a list or tuple of variable index values representing position in the active dataset, starting with 0 for the first variable in file order.

- This method is available in read or write mode.

Example

```
DATA LIST FREE /var1 (F) var2 (A2)  var3 (F).
BEGIN DATA
11 ab 13
21 cd 23
31 ef  33
END DATA.
BEGIN PROGRAM.
import spss
cur=spss.Cursor()
oneRow=cur.fetchone()
cur.SetFetchVarList([0])
cur.reset()
oneVar=cur.fetchall()
cur.close()
print "One row (case): ", oneRow
print "One column (variable): ", oneVar
END PROGRAM.
```

SetOneVarNameAndType Method: .SetOneVarNameAndType(varName,varType). *Creates one new variable in the active dataset.* The argument *varName* is a string that specifies the name of the new variable. The argument *varType* is an integer specifying the variable type of the new variable. You can create multiple variables with a single call using the SetVarNameAndType method.

- This method is only available in write mode.
- Numeric variables are specified by a value of 0 for the variable type. String variables are specified with a type equal to the defined length of the string (maximum of 32767).
- Use of the `SetOneVarNameAndType` method requires the `AllocNewVarsBuffer` method to allocate space for the variable.

Example

```
DATA LIST FREE /var1 (F) var2 (A2)  var3 (F).
BEGIN DATA
11 ab 13
21 cd 23
31 ef  33
END DATA.
BEGIN PROGRAM.
import spss
cur=spss.Cursor(accessType='w')
cur.AllocNewVarsBuffer(8)
cur.SetOneVarNameAndType('var4',0)
cur.SetVarFormat('var4',5,2,0)
cur.CommitDictionary()
for i in range(cur.GetCaseCount()):
   cur.fetchone()
```

```
   cur.SetValueNumeric('var4',4+10*(i+1))
   cur.CommitCase()
cur.close()
END PROGRAM.
```

SetUserMissingInclude Method: .SetUserMissingInclude(incMissing). *Specifies the treatment of user-missing values read from the active dataset.* The argument is a Boolean with *True* specifying that user-missing values be treated as valid. A value of *False* specifies that user-missing values should be converted to the Python data type *None*.

- By default, user-missing values are converted to the Python data type *None*.
- System-missing values are always converted to *None*.
- This method is available in read or write mode.

In this example, we will use the following data to demonstrate both the default behavior and the behavior when user missing values are treated as valid.

```
DATA LIST LIST (',') /numVar (f) stringVar (a4).
BEGIN DATA
1,a
,b
3,,
0,d
END DATA.
MISSING VALUES stringVar (' ') numVar(0).
```

This first `BEGIN PROGRAM` block demonstrates the default behavior.

```
BEGIN PROGRAM.
import spss
cur=spss.Cursor()
print cur.fetchall()
cur.close()
END PROGRAM.
```

Result

```
((1.0, 'a   '), (None, 'b   '), (3.0, None), (None, 'd   '))
```

This second `BEGIN PROGRAM` block demonstrates the behavior when user-missing values are treated as valid.

```
BEGIN PROGRAM.
import spss
cur=spss.Cursor()
cur.SetUserMissingInclude(True)
print cur.fetchall()
cur.close()
END PROGRAM.
```

Result

```
((1.0, 'a   '), (None, 'b   '), (3.0, '    '), (0.0, 'd   '))
```

SetValueChar Method: .SetValueChar(varName,varValue). *Sets the value for the current case for a string variable.* The argument *varName* is a string specifying the name of a string variable. The argument *varValue* is a string specifying the value of this variable for the current case.

- This method is available in write or append mode.
- The `CommitCase` method must called for each case that is modified. This includes new cases created in append mode.

Example

```
DATA LIST FREE /var1 (F) var2(F).
BEGIN DATA
11 12
21 22
31 32
END DATA.
BEGIN PROGRAM.
import spss
cur=spss.Cursor(accessType='w')
```

```
cur.SetVarNameAndType(['strvar'],[8])
cur.CommitDictionary()
for i in range(cur.GetCaseCount()):
   cur.fetchone()
   cur.SetValueChar('strvar','row' + str(i+1))
   cur.CommitCase()
cur.close()
END PROGRAM.
```

SetValueNumeric Method: .SetValueNumeric(varName,varValue). *Sets the value for the current case for a numeric variable.* The argument *varName* is a string specifying the name of a numeric variable. The argument *varValue* specifies the numeric value of this variable for the current case.

- This method is available in write or append mode.
- The `CommitCase` method must be called for each case that is modified. This includes new cases created in append mode.
- The Python data type *None* specifies a missing value for a numeric variable.
- Values of numeric variables with a date or datetime format should be specified as Python `time.struct_time` or `datetime.datetime` objects, which are then converted to the appropriate IBM SPSS Statistics value. Values of variables with `TIME` and `DTIME` formats should be specified as the number of seconds in the time interval.

Example

```
DATA LIST FREE /var1 (F) var2 (F).
BEGIN DATA
11 12
21 22
31 32
END DATA.
BEGIN PROGRAM.
import spss
cur=spss.Cursor(accessType='w')
cur.SetVarNameAndType(['var3'],[0])
cur.SetVarFormat('var3',5,2,0)
cur.CommitDictionary()
for i in range(cur.GetCaseCount()):
   cur.fetchone()
   cur.SetValueNumeric('var3',3+10*(i+1))
   cur.CommitCase()
cur.close()
END PROGRAM.
```

SetVarAlignment Method: .SetVarAlignment(varName,alignment). *Sets the alignment of data values in the Data Editor for a new variable.* It has no effect on the format of the variables or the display of the variables or values in other windows or printed results. The argument *varName* is a string specifying the name of a new variable. The argument *alignment* is an integer and can take on one of the following values: 0 (left), 1 (right), 2 (center).

- This method is only available in write mode.

Example

```
cur=spss.Cursor(accessType='w')
cur.SetVarNameAndType(['numvar'],[0])
cur.SetVarAlignment('numvar',0)
cur.CommitDictionary()
cur.close()
```

SetVarAttributes Method: .SetVarAttributes(varName,attrName,attrValue,index). *Sets a value in an attribute array for a new variable.* The argument *varName* is a string specifying the name of a new variable. The argument *attrName* is a string specifying the name of the attribute array. The argument *attrValue* is a string specifying the attribute value, and *index* is the index position in the array, starting with the index 0 for the first element in the array.

- This method is only available in write mode.
- An attribute array with one element is equivalent to an attribute that is not specified as an array.

Example

```
cur=spss.Cursor(accessType='w')
cur.SetVarNameAndType(['numvar'],[0])
cur.SetVarAttributes('numvar','myattribute','first element',0)
cur.SetVarAttributes('numvar','myattribute','second element',1)
cur.CommitDictionary()
cur.close()
```

SetVarCMissingValues Method: .SetVarCMissingValues(varName,missingVal1,missingVal2,missingVal3). *Sets user-missing values for a new string variable.* The argument *varName* is a string specifying the name of a new string variable. The optional arguments *missingVal1*, *missingVal2*, and *missingVal3* are strings, each of which can specify one user-missing value. Use the SetVarNMissingValues method to set missing values for new numeric variables.

- This method is only available in write mode.

Example

```
cur=spss.Cursor(accessType='w')
cur.SetVarNameAndType(['strvar'],[8])
cur.SetVarCMissingValues('strvar',' ','NA')
cur.CommitDictionary()
cur.close()
```

SetVarCValueLabel Method: .SetVarCValueLabel(varName,value,label). *Sets the value label of a single value for a new string variable.* The argument *varName* is a string specifying the name of a new string variable. The arguments *value* and *label* are strings specifying the value and the associated label. Use the SetVarNValueLabel method to set value labels for new numeric variables.

- This method is only available in write mode.

Example

```
cur=spss.Cursor(accessType='w')
cur.SetVarNameAndType(['strvar'],[8])
cur.SetVarCValueLabel('strvar','f','female')
cur.CommitDictionary()
cur.close()
```

SetVarFormat Method: .SetVarFormat(varName,type,width,decimals). *Sets the display format for a new variable.* The argument *varName* is a string specifying the name of a new variable. The argument *type* is an integer that specifies one of the available format types (see Appendix A, "Variable Format Types," on page 247). The argument *width* is an integer specifying the defined width, which must include enough positions to accommodate any punctuation characters such as decimal points, commas, dollar signs, or date and time delimiters. The optional argument *decimals* is an integer specifying the number of decimal digits for numeric formats.

Allowable settings for decimal and width depend on the specified type. For a list of the minimum and maximum widths and maximum decimal places for commonly used format types, see Variable Types and Formats in the Universals section of the *Command Syntax Reference*, available in PDF from the Help menu and also integrated into the overall Help system.

- This method is only available in write mode.
- Setting the argument *width* for a string variable will not change the defined length of the string. If the specified value does not match the defined length, it is forced to be the defined length.

Example

```
cur=spss.Cursor(accessType='w')
cur.SetVarNameAndType(['numvar'],[0])
cur.SetVarFormat('numvar',5,2,0)
cur.CommitDictionary()
cur.close()
```

SetVarLabel Method: .SetVarLabel(varName,varLabel). *Sets the variable label for a new variable.* The argument *varName* is a string specifying the name of a new variable. The argument *varLabel* is a string specifying the label.

- This method is only available in write mode.

Example

```
cur=spss.Cursor(accessType='w')
cur.SetVarNameAndType(['numvar'],[0])
cur.SetVarLabel('numvar','New numeric variable')
cur.CommitDictionary()
cur.close()
```

SetVarMeasureLevel Method: .SetVarMeasureLevel(varName,measureLevel). *Sets the measurement level for a new variable.* The argument *varName* is a string specifying the name of a new variable. The argument *measureLevel* is an integer specifying the measurement level: 2 (nominal), 3 (ordinal), 4 (scale).

- This method is only available in write mode.

Example

```
cur=spss.Cursor(accessType='w')
cur.SetVarNameAndType(['numvar'],[0])
cur.SetVarMeasureLevel('numvar',3)
cur.CommitDictionary()
cur.close()
```

SetVarNameAndType Method: .SetVarNameAndType(varName,varType). *Creates one or more new variables in the active dataset.* The argument *varName* is a list or tuple of strings that specifies the name of each new variable. The argument *varType* is a list or tuple of integers specifying the variable type of each variable named in *varName*. *varName* and *varType* must be the same length. For creating a single variable you can also use the SetOneVarNameAndType method.

- This method is only available in write mode.
- Numeric variables are specified by a value of 0 for the variable type. String variables are specified with a type equal to the defined length of the string (maximum of 32767).

Example

```
DATA LIST FREE /var1 (F) var2 (A2)  var3 (F).
BEGIN DATA
11 ab 13
21 cd 23
31 ef  33
END DATA.
BEGIN PROGRAM.
import spss
cur=spss.Cursor(accessType='w')
cur.SetVarNameAndType(['var4','strvar'],[0,8])
cur.SetVarFormat('var4',5,2,0)
cur.CommitDictionary()
for i in range(cur.GetCaseCount()):
   cur.fetchone()
   cur.SetValueNumeric('var4',4+10*(i+1))
   cur.SetValueChar('strvar','row' + str(i+1))
   cur.CommitCase()
cur.close()
END PROGRAM.
```

SetVarNMissingValues Method: .SetVarNMissingValues(varName,missingFormat,missingVal1,missingVal2,missingVal3). *Sets user-missing values for a new numeric variable.* The argument *varName* is a string specifying the name of a new numeric variable. The argument *missingFormat* has the value 0 for a discrete list of missing values (for example, 0, 9, 99), the value 1 for a range of missing values (for example, 9–99), and the value 2 for a combination of a discrete value and a range (for example, 0 and 9–99). Use the SetVarCMissingValues method to set missing values for new string variables.

- This method is only available in write mode.
- To specify *LO* and *HI* in missing value ranges, use the values returned by the spss.GetSPSSLowHigh function.

Table 3. Specifications for arguments to SetVarNMissingValues

missingFormat	missingVal1	missingVal2	missingVal3
0	Discrete value (optional)	Discrete value (optional)	Discrete value (optional)
1	Start point of range	End point of range	Not applicable
2	Start point of range	End point of range	Discrete value

Examples

Specify the three discrete missing values 0, 9, and 99 for a new variable.

```
cur=spss.Cursor(accessType='w')
cur.SetVarNameAndType(['numvar'],[0])
cur.SetVarNMissingValues('numvar',0,0,9,99)
cur.CommitDictionary()
cur.close()
```

Specify the range of missing values 9–99 for a new variable.

```
cur=spss.Cursor(accessType='w')
cur.SetVarNameAndType(['numvar'],[0])
cur.SetVarNMissingValues('numvar',1,9,99)
cur.CommitDictionary()
cur.close()
```

Specify the range of missing values 9–99 and the discrete missing value 0 for a new variable.

```
cur=spss.Cursor(accessType='w')
cur.SetVarNameAndType(['numvar'],[0])
cur.SetVarNMissingValues('numvar',2,9,99,0)
cur.CommitDictionary()
cur.close()
```

SetVarNValueLabel Method: .SetVarNValueLabel(varName,value,label). *Sets the value label of a single value for a new variable.* The argument *varName* is a string specifying the name of a new numeric variable. The argument *value* is a numeric value and *label* is the string specifying the label for this value. Use the SetVarCValueLabel method to set value labels for new string variables.

- This method is only available in write mode.

Example

```
cur=spss.Cursor(accessType='w')
cur.SetVarNameAndType(['numvar'],[0])
cur.SetVarNValueLabel('numvar',1,'female')
cur.CommitDictionary()
cur.close()
```

SetVarRole Method: .SetVarRole(varName,varRole). *Sets the role for a new variable.* The argument *varName* is a string specifying the name of a new variable. The argument *varRole* is a string specifying the role: "Input", "Target", "Both", "None", "Partition" or "Split".

- This method is only available in write mode.

Example

```
cur=spss.Cursor(accessType='w')
cur.SetVarNameAndType(['targetvar'],[0])
cur.SetVarRole('targetvar','Target')
cur.CommitDictionary()
cur.close()
```

spss.Dataset Class

spss.Dataset(name,hidden,cvtDates). *Provides the ability to create new datasets, read from existing datasets, and modify existing datasets.* A `Dataset` object provides access to the case data and variable information contained in a dataset, and allows you to read from the dataset, add new cases, modify existing cases, add new variables, and modify properties of existing variables.

An instance of the Dataset class can only be created within a data step or StartProcedure-EndProcedure block, and cannot be used outside of the data step or procedure block in which it was created. Data steps are initiated with the spss.StartDataStep function. You can also use the spss.DataStep class to implicitly start and end a data step without the need to check for pending transformations. See the topic "spss.DataStep Class" on page 71 for more information.

- The argument *name* is optional and specifies the name of an open dataset for which a Dataset object will be created. Note that this is the name as assigned by IBM SPSS Statistics or as specified with DATASET NAME. Specifying name="*" or omitting the argument will create a Dataset object for the active dataset. If the active dataset is unnamed, then a name will be automatically generated for it in the case that the Dataset object is created for the active dataset.
- If the Python data type *None* or the empty string '' is specified for *name*, then a new empty dataset is created. The name of the dataset is automatically generated and can be retrieved from the name property of the resulting Dataset object. The name cannot be changed from within the data step. To change the name, use the DATASET NAME command following spss.EndDataStep.

  A new dataset created with the Dataset class is not set to be the active dataset. To make the dataset the active one, use the spss.SetActive function.
- The optional argument *hidden* specifies whether the Data Editor window associated with the dataset is hidden--by default, it is displayed. Use hidden=True to hide the associated Data Editor window.
- The optional argument *cvtDates* specifies whether IBM SPSS Statistics variables with date or datetime formats are converted to Python datetime.datetime objects when reading data from IBM SPSS Statistics. The argument is a boolean--*True* to convert all variables with date or datetime formats, *False* otherwise. If *cvtDates* is omitted, then no conversions are performed.

  Note: Values of variables with date or datetime formats that are not converted with *cvtDates* are returned as integers representing the number of seconds from October 14, 1582.
- Instances of the Dataset class created within StartProcedure-EndProcedure blocks cannot be set as the active dataset.
- The Dataset class does not honor case filters specified with the FILTER or USE commands. If you need case filters to be honored, then consider using the Cursor class.
- For release 22 Fix Pack 1 and higher, the Dataset class supports caching. Caching typically improves performance when cases are modified in a random manner, and is specified with the cache property of a Dataset object.

The number of variables in the dataset associated with a Dataset instance is available using the len function, as in:

```
len(datasetObj)
```

Note: Datasets that are not required outside of the data step or procedure in which they were accessed or created should be closed prior to ending the data step or procedure in order to free the resources allocated to the dataset. This is accomplished by calling the close method of the Dataset object.

Example: Creating a New Dataset

```
BEGIN PROGRAM.
import spss
spss.StartDataStep()
datasetObj = spss.Dataset(name=None)
datasetObj.varlist.append('numvar',0)
datasetObj.varlist.append('strvar',1)
datasetObj.varlist['numvar'].label = 'Sample numeric variable'
datasetObj.varlist['strvar'].label = 'Sample string variable'
datasetObj.cases.append([1,'a'])
datasetObj.cases.append([2,'b'])
spss.EndDataStep()
END PROGRAM.
```

- You add variables to a dataset using the append (or insert) method of the VariableList object associated with the dataset. The VariableList object is accessed from the varlist property of the Dataset object, as in datasetObj.varlist. See the topic "VariableList Class" on page 65 for more information.

- Variable properties, such as the variable label and measurement level, are set through properties of the associated `Variable` object, accessible from the `VariableList` object. For example, `datasetObj.varlist['numvar']` accesses the `Variable` object associated with the variable *numvar*. See the topic "Variable Class" on page 66 for more information.
- You add cases to a dataset using the `append` (or `insert`) method of the `CaseList` object associated with the dataset. The `CaseList` object is accessed from the `cases` property of the `Dataset` object, as in `datasetObj.cases`. See the topic "CaseList Class" on page 62 for more information.

Example: Saving New Datasets

When creating new datasets that you intend to save, you'll want to keep track of the dataset names since the save operation is done outside of the associated data step.

```
DATA LIST FREE /dept (F2) empid (F4) salary (F6).
BEGIN DATA
7  57  57000
5  23  40200
3  62  21450
3  18  21900
5  21  45000
5  29  32100
7  38  36000
3  42  21900
7  11  27900
END DATA.
DATASET NAME saldata.
SORT CASES BY dept.
BEGIN PROGRAM.
import spss
with spss.DataStep():
   ds = spss.Dataset()
   # Create a new dataset for each value of the variable 'dept'
   newds = spss.Dataset(name=None)
   newds.varlist.append('dept')
   newds.varlist.append('empid')
   newds.varlist.append('salary')
   dept = ds.cases[0,0][0]
   dsNames = {newds.name:dept}
   for row in ds.cases:
      if (row[0] != dept):
         newds = spss.Dataset(name=None)
         newds.varlist.append('dept')
         newds.varlist.append('empid')
         newds.varlist.append('salary')
         dept = row[0]
         dsNames[newds.name] = dept
      newds.cases.append(row)
# Save the new datasets
for name,dept in dsNames.iteritems():
   strdept = str(dept)
   spss.Submit(r"""
   DATASET ACTIVATE %(name)s.
   SAVE OUTFILE='/mydata/saldata_%(strdept)s.sav'.
   """ %locals())
spss.Submit(r"""
DATASET ACTIVATE saldata.
DATASET CLOSE ALL.
""" %locals())
END PROGRAM.
```

- The code `newdsObj = spss.Dataset(name=None)` creates a new dataset. The name of the dataset is available from the *name* property, as in `newdsObj.name`. In this example, the names of the new datasets are stored to the Python dictionary *dsNames*.
- To save new datasets created with the `Dataset` class, use the `SAVE` command after calling `spss.EndDataStep`. In this example, `DATASET ACTIVATE` is used to activate each new dataset, using the dataset names stored in *dsNames*.

Example: Modifying Case Values

```
DATA LIST FREE /cust (F2) amt (F5).
BEGIN DATA
210 4500
242 6900
370 32500
END DATA.
```

```
BEGIN PROGRAM.
import spss
spss.StartDataStep()
datasetObj = spss.Dataset()
for i in range(len(datasetObj.cases)):
   # Multiply the value of amt by 1.05 for each case
   datasetObj.cases[i,1] = 1.05*datasetObj.cases[i,1][0]
spss.EndDataStep()
END PROGRAM.
```

- The `CaseList` object, accessed from the `cases` property of a `Dataset` object, allows you to read or modify case data. To access the value for a given variable within a particular case you specify the case number and the index of the variable (index values represent position in the active dataset, starting with 0 for the first variable in file order, and case numbers start from 0). For example, `datasetObj.cases[i,1]` specifies the value of the variable with index `1` for case number `i`.
- When reading case values, results are returned as a list. In the present example we're accessing a single value within each case so the list has one element.

See the topic "CaseList Class" on page 62 for more information.

Example: Comparing Datasets

`Dataset` objects allow you to concurrently work with the case data from multiple datasets. As a simple example, we'll compare the cases in two datasets and indicate identical cases with a new variable added to one of the datasets.

```
DATA LIST FREE /id (F2) salary (DOLLAR8) jobcat (F1).
BEGIN DATA
1 57000 3
3 40200 1
2 21450 1
END DATA.
SORT CASES BY id.
DATASET NAME empdata1.
DATA LIST FREE /id (F2) salary (DOLLAR8) jobcat (F1).
BEGIN DATA
3 41000 1
1 59280 3
2 21450 1
END DATA.
SORT CASES BY id.
DATASET NAME empdata2.
BEGIN PROGRAM.
import spss
spss.StartDataStep()
datasetObj1 = spss.Dataset(name="empdata1")
datasetObj2 = spss.Dataset(name="empdata2")
nvars = len(datasetObj1)
datasetObj2.varlist.append('match')
for i in range(len(datasetObj1.cases)):
   if datasetObj1.cases[i] == datasetObj2.cases[i,0:nvars]:
      datasetObj2.cases[i,nvars] = 1
   else:
      datasetObj2.cases[i,nvars] = 0
spss.EndDataStep()
END PROGRAM.
```

- The two datasets are first sorted by the variable *id* which is common to both datasets.
- Since `DATA LIST` creates unnamed datasets (the same is true for `GET`), the datasets are named using `DATASET NAME` so that you can refer to them when calling `spss.Dataset`.
- `datasetObj1` and `datasetObj2` are `Dataset` objects associated with the two datasets *empdata1* and *empdata2* to be compared.
- The code `datasetObj1.cases[i]` returns case number `i` from *empdata1*. The code `datasetObj2.cases[i,0:nvars]` returns the slice of case number `i` from *empdata2* that includes the variables with indexes 0,1,...,*nvars*-1.
- The new variable *match*, added to *empdata2*, is set to 1 for cases that are identical and 0 otherwise.

cases Property

The cases property of a Dataset object returns an instance of the CaseList class. The CaseList class provides access to the cases in the associated dataset, allowing you to read existing cases, modify case values, and add new cases. See the topic "CaseList Class" on page 62 for more information.

Example

```
import spss
spss.StartDataStep()
datasetObj = spss.Dataset('data1')
caseListObj = datasetObj.cases
spss.EndDataStep()
```

name Property

The name property of a Dataset object gets the name of the associated dataset. The name cannot be changed from within the data step. To change the name, use the DATASET NAME command following spss.EndDataStep.

Example

```
import spss
spss.StartDataStep()
datasetObj = spss.Dataset('data1')
datasetName = datasetObj.name
spss.EndDataStep()
```

varlist Property

The varlist property of a Dataset object returns an instance of the VariableList class. The VariableList class provides access to the variables in the associated dataset, allowing you to retrieve the properties of existing variables, modify variable properties, and add new variables to the dataset. See the topic "VariableList Class" on page 65 for more information.

Example

```
import spss
spss.StartDataStep()
datasetObj = spss.Dataset('data1')
varListObj = datasetObj.varlist
spss.EndDataStep()
```

dataFileAttributes Property

The dataFileAttributes property of a Dataset object gets or sets datafile attributes for the dataset. The dataFileAttributes property behaves like a Python dictionary in terms of getting, setting, and deleting values. A Python dictionary consists of a set of keys, each of which has an associated value that can be accessed simply by specifying the key. In the case of datafile attributes, each key is the name of an attribute and the associated value is the value of the attribute, which can be a single value or a list or tuple of values. A list or tuple of values specifies an attribute array.

- When setting attributes, attribute names and values must be given as quoted strings.

Retrieving Datafile Attributes. You retrieve datafile attributes for a dataset from the dataFileAttributes property of the associated Dataset object. You can retrieve the value of a particular attribute by specifying the attribute name, as in:

```
dsObj = spss.Dataset()
attr = dsObj.dataFileAttributes['attrName']
```

Attribute values are always returned as a tuple.

You can iterate through the set of datafile attributes using the data property, as in:

```
dsObj = spss.Dataset()
for attrName, attrValue in dsObj.dataFileAttributes.data.iteritems():
   print attrName, attrValue
```

Adding and Modifying Datafile Attributes. You can add new datafile attributes and modify existing ones. For example:

```
dsObj.dataFileAttributes['attrName'] = 'value'
```

- If the attribute *attrName* exists, it is updated with the specified value. If the attribute *attrName* doesn't exist, it is added to any existing ones for the dataset.

Resetting Datafile Attributes. You can reset the datafile attributes associated with a dataset. For example:

```
dsObj.dataFileAttributes = {'attr1':'value','attr2':['val1','val2']}
```

- You reset the datafile attributes by setting the *dataFileAttributes* property to a new Python dictionary. Any existing datafile attributes are cleared and replaced with the specified ones.

Deleting Datafile Attributes. You can delete a particular datafile attribute or all of them. For example:

```
#Delete a specified attribute
del dsObj.dataFileAttributes['attrName']
#Delete all attributes
del dsObj.dataFileAttributes
```

multiResponseSet Property

The `multiResponseSet` property of a `Dataset` object gets or sets multiple response sets for the dataset. The `multiResponseSet` property behaves like a Python dictionary in terms of getting, setting, and deleting values. A Python dictionary consists of a set of keys, each of which has an associated value that can be accessed simply by specifying the key. In the case of multiple response sets, each key is the name of a set and the associated value specifies the details of the set.

- The multiple response set name is a string of maximum length 63 bytes that must follow IBM SPSS Statistics variable naming conventions. If the specified name does not begin with a dollar sign ($), then one is added. If the name refers to an existing set, the set definition is overwritten.
- When setting a multiple response set, the details of the set are specified as a list or tuple with the following elements in the presented order.

  mrsetLabel. A string specifying a label for the set. The value cannot be wider than the limit for IBM SPSS Statistics variable labels.

  mrsetCodeAs. An integer or string specifying the variable coding: 1 or "Categories" for multiple category sets, 2 or "Dichotomies" for multiple dichotomy sets.

  mrsetCountedValue. A string specifying the value that indicates the presence of a response for a multiple dichotomy set. This is also referred to as the “counted” value. If the set type is numeric, the value must be a string representation of an integer. If the set type is string, the counted value, after trimming trailing blanks, cannot be wider than the narrowest elementary variable.

  varNames. A tuple or list of strings specifying the names of the elementary variables that define the set (the list must include at least two variables).
- When getting a multiple response set, the result is a tuple of 5 elements. The first element is the label, if any, for the set. The second element specifies the variable coding--'Categories' or 'Dichotomies'. The third element specifies the counted value and only applies to multiple dichotomy sets. The fourth element specifies the data type--'Numeric' or 'String'. The fifth element is a list of the elementary variables that define the set.

Retrieving Multiple Response Sets. You retrieve multiple response sets for a dataset from the `multiResponseSet` property of the associated `Dataset` object. You retrieve the value of a particular set by specifying the set name, as in:

```
dsObj = spss.Dataset()
mrset = dsObj.multiResponseSet['setName']
```

You can iterate through the multiple response sets using the `data` property, as in:

```
dsObj = spss.Dataset()
for name, set in dsObj.multiResponseSet.data.iteritems():
   print name, set
```

Adding and Modifying Multiple Response Sets. You can add new multiple response sets and modify details of existing ones. For example:

```
dsObj.multiResponseSet['$mltnews'] = \
                      ["News Sources",2,"1",["Newspaper","TV","Web"]]
```

- If the set *$mltnews* exists, it is updated with the specified values. If the set *$mltnews* doesn't exist, it is added to any existing ones for the dataset.

Resetting Multiple Response Sets. You can reset the multiple response sets associated with a dataset. For example:

```
dsObj.multiResponseSet = \
{'$mltnews':["News Sources",2,"1",["Newspaper","TV","Web"]],
 '$mltent':["Entertainment Sources",2,"1",["TV","Movies","Theatre","Music"]]}
```

- You reset the multiple response sets by setting the *multiResponseSet* property to a new Python dictionary. Any existing multiple response sets are cleared and replaced with the specified ones.

Deleting Multiple Response Sets. You can delete a particular multiple response set or all sets. For example:

```
#Delete a specified set
del dsObj.multiResponseSet['setName']
#Delete all sets
del dsObj.multiResponseSet
```

cache Property

The `cache` property of a `Dataset` object specifies whether caching is used when cases in the associated dataset are read or modified. Caching typically improves performance when cases are modified in a random manner. It is not recommended when cases are read or modified sequentially.

- The `cache` property is Boolean, where *True* specifies that caching is used. The default is *False*.
- When `cache=True`, you cannot make the following changes: append or insert cases, add or delete variables, change the variable type.
- The value of the `cache` property can be modified over the life of a `Dataset` object. Setting `cache=False` commits any changes that were specified while `cache` was set to *True*.
- The `cache` property is available for release 22 Fix Pack 1 and higher.

Example

```
import spss
spss.StartDataStep()
datasetObj = spss.Dataset('data1')
datasetObj.cache = True
spss.EndDataStep()
```

optimized Property

Note: This property is deprecated for release 22 Fix Pack 1 and higher. Please use the `cache` property instead.

close Method

.close(). *Closes the dataset.* This method closes a dataset accessed through or created by the `Dataset` class. It cannot be used to close an arbitrary open dataset. When used, it must be called prior to `EndDataStep` or `EndProcedure`.

- If the associated dataset is not the active dataset, that dataset is closed and no longer available in the session. The associated dataset will, however, remain open outside of the data step or procedure in which it was created if the `close` method is not called.
- If the associated dataset is the active dataset, the association with the dataset's name is broken. The active dataset remains active but has no name.

Note: Datasets that are not required outside of the data step or procedure in which they were accessed or created should be closed prior to ending the data step or procedure in order to free the resources allocated to the dataset.

Example

```
import spss
spss.StartDataStep()
datasetObj1 = spss.Dataset()
datasetObj2 = datasetObj1.deepCopy(name="copy1")
datasetObj1.close()
spss.EndDataStep()
```

deepCopy Method

.deepCopy(name). *Creates a copy of the Dataset instance as well as a copy of the dataset associated with the instance.* The argument is required and specifies the name of the new dataset, as a quoted string. The name cannot be the name of the dataset being copied or a blank string. If `'*'` is specified the copy becomes the active dataset with a name that is automatically generated. You can retrieve the dataset name from the `name` property of the new `Dataset` instance.

Example

```
import spss
spss.StartDataStep()
datasetObj1 = spss.Dataset()
# Make a copy of the active dataset and assign it the name "copy1"
datasetObj2 = datasetObj1.deepCopy(name="copy1")
spss.EndDataStep()
```

CaseList Class

The `CaseList` class provides access to the cases in a dataset, allowing you to read existing cases, modify case values, and add new cases. You get an instance of the `CaseList` class from the `cases` property of the `Dataset` class, as in:

```
datasetObj = spss.Dataset('data1')
caseListObj = datasetObj.cases
```

The number of cases in a `CaseList` instance, which is also the number of cases in the associated dataset, is available using the `len` function, as in:

```
len(caseListObj)
```

Note: An instance of the `CaseList` class can only be created within a data step, and cannot be used outside of the data step in which it was created. Data steps are initiated with the `spss.StartDataStep` function.

Looping through the cases in an instance of CaseList. You can loop through the cases in an instance of the `CaseList` class. For example:

```
for row in datasetObj.cases:
   print row
```

- On each iteration of the loop, *row* is a case from the associated dataset.

Note: The `CaseList` class does not provide any special handling for datasets with split groups--it simply returns all cases in the dataset. If you need to differentiate the data in separate split groups, consider using the `Cursor` class to read your data, or you may want to use the `spss.GetSplitVariableNames` function to manually process the split groups.

Accessing specific cases and case values. You can access a specific case or a range of cases, and you can specify a variable or a range of variables within those cases. The result is a list, even if accessing the value of a single variable within a single case.

- System-missing values are returned as the Python data type *None*.
- Values of variables with `TIME` and `DTIME` formats are returned as integers representing the number of seconds in the time interval.
- By default, values of variables with date or datetime formats are returned as integers representing the number of seconds from October 14, 1582. You can specify to convert values of those variables to Python `datetime.datetime` objects with the *cvtDates* argument to the `Dataset` class. See the topic "spss.Dataset Class" on page 55 for more information.

Example: Accessing a Single Case

Case values are accessed by specifying the case number, starting with 0, as in:

```
oneCase = datasetObj.cases[0]
```

Case values are returned as a list where each element of the list is the value of the associated variable.

Example: Accessing a Single Value Within a Case

You can access the value for a single variable within a case by specifying the case number and the index of the variable (index values represent position in the active dataset, starting with 0 for the first variable in file order). The following gets the value of the variable with index 1 for case number 0.

```
oneValue = datasetObj.cases[0,1]
```

Note that *oneValue* is a list with a single element.

Example: Accessing a Range of Values

You can use the Python slice notation to specify ranges of cases and ranges of variables within a case. Values for multiple cases are returned as a list of elements, each of which is a list of values for a single case.

```
# Get the values for cases 0,1, and 2
data = datasetObj.cases[0:3]

# Get the values for variables with index values 0,1, and 2
# for case number 0
data = datasetObj.cases[0,0:3]

# Get the value for the variable with index 1 for case numbers 0,1, and 2
data = datasetObj.cases[0:3,1]

# Get the values for the variables with index values 1,2 and 3
# for case numbers 4,5, and 6
data = datasetObj.cases[4:7,1:4]
```

Example: Negative Index Values

Case indexing supports the use of negative indices, both for the case number and the variable index. The following gets the value of the second to last variable (in file order) for the last case.

```
value = datasetObj.cases[-1,-2]
```

Modifying case values. You can modify the values for a specific case or a range of cases, and you can set the value of a particular variable or a range of variables within those cases.

- Values of *None* are converted to system-missing for numeric variables and blanks for string variables.
- Values of numeric variables with a date or datetime format should be specified as Python `time.struct_time` or `datetime.datetime` objects, which are then converted to the appropriate IBM SPSS Statistics value. Values of variables with `TIME` and `DTIME` formats should be specified as the number of seconds in the time interval.

Example: Setting Values for a Single Case

Values for a single case are provided as a list or tuple of values. The first element corresponds to the first variable in file order, the second element corresponds to the second variable in file order, and so on. Case numbers start from 0.

```
datasetObj.cases[1] = [35,150,100,2110,19,2006,3,4]
```

Example: Setting a Single Value Within a Case

You can set the value for a single variable within a case by specifying the case number and the index of the variable (index values represent position in the active dataset, starting with 0 for the first variable in file order). The following sets the value of the variable with index 0 for case number 12 (case numbers start from 0).

```
datasetObj.cases[12,0] = 14
```

Example: Setting Ranges of Values

You can use the Python slice notation to specify ranges of cases and ranges of variables within a case. Values for multiple cases are specified as a list or tuple of elements, each of which is a list or tuple of values for a single case.

```
# Set the values for cases 0,1, and 2
datasetObj.cases[0:3] = ([172,'m',27,34500],[67,'f',32,32500],
                         [121,'f',37,23000])

# Set the values for variables with index values 5,6, and 7 for
# case number 34
datasetObj.cases[34,5:8] = [70,1,4]

# Set the value for the variable with index 5 for case numbers 0,1, and 2
datasetObj.cases[0:3,5] = [70,72,71]

# Set the values for the variables with index values 5 and 6 for
# case numbers 4,5, and 6
datasetObj.cases[4:7,5:7] = ([70,1],[71,2],[72,2])
```

Example: Negative Index Values

Case indexing supports the use of negative indices, both for the case number and the variable index. The following specifies the value of the second to last variable (in file order) for the last case.

```
datasetObj.cases[-1,-2] = 8
```

Deleting cases. You can delete a specified case from the `CaseList` object, which results in deleting that case from the associated dataset. For example:

```
del datasetObj.cases[0]
```

append Method: .append(case). *Appends a new case to the associated dataset and appends an element representing the case to the corresponding CaseList instance.* The argument *case* is a tuple or list specifying the case values. The first element in the tuple or list is the value for the first variable in file order, the second is the value of the second variable in file order and so on.

- The elements of *case* can be numeric or string values and must match the variable type of the associated variable. Values of *None* are converted to system-missing for numeric variables and blanks for string variables.
- Values of numeric variables with a date or datetime format should be specified as Python `time.struct_time` or `datetime.datetime` objects, which are then converted to the appropriate IBM SPSS Statistics value. Values of variables with `TIME` and `DTIME` formats should be specified as the number of seconds in the time interval.

Example

```
DATA LIST FREE/numvar (F2) strvar (A1).
BEGIN DATA.
1 a
END DATA.
BEGIN PROGRAM.
import spss
spss.StartDataStep()
datasetObj = spss.Dataset()
# Append a single case to the active dataset
datasetObj.cases.append([2,'b'])
spss.EndDataStep()
END PROGRAM.
```

insert Method: .insert(case, caseNumber). *Inserts a new case into the associated dataset and inserts an element representing the case into the corresponding CaseList instance.* The argument *case* is a tuple or list

specifying the case values. The first element in the tuple or list is the value for the first variable in file order, the second is the value of the second variable in file order and so on. The optional argument *caseNumber* specifies the location at which the case is inserted (case numbers start from 0) and can take on the values 0,1,...,*n* where *n* is the number of cases in the dataset. If *caseNumber* is omitted or equal to *n*, the case is appended.

- The elements of *case* can be numeric or string values and must match the variable type of the associated variable. Values of *None* are converted to system-missing for numeric variables and blanks for string variables.
- Values of numeric variables with a date or datetime format should be specified as Python `time.struct_time` or `datetime.datetime` objects, which are then converted to the appropriate IBM SPSS Statistics value. Values of variables with `TIME` and `DTIME` formats should be specified as the number of seconds in the time interval.

Example

```
DATA LIST FREE/numvar (F2) strvar (A1).
BEGIN DATA.
1 a
3 c
END DATA.
BEGIN PROGRAM.
import spss
spss.StartDataStep()
datasetObj = spss.Dataset()
# Insert a single case into the active dataset at case number 1
datasetObj.cases.insert([2,'b'],1)
spss.EndDataStep()
END PROGRAM.
```

VariableList Class

The `VariableList` class provides access to the variables in a dataset, allowing you to get and set properties of existing variables, as well as add new variables to the dataset. You get an instance of the `VariableList` class from the `varlist` property of the `Dataset` class, as in:

```
datasetObj = spss.Dataset('data1')
varListObj = datasetObj.varlist
```

The number of variables in a `VariableList` instance, which is also the number of variables in the associated dataset, is available using the `len` function, as in:

```
len(varListObj)
```

Note: An instance of the `VariableList` class can only be created within a data step, and cannot be used outside of the data step in which it was created. Data steps are initiated with the `spss.StartDataStep` function.

Looping through the variables in an instance of VariableList. You can loop through the variables in an instance of the `VariableList` class, obtaining a `Variable` object (representing the properties of a single variable) on each iteration. For example:

```
for var in datasetObj.varlist:
   print var.name
```

- On each iteration of the loop, *var* is an instance of the `Variable` class, representing a particular variable in the `VariableList` instance. The `Variable` class allows you to get and set variable properties, like the measurement level and missing values. See the topic "Variable Class" on page 66 for more information.

Accessing a variable by name or index. You can obtain a `Variable` object for a specified variable in the `VariableList` instance. The desired variable can be specified by name or index. For example:

```
#Get variable by name
varObj = datasetObj.varlist['salary']
#Get variable by index
varObj = datasetObj.varlist[5]
```

Deleting a variable. You can delete a specified variable from the `VariableList` instance, which results in deleting it from the associated dataset. The variable to be deleted can be specified by name or index. For example:

```
#Delete variable by name
del datasetObj.varlist['salary']
#Delete variable by index
del datasetObj.varlist[5]
```

append Method: .append(name,type). *Appends a new variable to the associated dataset and appends a corresponding Variable object to the associated VariableList instance.* The argument *name* specifies the variable name. The argument *type* is optional and specifies the variable type--numeric or string. The default is numeric.

- Numeric variables are specified by a value of 0 for the variable type. String variables are specified with a type equal to the defined length of the string (maximum of 32767).
- The properties of the new variable are set using the `Variable` object created by the `append` method. See the topic "Variable Class" for more information.

Example

```
DATA LIST FREE/numvar (F2).
BEGIN DATA.
1
END DATA.
BEGIN PROGRAM.
import spss
spss.StartDataStep()
datasetObj = spss.Dataset()
# Append a string variable of length 1 to the active dataset
datasetObj.varlist.append('strvar',1)
spss.EndDataStep()
END PROGRAM.
```

insert Method: .insert(name,type,index). *Inserts a new variable into the associated dataset and inserts a corresponding Variable object into the associated VariableList instance.* The argument *name* specifies the variable name. The optional argument *type* specifies the variable type--numeric or string. If *type* is omitted, the variable is numeric. The optional argument *index* specifies the position for the inserted variable and `Variable` object (the first position has an index value of 0) and can take on the values 0,1,...,*n* where *n* is the number of variables in the dataset. If *index* is omitted or equal to *n*, the variable is appended to the end of the list.

- Numeric variables are specified by a value of 0 for the variable type. String variables are specified with a type equal to the defined length of the string (maximum of 32767).
- The properties of the new variable are set using the `Variable` object created by the `insert` method. See the topic "Variable Class" for more information.

Example

```
DATA LIST FREE/var1 (F2) var3 (A1).
BEGIN DATA.
1 a
END DATA.
BEGIN PROGRAM.
import spss
spss.StartDataStep()
datasetObj = spss.Dataset()
# Insert a numeric variable at index position 1 in the active dataset
datasetObj.varlist.insert('var2',0,1)
spss.EndDataStep()
END PROGRAM.
```

Variable Class

The `Variable` class allows you to get and set the properties of a variable. Instances of the `Variable` class for each variable in the associated dataset are generated when the `VariableList` class is instantiated. In addition, the `append` and `insert` methods of a `VariableList` object create associated instances of the `Variable` class for appended and inserted variables. Specific variables can be accessed by name or index (index values represent position in the dataset, starting with 0 for the first variable in file order).

```
datasetObj = spss.Dataset('data1')
# Create a Variable object, specifying the variable by name
varObj = datasetObj.varlist['bdate']
# Create a Variable object, specifying the variable by index
varObj = datasetObj.varlist[3]
```

Note: An instance of the `Variable` class can only be created within a data step, and cannot be used outside of the data step in which it was created. Data steps are initiated with the `spss.StartDataStep` function.

alignment Property: The `alignment` property of a `Variable` object gets or sets the alignment of data values displayed in the Data Editor. It has no effect on the format of the variables or the display of the variables or values in other windows or printed results. The variable alignment is specified as an integer with one of the following values: 0 (left), 1 (right), 2 (center).

Example

```
varObj = datasetObj.varlist['gender']
#Get the variable alignment
align = varObj.alignment
#Set the variable alignment
varObj.alignment = 1
```

attributes Property: The `attributes` property of a `Variable` object gets or sets custom variable attributes. It can also be used to clear any custom attributes. The `attributes` property behaves like a Python dictionary in terms of getting, setting, and deleting values. A Python dictionary consists of a set of keys, each of which has an associated value that can be accessed simply by specifying the key. In the case of variable attributes, each key is the name of an attribute and the associated value is the value of the attribute, which can be a single value or a list or tuple of values. A list or tuple of values specifies an attribute array.

- When setting attributes, attribute names and values must be given as quoted strings.

Retrieving Variable Attributes. You retrieve custom variable attributes for a specified variable from the `attributes` property of the associated `Variable` object. You retrieve the value of a particular attribute by specifying the attribute name, as in:

```
varObj = datasetObj.varlist['gender']
attrValue = varObj.attributes['attrName']
```

Attribute values are always returned as a tuple.

You can iterate through the set of variable attributes using the `data` property, as in:

```
varObj = datasetObj.varlist['gender']
for attrName, attrValue in varObj.attributes.data.iteritems():
   print attrName, attrValue
```

Adding and Modifying Attributes. You can add new attributes and modify values of existing ones. For example:

```
varObj = datasetObj.varlist['age']
varObj.attributes['AnswerFormat'] = 'Fill-in'
```

- If the attribute *AnswerFormat* exists, its value is updated to 'Fill-in'. If the attribute *AnswerFormat* doesn't exist, it is added to any existing ones for the variable *age*.

Resetting Attributes. You can reset the attributes to a new specified set. For example:

```
varObj = datasetObj.varlist['gender']
varObj.attributes = {'DemographicVars':'1','Binary':'Yes'}
```

- You reset the attributes by setting the *attributes* property to a new Python dictionary. Any existing attributes for the variable are cleared and replaced with the specified set.

Deleting Attributes. You can delete a particular attribute or the entire set of attributes for a specified variable. For example:

```
varObj = datasetObj.varlist['gender']
#Delete the attribute Binary
del varObj.attributes['Binary']
#Delete all attributes
del varObj.attributes
```

columnWidth Property: The `columnWidth` property of a `Variable` object gets or sets the column width of data values displayed in the Data Editor. Changing the column width does not change the defined width of a variable. When setting the column width, the specified value must be a positive integer.

Example

```
varObj = datasetObj.varlist['prevexp']
#Get the column width
width = varObj.columnWidth
#Set the column width
varObj.columnWidth = 3
```

format Property: The `format` property of a `Variable` object gets or sets the display format of a variable.

Example

```
varObj = datasetObj.varlist['id']
#Get the variable format
format = varObj.format
#Set the variable format
varObj.format = (5,5,0)
```

- When getting the format, the returned value is a string consisting of a character portion (in upper case) that specifies the format type, followed by a numeric component that indicates the defined width, followed by a component that specifies the number of decimal positions and is only included for numeric formats. For example, A4 is a string format with a maximum width of four, and F8.2 is a standard numeric format with a display format of eight digits, including two decimal positions and a decimal indicator.
- When setting the format, you provide a tuple or list of three integers specifying the format type, width, and the number of decimal digits (for numeric formats) in that order. The width must include enough positions to accommodate any punctuation characters such as decimal points, commas, dollar signs, or date and time delimiters. If decimal digits do not apply, use 0 for the third element of the list or tuple. The available format types are listed in Appendix A, "Variable Format Types," on page 247.

Notes

- Allowable settings for decimal and width depend on the specified type. For a list of the minimum and maximum widths and maximum decimal places for commonly used format types, see Variable Types and Formats in the Universals section of the *Command Syntax Reference*, available in PDF from the Help menu and also integrated into the overall Help system.
- Setting the width for a string variable will not change the defined length of the string. If the specified value does not match the defined length, it is forced to be the defined length.

index Property: The `index` property of a `Variable` object gets the variable index. The index value represents position in the dataset starting with 0 for the first variable in file order.

Example

```
varObj = datasetObj.varlist['bdate']
index = varObj.index
```

label Property: The `label` property of a `Variable` object gets or sets the variable label.

Example

```
varObj = datasetObj.varlist['bdate']
#Get the variable label
label = varObj.label
#Set the variable label
varObj.label = 'Birth Date'
```

measurementLevel Property: The `measurementLevel` property of a `Variable` object gets or sets the measurement level of a variable. The measurement level is specified as a string. When setting the measurement level the allowed values are: `"NOMINAL"`, `"ORDINAL"`, and `"SCALE"`. When getting the measurement level the additional value `"UNKNOWN"` may be returned for numeric variables prior to the first data pass when the measurement level has not been explicitly set, such as data read from an external source or newly created variables. The measurement level for string variables is always known.

Example

```
varObj = datasetObj.varlist['minority']
#Get the measurement level
level = varObj.measurementLevel
#Set the measurement level
varObj.measurementLevel = "NOMINAL"
```

missingValues Property: The `missingValues` property of a `Variable` object gets or sets user-missing values. The missing values are specified as a tuple or list of four elements where the first element specifies the missing value type: 0,1,2, or 3 for that number of discrete values, -2 for a range of values, and -3 for a range of values and a single discrete value. The remaining three elements specify the missing values. When getting missing values, the result is returned as a tuple with this same structure.

- For string variables, returned values are right-padded to the defined width of the string variable.
- To specify *LO* and *HI* in missing value ranges, use the values returned by the spss.GetSPSSLowHigh function.

Table 4. Specifications for missing values

missingVals[0]	missingVals[1]	missingVals[2]	missingVals[3]
-3	Start point of range	End point of range	Discrete value
-2	Start point of range	End point of range	*None*
0	*None*	*None*	*None*
1	Discrete value	*None*	*None*
2	Discrete value	Discrete value	*None*
3	Discrete value	Discrete value	Discrete value

Examples

In the following examples, *varObj* is an instance of the `Variable` class.

Get the user-missing values.

```
missingVals = varObj.missingValues
```

Specify the discrete missing values 0 and 9 for a numeric variable.

```
varObj.missingValues = [2,0,9,None]
```

Specify the range of missing values 9–99 for a numeric variable.

```
varObj.missingValues = [-2,9,99,None]
```

Specify the range of missing values 9–99 and the discrete missing value 0 for a numeric variable.

```
varObj.missingValues = [-3,9,99,0]
```

Specify two missing values for a string variable.

```
varObj.missingValues = [2,' ','NA',None]
```

Clear all missing values

```
varObj.missingValues = [0,None,None,None]
```

name Property: The `name` property of a `Variable` object gets or sets the variable name.

Example

```
varObj = datasetObj.varlist['bdate']
#Get the variable name
name = varObj.name
#Set the variable name
varObj.name = 'birthdate'
```

role Property: The `role` property of a `Variable` object gets or sets the variable role. Valid values for getting and setting are the following strings: "Input", "Target", "Both", "None", "Partition" or "Split".

Example

```
varObj = datasetObj.varlist['var1']
#Get the variable role
role = varObj.role
#Set the variable role
varObj.role = 'Target'
```

type Property: The `type` property of a `Variable` object gets or sets the variable type--numeric or string. The variable type for numeric variables is 0. The variable type for string variables is an integer equal to the defined length of the string (maximum of 32767).

Example

```
varObj = datasetObj.varlist['strvar']
#Get the variable type
type = varObj.type
#Set the variable type to a string of length 10
varObj.type = 10
```

valueLabels Property: The `valueLabels` property of a `Variable` object gets or sets value labels. It can also be used to clear any value labels. The `valueLabels` property behaves like a Python dictionary in terms of getting, setting, and deleting values. A Python dictionary consists of a set of keys, each of which has an associated value that can be accessed simply by specifying the key. In the case of value labels, each key is a value and the associated value is the label.

- When setting value labels for string variables, values must be specified as quoted strings.

Retrieving Value Labels. You retrieve value labels for a specified variable from the `valueLabels` property of the associated `Variable` object. You retrieve the label for a particular value by specifying the value, as in the following, which retrieves the label for the value 1:

```
varObj = datasetObj.varlist['origin']
valLab = varObj.valueLabels[1]
```

You can iterate through the set of value labels for a variable using the `data` property, as in:

```
varObj = datasetObj.varlist['origin']
for val, valLab in varObj.valueLabels.data.iteritems():
   print val, valLab
```

Adding and Modifying Value Labels. You can add new value labels and modify existing ones. For example:

```
varObj = datasetObj.varlist['origin']
varObj.valueLabels[4] = 'Korean'
```

- If a label for the value 4 exists, its value is updated to 'Korean'. If a label for the value 4 doesn't exist, it is added to any existing value labels for the variable *origin*.

Resetting Value Labels. You can reset the value labels to a new specified set. For example:

```
varObj = datasetObj.varlist['origin']
varObj.valueLabels = {1:'American',2:'Japanese',3:'European',
                      4:'Korean',5:'Chinese'}
```

- You reset the value labels by setting the *valueLabels* property to a new Python dictionary. Any existing value labels for the variable are cleared and replaced with the specified set.

Deleting Value Labels. You can delete a particular value label or the entire set of value labels for a specified variable. For example:

```
varObj = datasetObj.varlist['origin']
#Delete the value label for the value 1
del varObj.valueLabels[1]
#Delete all value labels
del varObj.valueLabels
```

spss.DataStep Class

The `DataStep` class implicitly starts and ends a data step without the need to explicitly call `StartDataStep` and `EndDataStep`. In addition, it executes any pending transformations, eliminating the need to check for them prior to starting a data step. The `DataStep` class is designed to be used with the Python `with` statement as shown in the following example.

Example

```
BEGIN PROGRAM.
import spss
with spss.DataStep():
   datasetObj = spss.Dataset(name=None)
   datasetObj.varlist.append('numvar')
   datasetObj.varlist.append('strvar',1)
   datasetObj.varlist['numvar'].label = 'Sample numeric variable'
   datasetObj.varlist['strvar'].label = 'Sample string variable'
   datasetObj.cases.append([1,'a'])
   datasetObj.cases.append([2,'b'])
END PROGRAM.
```

- `with spss.DataStep():` initiates a block of code associated with a data step. The data step is implicitly started after executing any pending transformations. All code associated with the data step should reside in the block as shown here. When the block completes, the data step is implicitly ended.

spss.DeleteXPathHandle Function

spss.DeleteXPathHandle(handle). *Deletes the XPath dictionary DOM or output DOM with the specified handle name.* The argument is a handle name that was defined with a previous `spss.CreateXPathDictionary` function or an IBM SPSS Statistics `OMS` command.

Example

```
handle = 'demo'
spss.DeleteXPathHandle(handle)
```

spss.EndDataStep Function

spss.EndDataStep(). *Signals the end of a data step.*

- `EndDataStep` must be called to end a data step initiated with `StartDataStep`.

For an example that uses `EndDataStep`, see the topic on the Dataset class.

spss.EndProcedure Function

spss.EndProcedure(). *Signals the end of pivot table or text block output.*

- `spss.EndProcedure` must be called to end output initiated with spss.StartProcedure.

spss.EvaluateXPath Function

spss.EvaluateXPath(handle,context,xpath). Evaluates an XPath expression against a specified XPath DOM and returns the result as a list. The argument `handle` specifies the particular XPath DOM and must be a valid handle name that is defined by a previous `spss.CreateXPathDictionary` function or IBM SPSS Statistics `OMS` command. The argument `context` defines the XPath context for the expression and must be set to `"/dictionary"` for a dictionary DOM or `"/outputTree"` for an output XML DOM created by the `OMS` command. The argument `xpath` specifies the remainder of the XPath expression and must be quoted.

Example

```
#retrieve a list of all variable names for the active dataset.
handle='demo'
spss.CreateXPathDictionary(handle)
context = "/dictionary"
xpath = "variable/@name"
varnames = spss.EvaluateXPath(handle,context,xpath)
```

Example

```
*Use OMS and a Python program to determine the number of uniques values
  for a specific variable.
OMS SELECT TABLES
  /IF COMMANDs=['Frequencies'] SUBTYPES=['Frequencies']
 /DESTINATION FORMAT=OXML XMLWORKSPACE='freq_table'.
FREQUENCIES VARIABLES=var1.
OMSEND.

BEGIN PROGRAM.
import spss
handle='freq_table'
context="/outputTree"
#get rows that are totals by looking for varName attribute
#use the group element to skip split file category text attributes
xpath="//group/category[@varName]/@text"
values=spss.EvaluateXPath(handle,context,xpath)
#the "set" of values is the list of unique values
#and the length of that set is the number of unique values
uniqueValuesCount=len(set(values))
END PROGRAM.
```

Note: In the IBM SPSS Statistics documentation, XPath examples for the `OMS` command use a namespace prefix in front of each element name (the prefix `oms:` is used in the OMS examples). Namespace prefixes are not valid for `EvaluateXPath`.

spss.GetCaseCount Function

spss.GetCaseCount(). *Returns the number of cases (rows) in the active dataset.* Returns a value of -1 if the case count is not known.

Example

```
#build SAMPLE syntax of the general form:
#SAMPLE [NCases] FROM [TotalCases]
#Where Ncases = 10% truncated to integer
TotalCases=spss.GetCaseCount()
NCases=int(TotalCases/10)
command1="SAMPLE " + str(NCases) + " FROM " + str(TotalCases) +  "."
command2="Execute."
spss.Submit([command1, command2])
```

spss.GetDataFileAttributeNames Function

spss.GetDataFileAttributeNames(). *Returns the names of any datafile attributes, as a tuple, for the active dataset.*

Example

```
import spss
fileattrs = spss.GetDataFileAttributeNames()
```

spss.GetDataFileAttributes Function

spss.GetDataFileAttributes(attrName). *Returns the attribute values, as a tuple, for the specified datafile attribute.* The argument *attrName* is a string that specifies the name of the attribute--for instance, a name returned by `GetDataFileAttributeNames`.

Example

```
# Build a Python dictionary of the datafile attributes
import spss
attrDict = {}
for name in spss.GetDataFileAttributeNames():
   attrDict[name] = spss.GetDataFileAttributes(name)
```

spss.GetDatasets Function

spss.GetDatasets(). *Returns a list of the available Dataset objects.* Each object in the list is an instance of the `Dataset` class. The `GetDatasets` function is intended for use within a data step or a `StartProcedure-EndProcedure` block and will return an empty list if used elsewhere. Data steps are initiated with the `spss.StartDataStep` function and are used to create and manage multiple datasets.

Example

```
import spss
spss.StartDataStep()
# Create a Dataset object for the active dataset
datasetObj1 = spss.Dataset()
# Create a new and empty dataset
datasetObj2 = spss.Dataset(name=None)
datasetNames = [item.name for item in spss.GetDatasets()]
spss.EndDataStep()
```

spss.GetDefaultPlugInVersion Function

spss.GetDefaultPlugInVersion(). *Returns the default version of the IBM SPSS Statistics - Integration Plug-in for Python used for Python programs.* The result is a string specifying a version--for example, `"spss170"` for version 17.0--and is useful when working with multiple versions of the plug-in on a given machine (see Note below). You can change the default using the spss.SetDefaultPlugInVersion function.

Note: For version 22 and higher, this function always returns the version of IBM SPSS Statistics from which it was called.

Example

```
import spss
version = spss.GetDefaultPlugInVersion()
```

Note: The functions for managing multiple versions of the plug-in (`spss.GetDefaultPlugInVersion`, `spss.SetDefaultPlugInVersion`, and `spss.ShowInstalledPlugInVersions`) operate within a given Python version, not across Python versions. For example, if you are driving IBM SPSS Statistics from a Python IDE installed for Python 2.6 then you can view and control the versions of the plug-in installed for Python 2.6.

spss.GetFileHandles Function

spss.GetFileHandles(). *Returns a list of currently defined file handles*. Each item in the list consists of the following three elements: the name of the file handle; the path associated with the file handle; and the encoding, if any, specified for the file handle. File handles are created with the `FILE HANDLE` command.

spss.GetHandleList Function

spss.GetHandleList(). *Returns a list of currently defined dictionary and output XPath DOMs available for use with spss.EvaluateXpath*.

spss.GetImage Function

spss.GetImage(handle,imagename). *Retrieves an image associated with an output XPath DOM*. The argument *handle* specifies the particular XPath DOM and must be a valid handle name defined by a previous IBM SPSS Statistics `OMS` command. The argument *imagename* is the filename associated with the image in the OXML output--specifically, the value of the `imageFile` attribute of the `chart`, `modelView` or `treeView` element associated with the image.

The returned value is a tuple with 3 elements. The first element is the binary image. The second element is the amount of memory required for the image. The third element is a string specifying the image type: “PNG”, “JPG”, “BMP”.

Example

```
OMS
  /SELECT CHARTS
  /IF COMMANDS=['Frequencies']
  /DESTINATION FORMAT=OXML IMAGES=YES
   CHARTFORMAT=IMAGE IMAGEROOT='myimages' IMAGEFORMAT=JPG XMLWORKSPACE='demo'.

FREQUENCIES VARIABLES=jobcat
  /BARCHART PERCENT
  /ORDER=ANALYSIS.

OMSEND.

BEGIN PROGRAM.
import spss
imagename=spss.EvaluateXPath('demo','/outputTree',
 '//command[@command="Frequencies"]/chartTitle[@text="Bar Chart"]/chart/@imageFile')[0]
image = spss.GetImage('demo',imagename)
f = file('/temp/myimage.jpg','wb')
f.truncate(image[1])
f.write(image[0])
f.close()
spss.DeleteXPathHandle('demo')
END PROGRAM.
```

- The `OMS` command routes output from the `FREQUENCIES` command to an output XPath DOM with the handle name of *demo*.
- To route images along with the OXML output, the `IMAGES` keyword on the `DESTINATION` subcommand (of the `OMS` command) must be set to `YES`, and the `CHARTFORMAT`, `MODELFORMAT`, or `TREEFORMAT` keyword must be set to `IMAGE`.
- The `spss.EvaluateXPath` function is used to retrieve the name of the image associated with the bar chart output from the `FREQUENCIES` command. In the present example, the value returned by `spss.EvaluateXPath` is a list with a single element, which is then stored to the variable *imagename*.
- The `spss.GetImage` function retrieves the image, which is then written to an external file.

spss.GetLastErrorLevel and spss.GetLastErrorMessage Functions

spss.GetLastErrorLevel(). *Returns a number corresponding to an error in the preceding Python Integration Package for IBM SPSS Statistics function.*

- For the `spss.Submit` function, it returns the maximum IBM SPSS Statistics error level for the submitted command syntax. IBM SPSS Statistics error levels range from 1 to 5. An error level of 3 or higher causes an exception in Python.
- For other functions, it returns an error code with a value greater than 5.
- Error codes from 6 to 99 are from the IBM SPSS Statistics XD API.
- Error codes from 1000 to 1064 are from the Python Integration Package.

IBM SPSS Statistics error levels (return codes), their meanings, and any associated behaviors are shown in the following table.

Table 5. IBM SPSS Statistics error levels.

Value	Definition	Behavior
0	None	Command runs
1	Comment	Command runs
2	Warning	Command runs
3	Serious error	Command does not run, subsequent commands are processed
4	Fatal error	Command does not run, subsequent commands are not processed, and the current job terminates

Table 5. IBM SPSS Statistics error levels (continued).

Value	Definition	Behavior
5	Catastrophic error	Command does not run, subsequent commands are not processed, and the IBM SPSS Statistics processor terminates

spss.GetLastErrorMessage(). *Returns a text message corresponding to an error in the preceding Python Integration Package for IBM SPSS Statistics function.*

- For the `spss.Submit` function, it returns text associated with the highest level error for the submitted command syntax.
- For other functions in the Python Integration Package, it returns the error message text from the IBM SPSS Statistics XD API or from Python.

Example

```
DATA LIST FREE/var1 var2.
BEGIN DATA
1 2 3 4
END DATA.
BEGIN PROGRAM.
try:
  spss.Submit("""
COMPUTE newvar=var1*10.
COMPUTE badvar=nonvar/4.
FREQUENCIES VARIABLES=ALL.
""")
except:
  errorLevel=str(spss.GetLastErrorLevel())
  errorMsg=spss.GetLastErrorMessage()
  print("Error level " + errorLevel + ": " + errorMsg)
  print("At least one command did not run.")
END PROGRAM.
```

- The first `COMPUTE` command and the `FREQUENCIES` command will run without errors, generating error values of 0.
- The second `COMPUTE` command will generate a level 3 error, triggering the exception handling in the `except` clause.

spss.GetMultiResponseSetNames Function

spss.GetMultiResponseSetNames(). *Returns the names of any multiple response sets for the active dataset.*

Example

```
import spss
names = spss.GetMultiResponseSetNames()
```

spss.GetMultiResponseSet Function

spss.GetMultiResponseSet(mrsetName). *Returns the details of the specified multiple response set.* The argument *mrsetName* is a string that specifies the name of the multiple response set--for instance, a name returned by `GetMultiResponseSetNames`.

- The result is a tuple of 5 elements. The first element is the label, if any, for the set. The second element specifies the variable coding--'Categories' or 'Dichotomies'. The third element specifies the counted value and only applies to multiple dichotomy sets. The fourth element specifies the data type--'Numeric' or 'String'. The fifth element is a list of the elementary variables that define the set.

Example

```
# Build a Python dictionary of the multiple response sets
import spss
dict = {}
for name in spss.GetMultiResponseSetNames():
  dict[name]=spss.GetMultiResponseSet(name)
```

spss.GetOMSTagList Function

spss.GetOMSTagList(). *Returns a list of tags associated with any active OMS requests*. Each OMS request has a tag which identifies the request. The tag is specified with the TAG subcommand of the OMS command, or automatically generated if not specified.

spss.GetSetting Function

spss.GetSetting(setting,option). *Returns the value of an options setting*. Specifically, this function returns values for options that can be set with the SET command.

- The argument *setting* is a string specifying the name of the subcommand (of the SET command), whose value is desired--for example "OLANG". The case of the specified string is ignored.

  Note: GetSetting does not support retrieving the value of the MTINDEX subcommand of the SET command.
- The argument *option* is a string specifying an option associated with the value of the *setting* argument. It only applies to the MIOUTPUT subcommand of SET, for which there is a separate setting for each of the keywords "OBSERVED", "IMPUTED", "POOLED", and "DIAGNOSTICS". When *setting* equals "MIOUTPUT", *option* can be set to any of those four keywords to obtain the associated value of the keyword--'Yes' or 'No'. The case of the string specified for *option* is ignored.

spss.GetSplitVariableNames Function

spss.GetSplitVariableNames(). *Returns the names of the split variables, if any, in the active dataset*.

Example

```
import spss
splitvars = spss.GetSplitVariableNames()
```

spss.GetSPSSLocale Function

spss.GetSPSSLocale(). . *Returns the current IBM SPSS Statistics locale.*

Example

```
import spss
locale = spss.GetSPSSLocale()
```

spss.GetSPSSLowHigh Function

spss.GetSPSSLowHigh(). *Returns the values IBM SPSS Statistics uses for LO and HI as a tuple of two values*. The first element in the tuple is the value for *LO* and the second is the value for *HI*. These values can used to specify missing value ranges for new numeric variables with the SetVarNMissingValues method.

Example

```
import spss
spsslow, spsshigh = spss.GetSPSSLowHigh()
```

spss.GetVarAttributeNames Function

spss.GetVarAttributeNames(index). *Returns the names of any variable attributes, as a tuple, for the variable in the active dataset indicated by the index value.* The argument is the index value. Index values represent position in the active dataset, starting with 0 for the first variable in file order.

Example

```
#Create a list of variables that have a specified attribute
import spss
varList=[]
attribute='demographicvars'
for i in range(spss.GetVariableCount()):
   if (attribute in spss.GetVarAttributeNames(i)):
      varList.append(spss.GetVariableName(i))
if varList:
```

```
   print "Variables with attribute " + attribute + ":"
   print '\n'.join(varList)
else:
   print "No variables have the attribute " + attribute
```

spss.GetVarAttributes Function

spss.GetVarAttributes(index,attrName). *Returns the attribute values, as a tuple, for the specified attribute of the variable in the active dataset indicated by the index value.* The argument *index* is the index value. Index values represent position in the active dataset, starting with 0 for the first variable in file order. The argument *attrName* is a string that specifies the name of the attribute--for instance, a name returned by `GetVarAttributeNames`.

Example

```
#Create a list of variables whose attribute array contains
#a specified value
import spss
varList=[]
attrName='demographicvartypes'
attrVal='2'
for i in range(spss.GetVariableCount()):
   try:
      if(attrVal in spss.GetVarAttributes(i,attrName)):
         varList.append(spss.GetVariableName(i))
   except:
      pass
if varList:
   print "Variables with attribute value " + attrVal + \
         " for attribute " + attrName + ":"
   print '\n'.join(varList)
else:
   print "No variables have the attribute value " + attrVal + \
         " for attribute " + attrName
```

spss.GetVariableCount Function

spss.GetVariableCount(). *Returns the number of variables in the active dataset.*

Example

```
#build a list of all variables by using the value of
#spssGetVariableCount to set the number of for loop interations
varcount=spss.GetVariableCount()
varlist=[]
for i in xrange(varcount):
    varlist.append(spss.GetVariableName(i))
```

spss.GetVariableFormat Function

GetVariableFormat(index). *Returns a string containing the display format for the variable in the active dataset indicated by the index value.* The argument is the index value. Index values represent position in the active dataset, starting with 0 for the first variable in file order.

- The character portion of the format string is always returned in all upper case.
- Each format string contains a numeric component after the format name that indicates the defined width, and optionally, the number of decimal positions for numeric formats. For example, A4 is a string format with a maximum width of four bytes, and F8.2 is a standard numeric format with a display format of eight digits, including two decimal positions and a decimal indicator. The supported format types are listed in Variable Format Types (the type code shown in the table does not apply to the `GetVariableFormat` function).

Example

```
DATA LIST FREE
  /numvar (F4) timevar1 (TIME5) stringvar (A2) timevar2 (TIME12.2).
BEGIN DATA
1 10:05 a 11:15:33.27
END DATA.

BEGIN PROGRAM.
import spss
```

```
#create a list of all formats and a list of time format variables
varcount=spss.GetVariableCount()
formatList=[]
timeVarList=[]
for i in xrange(varcount):
  formatList.append(spss.GetVariableFormat(i))
  #check to see if it's a time format
  if spss.GetVariableFormat(i).find("TIME")==0:
     timeVarList.append(spss.GetVariableName(i))
print formatList
print timeVarList
END PROGRAM.
```

spss.GetVariableLabel Function

spss.GetVariableLabel(index). *Returns a character string containing the variable label for the variable in the active dataset indicated by the index value.* The argument is the index value. Index values represent position in the active dataset, starting with 0 for the first variable in file order. If the variable does not have a defined variable label, a null string is returned.

Example

```
#create a list of all variable labels
varcount=spss.GetVariableCount()
labellist=[]
for i in xrange(varcount):
  labellist.append(spss.GetVariableLabel(i))
```

spss.GetVariableMeasurementLevel Function

spss.GetVariableMeasurementLevel(index). *Returns a string value that indicates the measurement level for the variable in the active dataset indicated by the index value.* The argument is the index value. Index values represent position in the active dataset, starting with 0 for the first variable in file order. The value returned can be: `"nominal"`, `"ordinal"`, `"scale"`, or `"unknown"`.

- "Unknown" occurs only for numeric variables prior to the first data pass when the measurement level has not been explicitly set, such as data read from an external source or newly created variables. The measurement level for string variables is always known.

Example

```
#build a string containing  scale variable names
varcount=spss.GetVariableCount()
ScaleVarList=''
for i in xrange(varcount):
  if spss.GetVariableMeasurementLevel(i)=="scale":
    ScaleVarList=ScaleVarList + " " + spss.GetVariableName(i)
```

spss.GetVariableName Function

spss.GetVariableName(index). *Returns a character string containing the variable name for the variable in the active dataset indicated by the index value.* The argument is the index value. Index values represent position in the active dataset, starting with 0 for the first variable in file order.

Example

```
#get names of first and last variables in the file
#last variable is index value N-1 because index values start at 0
firstVar=spss.GetVariableName(0)
lastVar=spss.GetVariableName(spss.GetVariableCount()-1)
print firstVar, lastVar
#sort the data file in alphabetic order of variable names
varlist=[]
varcount=spss.GetVariableCount()
for i in xrange(varcount):
  varlist.append(spss.GetVariableName(i))
sortedlist=' '.join(sorted(varlist))
spss.Submit(
  ["ADD FILES FILE=* /KEEP ",sortedlist, ".", "EXECUTE."])
```

spss.GetVariableRole Function

spss.GetVariableRole(index). *Returns a character string containing the role for the variable in the active dataset indicated by the index value.* The argument is the index value. Index values represent position in the active dataset, starting with 0 for the first variable in file order. The value returned is one of the following strings: "Input", "Target", "Both", "None", "Partition" or "Split".

Example

```
#Find the variable(s) with the role of "Target"
targets=[]
for i in range(spss.GetVariableCount()):
   if spss.GetVariableRole(i)=="Target":
      targets.append(spss.GetVariableName(i))
if len(targets):
   print "Target variables:"
   for i in range(len(targets)):
      print targets[i]
else:
   print "No target variables found"
```

spss.GetVariableType Function

spss.GetVariableType(index). *Returns 0 for numeric variables or the defined length for string variables for the variable in the active dataset indicated by the index value.* The argument is the index value. Index values represent position in the active dataset, starting with 0 for the first variable in file order.

Example

```
#create separate strings of numeric and string variables
numericvars=''
stringvars=''
varcount=spss.GetVariableCount()
for i in xrange(varcount):
  if spss.GetVariableType(i) > 0:
    stringvars=stringvars + " " + spss.GetVariableName(i)
  else:
    numericvars=numericvars + " " + spss.GetVariableName(i)
```

spss.GetVarMissingValues Function

spss.GetVarMissingValues(index). *Returns the user-missing values for the variable in the active dataset indicated by the index value.* The argument is the index value. Index values represent position in the active dataset, starting with 0 for the first variable in file order.

- The result is a tuple of four elements where the first element specifies the missing value type: 0 for discrete values, 1 for a range of values, and 2 for a range of values and a single discrete value. The remaining three elements in the result specify the missing values.
- For string variables, the missing value type is always 0 since only discrete missing values are allowed. Returned values are right-padded to the defined width of the string variable.
- If there are no missing values, the result is `(0,None,None,None)`.

Table 6. Structure of the result

tuple[0]	tuple[1]	tuple[2]	tuple[3]
0	Discrete value or *None*	Discrete value or *None*	Discrete value or *None*
1	Start point of range	End point of range	*None*
2	Start point of range	End point of range	Discrete value

Example

```
#List all variables without user-missing values
nomissList=[]
for i in range(spss.GetVariableCount()):
   missing=spss.GetVarMissingValues(i)
   if (missing[0]==0 and missing[1]==None):
      nomissList.append(spss.GetVariableName(i))
if nomissList:
```

```
   print "Variables without user-missing values:"
   print '\n'.join(nomissList)
else:
   print "All variables have user-missing values"
```

spss.GetWeightVar Function

spss.GetWeightVar(). *Returns the name of the weight variable, or None if unweighted.*

Example

```
import spss
weightVar = spss.GetWeightVar()
```

spss.GetXmlUtf16 Function

spss.GetXmlUtf16(handle, filespec). *Writes the XML for the specified handle (dictionary or output XML) to a file or returns the XML if no filename is specified.* When writing and debugging XPath expressions, it is often useful to have a sample file that shows the XML structure. This function is particularly useful for dictionary DOMs, since there are not any alternative methods for writing and viewing the XML structure. (For output XML, the `OMS` command can also write XML to a file.) You can also use this function to retrieve the XML for a specified handle, enabling you to process it with third-party utilities like XML parsers.

Example

```
handle = "activedataset"
spss.CreateXPathDictionary(handle)
spss.GetXmlUtf16(handle,'/temp/temp.xml')
```

spss.HasCursor Function

spss.HasCursor(). *Returns an integer indicating whether there is an open cursor.* A value of 0 indicates there is no open cursor, and a value of 1 indicates there is an open cursor. Cursors allow you to read data from the active dataset, create new variables in the active dataset, and append cases to the active dataset. For information on working with cursors, see the topic on the Cursor class .

spss.IsActive Function

spss.IsActive(datasetObj). *Indicates whether the specified dataset is the active one.* The result is Boolean—*True* if the specified dataset is active, *False* otherwise. The argument must be an instance of the `Dataset` class. The `IsActive` function is intended for use within a data step. Data steps are initiated with the `spss.StartDataStep` function and are used to create and manage multiple datasets.

Example

```
import spss
spss.StartDataStep()
datasetObj = spss.Dataset(name="file1")
if not spss.IsActive(datasetObj):
   spss.SetActive(datasetObj)
spss.EndDataStep()
```

spss.IsDistributedMode Function

spss.IsDistributedMode(). *Indicates whether IBM SPSS Statistics is in distributed mode.* The result is Boolean—*True* if SPSS Statistics is in distributed mode, *False* otherwise. The `IsDistributedMode` function always returns False when an external Python process is controlling the SPSS Statistics backend.

Example

```
import spss
if spss.IsDistributedMode():
   print "SPSS Statistics is in distributed mode"
else:
   print "SPSS Statistics is not in distributed mode"
```

spss.IsOutputOn Function

spss.IsOutputOn(). *Returns the status of IBM SPSS Statistics output display in Python.* The result is Boolean—*True* if output display is on in Python, *False* if it is off. See the topic "spss.SetOutput Function" on page 83 for more information.

Example

```
import spss
spss.SetOutput("on")
if spss.IsOutputOn():
    print "The current IBM SPSS Statistics output setting is 'on'."
else:
    print "The current IBM SPSS Statistics output setting is 'off'."
```

spss.Procedure Class

spss.Procedure(procName,omsIdentifier). The `Procedure` class implicitly starts and ends a user procedure without the need to explicitly call `StartProcedure` and `EndProcedure`.

- The argument *procName* is a string and is the name that appears in the outline pane of the Viewer associated with the output from the procedure. It has the same specifications as the *procedureName* argument to the `StartProcedure` function.
- The optional argument *omsIdentifier* specifies the OMS identifier for output from this procedure and has the same specifications as the *omsIdentifier* argument to the `StartProcedure` function. *omsIdentifier* is only necessary when creating procedures with localized output so that the procedure name can be localized but not the OMS identifier. See the topic "Localizing Output from Python Programs" on page 11 for more information.

The `Procedure` class is designed to be used with the Python `with` statement as shown in the following example.

Example

```
BEGIN PROGRAM.
import spss
with spss.Procedure("demoProc"):
   table = spss.BasePivotTable("Table Title",
                               "OMS table subtype")

   table.SimplePivotTable(rowdim = "row dimension",
                          rowlabels = ["first row","second row"],
                          coldim = "column dimension",
                          collabels = ["first column","second column"],
                          cells = [11,12,21,22])
END PROGRAM.
```

- `with spss.Procedure("demoProc"):` initiates a block of code associated with a procedure named *demoProc* and implicitly starts the procedure. All code associated with the procedure should reside in the block as shown here. When the block completes, the procedure is implicitly ended.

spss.PyInvokeSpss.IsUTF8mode Function

spss.PyInvokeSpss.IsUTF8mode(). *Indicates whether IBM SPSS Statistics is running in Unicode mode or code page mode.* The result is 1 if IBM SPSS Statistics is in Unicode mode, 0 if IBM SPSS Statistics is in code page mode.

Example

```
import spss
isUTF8 = spss.PyInvokeSpss.IsUTF8mode()
if isUTF8==1:
  print "IBM SPSS Statistics is running in Unicode mode."
else:
  print "IBM SPSS Statistics is running in code page mode."
```

spss.PyInvokeSpss.IsXDriven Function

spss.PyInvokeSpss.IsXDriven(). *Checks to see how the IBM SPSS Statistics backend is being run.* The result is 1 if Python is controlling the IBM SPSS Statistics backend or 0 if IBM SPSS Statistics is controlling the backend.

Example

```
import spss
spss.Submit("""
GET FILE
 '/examples/data/employee data.sav'.
""")
isxd = spss.PyInvokeSpss.IsXDriven()
if isxd==1:
  print "Python is driving IBM SPSS Statistics."
else:
  print "IBM SPSS Statistics is driving Python."
```

spss.SetActive Function

spss.SetActive(datasetObj). *Sets the specified dataset as the active one.* The argument must be an instance of the `Dataset` class. The `SetActive` function can only be used within a data step. Data steps are initiated with the `spss.StartDataStep` function and are used to create and manage multiple datasets.

Example

```
# Set a newly created dataset to be active
spss.StartDataStep()
ds1 = spss.Dataset(name=None)
spss.SetActive(ds1)
spss.EndDataStep()
```

spss.SetDefaultPlugInVersion Function

Note: This function is deprecated in release 22. See the topic "Working with Multiple Versions of IBM SPSS Statistics" on page 8 for more information.

spss.SetDefaultPlugInVersion(value). *Sets the default version of the IBM SPSS Statistics - Integration Plug-in for Python used for Python programs.* This function is useful when working with multiple versions of the plug-in on a given machine (see Note below). The value of the argument is a quoted string or an integer specifying a plug-in version--for example, `"spss160"` or `160` for version 16.0. The strings representing the installed versions of the plug-in are available from the function spss.ShowInstalledPlugInVersions.

- For versions 17.0 to 21.0, `SetDefaultPlugInVersion` also sets the default version of the IBM SPSS Statistics - Integration Plug-in for Python used for Python scripts (Python code that utilizes the `SpssClient` module).

Example

```
import spss
spss.SetDefaultPlugInVersion("spss160")
```

Note: The functions for managing multiple versions of the plug-in (`spss.GetDefaultPlugInVersion`, `spss.SetDefaultPlugInVersion`, and `spss.ShowInstalledPlugInVersions`) operate within a given Python version, not across Python versions. For example, if you are driving IBM SPSS Statistics from a Python IDE installed for Python 2.6 then you can view and control the versions of the plug-in installed for Python 2.6.

spss.SetMacroValue Function

spss.SetMacroValue(name, value). *Defines a macro variable that can be used outside a program block in command syntax.* The first argument is the macro name, and the second argument is the macro value. Both arguments must resolve to strings.

- The argument specifying the macro value cannot contain the characters \ or ^ unless they are contained within a quoted string.

Example

```
DATA LIST FREE /var1 var2 var3 var4.
begin data
1 2 3 4
end data.
VARIABLE LEVEL var1 var3 (scale) var2 var4 (nominal).

BEGIN PROGRAM.
import spss
macroValue=[]
macroName="!NominalVars"
varcount=spss.GetVariableCount()
for i in xrange(varcount):
  if spss.GetVariableMeasurementLevel(i)=="nominal":
    macroValue.append(spss.GetVariableName(i))
spss.SetMacroValue(macroName, macroValue)
END PROGRAM.
FREQUENCIES VARIABLES=!NominalVars.
```

spss.SetOutput Function

spss.SetOutput("value"). *Controls the display of IBM SPSS Statistics output in Python when running IBM SPSS Statistics from Python.* Output is displayed as standard output, and charts and classification trees are not included. When running Python from IBM SPSS Statistics within program blocks (`BEGIN PROGRAM-END PROGRAM`), this function has no effect. The value of the argument is a quoted string:

- **"on".** Display IBM SPSS Statistics output in Python.
- **"off".** Do not display IBM SPSS Statistics output in Python.

Example

```
import spss
spss.SetOutput("on")
```

spss.SetOutputLanguage Function

spss.SetOutputLanguage("language"). *Sets the language that is used in IBM SPSS Statistics output.* The argument is a quoted string specifying one of the following languages: "English", "French", "German", "Italian", "Japanese", "Korean", "Polish", "Russian", "SChinese" (Simplified Chinese), "Spanish", "TChinese" (Traditional Chinese), or "BPortugu" (Brazilian Portuguese). The setting does not apply to simple text output.

Example

```
import spss
spss.SetOutputLanguage("German")
```

spss.ShowInstalledPlugInVersions Function

Note: This function is deprecated in release 22. See the topic "Working with Multiple Versions of IBM SPSS Statistics" on page 8 for more information.

spss.ShowInstalledPlugInVersions(). *Displays the installed versions of the IBM SPSS Statistics - Integration Plug-in for Python.* This function displays the installed versions of the plug-in--for example, `"spss200"` and `"spss210"` for versions 20.0 and 21.0--and is useful when working with multiple versions of the plug-in on a given machine (see Note below). Use an identifier from this list as the argument to the spss.SetDefaultPlugInVersion function.

Example

```
import spss
spss.ShowInstalledPlugInVersions()
```

Note: The functions for managing multiple versions of the plug-in (`spss.GetDefaultPlugInVersion`, `spss.SetDefaultPlugInVersion`, and `spss.ShowInstalledPlugInVersions`) operate within a given Python version, not across Python versions. For example, if you are driving IBM SPSS Statistics from a Python IDE installed for Python 2.6 then you can view and control the versions of the plug-in installed for Python 2.6.

spss.SplitChange Function

spss.SplitChange(outputName). *Used to process splits when creating pivot tables from data that have splits.* The argument *outputName* is the name associated with the output, as specified on the associated call to the `StartProcedure` function. See the topic "spss.StartProcedure Function" on page 85 for more information.

- This function should be called after detecting a split and reading the first case of the new split. It should also be called after reading the first case in the active dataset.
- The creation of pivot table output does not support operations involving data in different split groups. When working with splits, each split should be treated as a separate set of data.
- Use the `SPLIT FILE` command to control whether split-file groups will be displayed in the same table or in separate tables. The `SPLIT FILE` command should be called before the `StartProcedure` function.
- The IsEndSplit method from the `Cursor` class is used to detect a split change.

Example

In this example, a split is created and separate averages are calculated for the split groups. Results for different split groups are shown in a single pivot table. In order to understand the example, you will need to be familiar with creating pivot tables using the BasePivotTable class and creating output with the spss.StartProcedure function.

```
import spss
from spss import CellText
from spss import FormatSpec

spss.Submit(r"""
GET FILE="/examples/data/employee data.sav".
SORT CASES BY GENDER.
SPLIT FILE LAYERED BY GENDER.
""")

spss.StartProcedure("spss.com.demo")

table = spss.BasePivotTable("Table Title","OMS table subtype")
table.Append(spss.Dimension.Place.row,"Minority Classification")
table.Append(spss.Dimension.Place.column,"coldim",hideName=True)

cur=spss.Cursor()
salary = 0; salarym = 0; n = 0; m = 0
minorityIndex = 9
salaryIndex = 5

row = cur.fetchone()
spss.SplitChange("spss.com.demo")
while True:
   if cur.IsEndSplit():
      if n>0:
         salary=salary/n
      if m>0:
         salarym=salarym/m
      # Populate the pivot table with values for the previous split group
      table[(CellText.String("No"),CellText.String("Average Salary"))] = \
             CellText.Number(salary,FormatSpec.Count)
      table[(CellText.String("Yes"),CellText.String("Average Salary"))] = \
             CellText.Number(salarym,FormatSpec.Count)
      salary=0; salarym=0; n = 0; m = 0
      # Try to fetch the first case of the next split group
      row=cur.fetchone()
      if not None==row:
         spss.SplitChange("spss.com.demo")
      else:
         #There are no more cases, so quit
         break
   if row[minorityIndex]==1:
      salarym += row[salaryIndex]
```

```
      m += 1
   elif row[minorityIndex]==0:
      salary += row[salaryIndex]
      n += 1
   row=cur.fetchone()

cur.close()
spss.EndProcedure()
```

- The `spss.Submit` function is used to submit command syntax to create a split on a gender variable. The `LAYERED` subcommand on the `SPLIT FILE` command indicates that results for different split groups are to be displayed in the same table. Notice that the command syntax is executed before calling `spss.StartProcedure`.
- The `spss.SplitChange` function is called after fetching the first case from the active dataset. This is required so that the pivot table output for the first split group is handled correctly.
- Split changes are detected using the IsEndSplit method from the `Cursor` class. Once a split change is detected, the pivot table is populated with the results from the previous split.
- The value returned from the `fetchone` method is *None* at a split boundary. Once a split has been detected, you will need to call `fetchone` again to retrieve the first case of the new split group, followed by `spss.SplitChange`. *Note*: `IsEndSplit` returns *True* when the end of the dataset has been reached. Although a split boundary and the end of the dataset both result in a return value of *True* from `IsEndSplit`, the end of the dataset is identified by a return value of *None* from a subsequent call to `fetchone`, as shown in this example.

spss.StartDataStep Function

spss.StartDataStep(). *Signals the beginning of a data step.* A data step allows you to create and manage multiple datasets.

- You cannot use the following classes and functions within a data step: the `Cursor` class, the `BasePivotTable` class, the `BaseProcedure` class, the `TextBlock` class, the `StartProcedure` function, the `Submit` function, and the `StartDataStep` function (data steps cannot be nested).
- The `StartDataStep` function cannot be used if there are pending transformations. If you need to access case data in the presence of pending transformations, use the `Cursor` class.
- To end a data step, use the `EndDataStep` function.

For an example of using `StartDataStep`, see the topic on the Dataset class.

To avoid the need to check for pending transformations before starting a data step, use the DataStep class. It implicitly starts and ends a data step and executes any pending transformations.

spss.StartProcedure Function

spss.StartProcedure(procedureName,omsIdentifier). *Signals the beginning of pivot table or text block output.* Pivot table and text block output is typically associated with procedures. Procedures are user-defined Python functions or custom Python classes that can read the data, perform computations, add new variables and/or new cases to the active dataset, create new datasets, and produce pivot table output and text blocks in the IBM SPSS Statistics Viewer. Procedures have almost the same capabilities as built-in IBM SPSS Statistics procedures, such as `DESCRIPTIVES` and `REGRESSION`, but they are written in Python by users. You read the data and create new variables and/or new cases in the active dataset using the Cursor class, or create new datasets with the Dataset class. Pivot tables are created using the BasePivotTable class. Text blocks are created using the TextBlock class.

- The argument *procedureName* is a string and is the name that appears in the outline pane of the Viewer associated with the output. If the optional argument *omsIdentifier* is omitted, then *procedureName* is also the command name associated with this output when routing it with OMS (Output Management System), as used in the `COMMANDS` keyword of the `OMS` command.
- The optional argument *omsIdentifier* is a string and is the command name associated with this output when routing it with OMS (Output Management System), as used in the `COMMANDS` keyword of the `OMS` command. If *omsIdentifier* is omitted, then the value of the *procedureName* argument is used as the OMS

identifier. *omsIdentifier* is only necessary when creating procedures with localized output so that the procedure name can be localized but not the OMS identifier. See the topic "Localizing Output from Python Programs" on page 11 for more information.

- In order that names associated with output not conflict with names of existing IBM SPSS Statistics commands (when working with OMS), it is recommended that they have the form *yourcompanyname.com.procedurename*.
- Within a `StartProcedure-EndProcedure` block you cannot use the `spss.Submit` function. You cannot nest `StartProcedure-EndProcedure` blocks.
- Within a `StartProcedure-EndProcedure` block, you can create a single cursor instance.
- Instances of the `Dataset` class created within `StartProcedure-EndProcedure` blocks cannot be set as the active dataset.
- Output from `StartProcedure-EndProcedure` blocks does not support operations involving data in different split groups. When working with splits, each split should be treated as a separate set of data. To cause results from different split groups to display properly in custom pivot tables, use the SplitChange function. Use the IsEndSplit method from the `Cursor` class to determine a split change.
- `spss.StartProcedure` must be followed by spss.EndProcedure.

  Note: You can use the `spss.Procedure` class to implicitly start and end a procedure without the need to call `StartProcedure` and `EndProcedure`. See the topic "spss.Procedure Class" on page 81 for more information.

Example

As an example, we will create a procedure that calculates group means for a selected variable using a specified categorical variable to define the groups. The output of the procedure is a pivot table displaying the group means. For an alternative approach to creating the same procedure, but with a custom class, see the example for the spss.BaseProcedure class.

```
def groupMeans(groupVar,sumVar):

    #Determine variable indexes from variable names
    varCount = spss.GetVariableCount()
    groupIndex = 0
    sumIndex = 0
    for i in range(varCount):
        varName = spss.GetVariableName(i)
        if varName == groupVar:
            groupIndex = i
            continue
        elif varName == sumVar:
            sumIndex = i
            continue

    varIndex = [groupIndex,sumIndex]
    cur = spss.Cursor(varIndex)
    Counts={};Statistic={}

    #Calculate group sums
    for i in range(cur.GetCaseCount()):
        row = cur.fetchone()
        cat=int(row[0])
        Counts[cat]=Counts.get(cat,0) + 1
        Statistic[cat]=Statistic.get(cat,0) + row[1]

    cur.close()

    #Call StartProcedure
    spss.StartProcedure("mycompany.com.groupMeans")

    #Create a pivot table
    table = spss.BasePivotTable("Group Means","OMS table subtype")
    table.Append(spss.Dimension.Place.row,
                 spss.GetVariableLabel(groupIndex))
    table.Append(spss.Dimension.Place.column,
                 spss.GetVariableLabel(sumIndex))

    category2 = spss.CellText.String("Mean")
    for cat in sorted(Counts):
        category1 = spss.CellText.Number(cat)
        table[(category1,category2)] = \
```

```
        spss.CellText.Number(Statistic[cat]/Counts[cat])

    #Call EndProcedure
    spss.EndProcedure()
```

- `groupMeans` is a Python user-defined function containing the procedure that calculates the group means.
- The arguments required by the procedure are the names of the grouping variable (*groupVar*) and the variable for which group means are desired (*sumVar*).
- The name associated with output from this procedure is *mycompany.com.groupMeans*. The output consists of a pivot table populated with the group means.
- `spss.EndProcedure` marks the end of output creation.

Saving and Running Procedures

To use a procedure you have written, you save it in a Python module on the Python search path so that you can call it. A Python module is simply a text file containing Python definitions and statements. You can create a module with a Python IDE, or with any text editor, by saving a file with an extension of *.py*. The name of the file, without the *.py* extension, is then the name of the module. You can have many functions in a single module. To be sure that Python can find your new module, you may want to save it to your Python "site-packages" directory, typically */Python27/Lib/site-packages*.

For the example procedure described above, you might choose to save the definition of the `groupMeans` function to a Python module named *myprocs.py*. And be sure to include an `import spss` statement in the module. Sample command syntax to run the function is:

```
import spss, myprocs
spss.Submit("get file='/examples/data/Employee data.sav'.")
myprocs.groupMeans("educ","salary")
```

- The import statement containing `myprocs` makes the contents of the Python module *myprocs.py* available to the current session (assuming that the module is on the Python search path).
- `myprocs.groupMeans("educ","salary")` runs the `groupMeans` function for the variables *educ* and *salary* in */examples/data/Employee data.sav*.

Result

	Current Salary
Educational Level (years)	Mean
8	24399
12	25887
14	31625
15	31685
16	48226
17	59527
18	65128
19	72520
20	64313
21	65000

Figure 16. Output from the groupMeans procedure

spss.StartSPSS Function

spss.StartSPSS(hide, show, notes, nfc, nl). *Starts a session of IBM SPSS Statistics.*

- This function starts a session of IBM SPSS Statistics, for use when driving IBM SPSS Statistics from Python. The function has no effect if a session is already running. *Note*: The `spss.Submit` function automatically starts a session of IBM SPSS Statistics.
- This function has no effect when running Python from IBM SPSS Statistics (within program blocks defined by `BEGIN PROGRAM-END PROGRAM`).
- The optional argument `hide` specifies types of output items that are omitted from generated output. The value is one or more characters that specify the output types to hide. Do not include spaces when multiple types are specified. For example, to hide warnings and text output, specify `hide='WT'`. See the table that follows for the allowed values.
- The optional argument `show` specifies types of output items that are included in generated output. The value is one or more characters that specify the output types to show. Do not include spaces when multiple types are specified. For example, to show notes tables and output titles, specify `show='NE'`. See the table that follows for the allowed values.
- The optional argument `notes` specifies whether notes tables are included in generated output. The value is a Boolean and the default is `False`, which specifies that notes tables are not included. Setting `notes=True` is equivalent to setting `show='N'`.
- The optional argument `nfc` specifies whether footnotes and captions are included in the generated output for tables. The value is a Boolean and the default is `True`, which specifies that footnotes and captions are included in the output.
- The optional argument `nl` specifies whether all layers of a multi-layer table are included in the generated output. The value is a Boolean and the default is `True`, which specifies that all layers are included in the output. The value `False` specifies that only the first layer is included in the output.

Table 7. Arguments and display defaults for output items

Output Item	Description	Value for show or hide argument	Displayed by default?
Warnings	Warning messages that occurred when the procedure was run	W	Yes
Note	Information about how the output was created	N	No
Output title	Title connected to the output of a procedure	E	No
Page title	Title connected to a page	G	No
Pivot table	Tabular statistical output	P	Yes
Text output	Output not displayed in pivot tables	T	Yes

spss.StopSPSS Function

spss.StopSPSS(). *Stops IBM SPSS Statistics, ending the session.*

- This function is ignored when running Python from IBM SPSS Statistics (within program blocks defined by `BEGIN PROGRAM-END PROGRAM`).
- When running IBM SPSS Statistics from Python, this function ends the IBM SPSS Statistics session, and any subsequent `spss.Submit` functions that restart IBM SPSS Statistics will not have access to the active dataset or to any other session-specific settings (for example, `OMS` output routing commands) from the previous session.

Example: Running IBM SPSS Statistics from Python

```
import spss
#start a session and run some commands
#including one that defines an active dataset
```

```
spss.Submit("""
GET FILE '/examples/data/employee data.sav'.
FREQUENCIES VARIABLES=gender jobcat.
""")
#shutdown the session
spss.StopSPSS()
#insert a bunch of Python statements
#starting a new session and running some commands without defining
#an active dataset results in an error
spss.Submit("""
FREQUENCIES VARIABLES=gender jobcat.
""")
```

Example: Running Python from IBM SPSS Statistics

```
BEGIN PROGRAM.
import spss
#start a session and run some commands
#including one that defines an active dataset
spss.Submit("""
GET FILE '/examples/data/employee data.sav'.
FREQUENCIES VARIABLES=gender jobcat.
""")
#following function is ignored
spss.StopSPSS()
#active dataset still exists and subsequent spss.Submit functions
#will work with that active dataset.
spss.Submit("""
FREQUENCIES VARIABLES=gender jobcat.
""")
END PROGRAM.
```

spss.Submit Function

spss.Submit(command text). *Submits the command text to IBM SPSS Statistics for processing.* The argument can be a quoted string, a list, or a tuple.

- The argument should resolve to one or more complete IBM SPSS Statistics commands.
- For lists and tuples, each element must resolve to a string.
- You can also use the Python triple-quoted string convention to specify blocks of IBM SPSS Statistics commands on multiple lines that more closely resemble the way you might normally write command syntax.
- If IBM SPSS Statistics is not currently running (when driving IBM SPSS Statistics from Python), `spss.Submit` will start the IBM SPSS Statistics backend processor.
- Submitted syntax for `MATRIX-END MATRIX` and `BEGIN DATA-END DATA` blocks cannot be split across `BEGIN PROGRAM-END PROGRAM` blocks.
- The following commands are not supported by `Submit` when driving IBM SPSS Statistics from Python: `OUTPUT EXPORT`, `OUTPUT OPEN` and `OUTPUT SAVE`.

Example

```
BEGIN PROGRAM.
import spss
#run a single command
spss.Submit("DISPLAY NAMES.")
#run two commands
spss.Submit(["DISPLAY NAMES.", "SHOW $VARS."])

#build and run two commands
command1="FREQUENCIES VARIABLES=var1."
command2="DESCRIPTIVES VARIABLES=var3."
spss.Submit([command1, command2])
END PROGRAM.
```

Example: Triple-Quoted Strings

```
BEGIN PROGRAM.
import spss
file="/examples/data/demo.sav"
varlist="marital gender inccat"
spss.Submit("""
GET FILE='%s'.
FREQUENCIES VARIABLES=%s
```

```
  /STATISTICS NONE
  /BARCHART.
""" %(file,varlist))
END PROGRAM.
```

Within the triple-quoted string, `%s` is used for string substitution; thus, you can insert Python variables that resolve to strings in the quoted block of commands.

spss.TextBlock Class

spss.TextBlock(name,content,outline). *Creates and populates a text block item in the Viewer.* The argument *name* is a string that specifies the name of this item in the outline pane of the Viewer. The argument *content* is a string that specifies the text. The string may include the escape sequence `\n` to specify line breaks, but must otherwise be specified as plain text (HTML and rich text formatting are not supported). You can also add lines using the append method. The optional argument *outline* is a string that specifies a title for this item that appears in the outline pane of the Viewer. The item for the text block itself will be placed one level deeper than the item for the *outline* title. If *outline* is omitted, the Viewer item for the text block will be placed one level deeper than the root item for the output containing the text block.

An instance of the `TextBlock` class can only be used within a `StartProcedure-EndProcedure` block or within a custom procedure class based on the `spss.BaseProcedure` class.

Example

```
import spss
spss.StartProcedure("mycompany.com.demo")
textBlock = spss.TextBlock("Text block name",
                           "A single line of text.")
spss.EndProcedure()
```

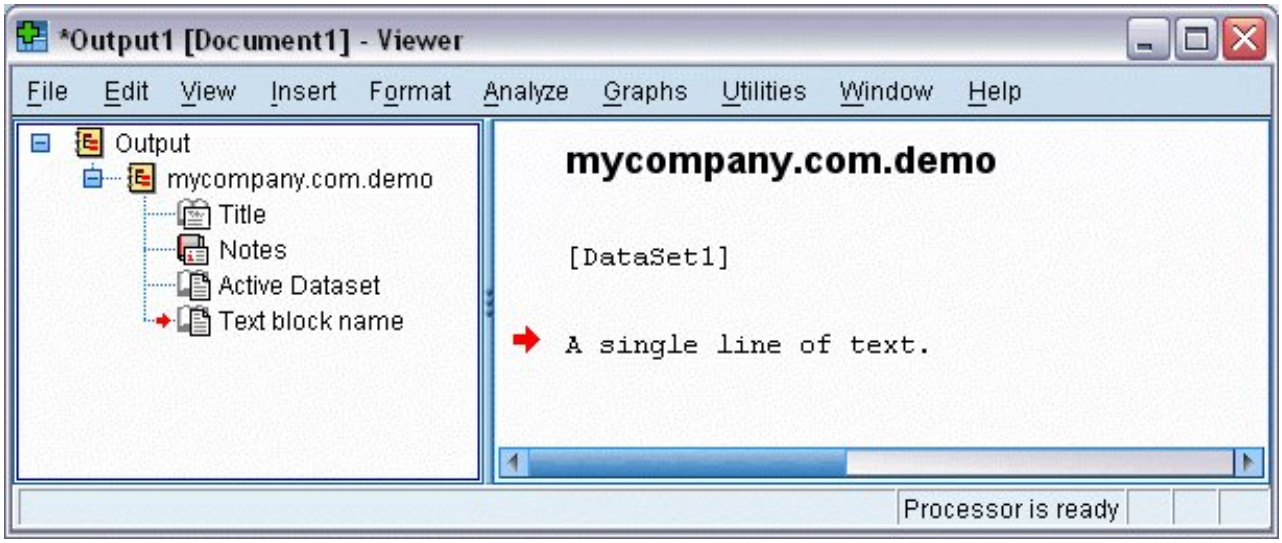


Figure 17. Sample text block

- This example shows how to generate a text block within a `spss.StartProcedure-spss.EndProcedure` block. The output will be contained under an item named *mycompany.com.demo* in the outline pane of the Viewer.
- The variable *textBlock* stores a reference to the instance of the text block object. You will need this object reference if you intend to append additional lines to the text block with the `append` method.

append Method

.append(line,skip). *Appends lines to an existing text block.* The argument *line* is a string that specifies the text. The string may include the escape sequence `\n` to specify line breaks, but must otherwise be

specified as plain text (HTML and rich text formatting are not supported). The optional argument *skip* specifies the number of new lines to create when appending the specified line. The default is 1 and results in appending the single specified line. Integers greater than 1 will result in blank lines preceding the appended line. For example, specifying `skip=3` will result in two blank lines before the appended line.

Example

```
import spss
spss.StartProcedure("mycompany.com.demo")
textBlock = spss.TextBlock("Text block name",
                           "A single line of text.")
textBlock.append("A second line of text.")
textBlock.append("A third line of text preceded by a blank line.",skip=2)
spss.EndProcedure()
```

Chapter 3. Scripting Guide

Introduction to Python Scripts

The Scripting Facility for IBM SPSS Statistics provides the ability to create **Python** ® **scripts** that operate on the IBM SPSS Statistics user interface, manipulate output objects, and run command syntax. This feature requires the IBM SPSS Statistics - Integration Plug-in for Python, which is installed by default with your IBM SPSS Statistics product..

A companion interface is available for creating **Python programs** that enable you to control the flow of command syntax jobs, read and write data, and create custom procedures. See the topic "Introduction to Python Programs" on page 3 for more information.

Scope

You can run Python scripts directly from within IBM SPSS Statistics, from within Python programs, or from an external Python process, such as a Python IDE or the Python interpreter.

Python Script Run from IBM SPSS Statistics. You can run a Python script from Utilities>Run Script or from the Python script editor which is launched when opening a Python file (.py) from File>Open>Script. Scripts run from the Python editor that is launched from IBM SPSS Statistics operate on the IBM SPSS Statistics client that launched the editor. This allows you to debug your Python code from a Python editor.

Python Script Run from an External Python Process. You can run a Python script from any external Python process, such as a Python IDE that is not launched from IBM SPSS Statistics, or the Python interpreter. The script will attempt to connect to an existing IBM SPSS Statistics client. If more than one client is found, a connection is made to the most recently launched one. If an existing client is not found, the Python script starts up a new instance of the IBM SPSS Statistics client. By default, the Data Editor and Viewer are invisible for the new client. You can choose to make them visible or work in invisible mode with datasets and output documents.

- **Mac.** To run a Python script from an external Python process on Mac, launch the *Programmability External Python Process* application, installed with IBM SPSS Statistics - Essentials for Python and located in the directory where IBM SPSS Statistics is installed. The application launches IDLE (the default IDE provided with Python) and sets environment variables necessary for driving IBM SPSS Statistics.

Python Script Run from Python Program. You can run a Python script from a Python program by importing the Python module containing the script and calling the function in the module that implements the script. You can also call Python script methods directly from within a Python program. See the topic "Running Python Scripts from Python Programs" on page 98 for more information.

- This feature is not available when running a Python program from an external Python process or when running a Python program from the IBM SPSS Statistics Batch Facility (available with IBM SPSS Statistics Server).
- When running Python scripting code from a Python program in distributed mode, you may need to configure your firewall to allow access from the remote server to which you are connected.

Limitations

- The interfaces exposed by the `spss` module (the module used for Python programs) cannot be used in a Python script.
- Calling methods in the `SpssClient` module with keyword arguments--in other words, keyword = value--is not supported.

Help

General information on the Scripting Facility for IBM SPSS Statistics and additional information on Python scripts is available from Core System>Scripting Facility in the Help system .

Script Editor for the Python Programming Language

For the Python programming language, the default editor is IDLE, which is provided with Python. IDLE provides an integrated development environment (IDE) with a limited set of features. Many IDE's are available for the Python programming language. For instance, on Windows you might choose to use the freely available PythonWin IDE.

To change the script editor for the Python programming language:

1. Open the file *clientscriptingcfg.ini*, located in the directory where IBM SPSS Statistics is installed. *Note*: *clientscriptingcfg.ini* must be edited with a UTF-16 aware editor, such as SciTE on Windows or the TextEdit application on Mac.
2. Under the section that is labeled `[Python2]` or `[Python3]`, change the value of `EDITOR_PATH` to point to the file that starts the editor. The specifications for Python 2 and Python 3 are independent.
3. In that same section, change the value of `EDITOR_ARGS` to handle any arguments that need to be passed to the editor. If no arguments are required, remove any existing values.

Working with Multiple Versions of IBM SPSS Statistics

For versions 16.0 to 21.0, special considerations apply when multiple versions of the IBM SPSS Statistics - Integration Plug-in for Python (each associated with a major version of IBM SPSS Statistics, such as 20 or 21) are installed on your computer.

Running Python Scripts from Within IBM SPSS Statistics

Important: This section only applies to versions 16.0 to 21.0.

By default, Python scripts run from within the last installed version of IBM SPSS Statistics will automatically use the appropriate version of the plug-in. To run Python scripts from within a different version of IBM SPSS Statistics, use the SpssClient.SetDefaultJCVersion method to set the default to a different version (the setting persists across sessions). You can then run Python scripts from within the other version. If you are attempting to change the default version from 16.0 to 17.0, additional configuration is required; please see the Notes below.

Running Python Scripts from an External Python Process

Important: This section only applies to versions 16.0 to 21.0. For version 22 and higher, see "Running IBM SPSS Statistics from an External Python Process" on page 9.

When running Python scripts from a separate Python process, such as the Python interpreter or a Python IDE, the plug-in will drive the version of IBM SPSS Statistics that matches the default plug-in version specified for that version of Python. Unless you change it, the default plug-in version for a given version of Python (such as Python 2.7) is the last one installed. You can view the default version using the SpssClient.GetDefaultJCVersion method and you can change the default version using the SpssClient.SetDefaultJCVersion method. The setting persists across sessions. If you are attempting to change the default version from 16.0 to 17.0, additional configuration is required; please see the Notes below.

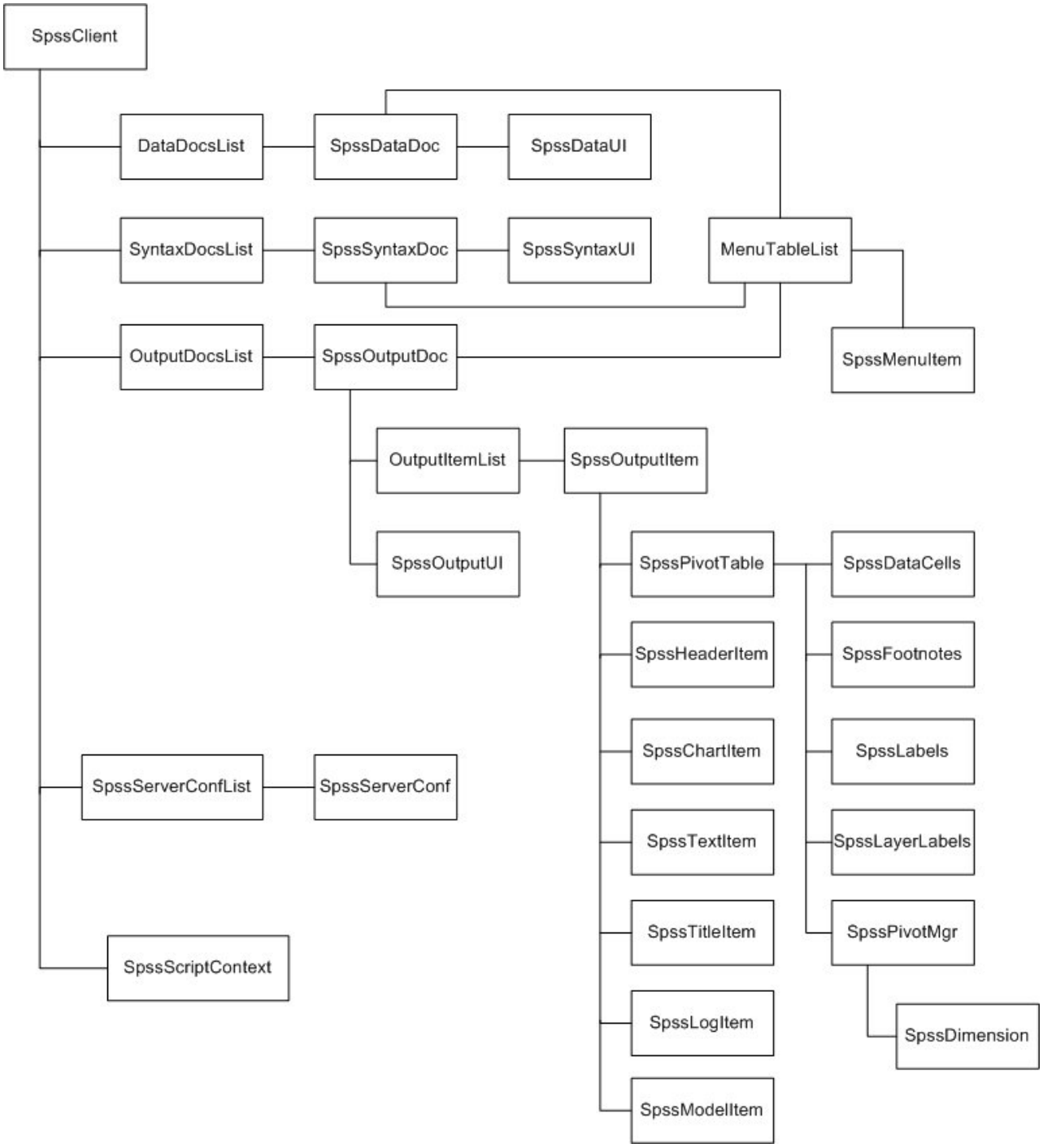
Note:

To change the default version from 16.0 to 17.0, you will need to manually modify the file *SpssClient.pth* located in the Python 2.5 *site-packages* directory. Change the order of entries in the file so that the first line is `SpssClient170`. You should also ensure that the first line in *spss.pth* (also located in *site-packages*) is `spss170`.

- **Windows.** The *site-packages* directory is located in the *Lib* directory under the Python 2.5 installation directory—for example, *C:\Python25\Lib\site-packages*.
- **Mac OS X 10.4 (Tiger).** The *site-packages* directory is located at */Library/Frameworks/Python.framework/Versions/2.5/lib/python2.5/site-packages*.
- **Mac OS X 10.5 (Leopard).** The *site-packages* directory is located at */Library/Python/2.5/site-packages*.
- **Linux and UNIX Server.** The *site-packages* directory is located in the */lib/python2.5/* directory under the Python 2.5 installation directory—for example, */usr/local/python25/lib/python2.5/site-packages*.

Class Hierarchy for Scripting Facility

The following diagram shows the hierarchy of classes available to Python scripts.



Getting Started with Python Scripts

The basic structure of a Python script is:

```
import SpssClient
SpssClient.StartClient()
<Python language statements>
SpssClient.StopClient()
```

- The `import SpssClient` statement imports the Python module containing the IBM SPSS Statistics classes and methods available in the Python scripting interface.

- `SpssClient.StartClient()` provides a connection to the associated IBM SPSS Statistics client, enabling the script to retrieve information from the client and to perform operations on objects managed by the client, such as pivot tables. Whether the script connects to an existing client or starts up a new client depends on how the script was invoked. See the topic "Introduction to Python Scripts" on page 93 for more information.
- `SpssClient.StopClient()` terminates the connection to the IBM SPSS Statistics client and should be called at the completion of each Python script.

  Note: If you're running a Python script from an external Python process that starts up a new client, call `SpssClient.Exit()` before `SpssClient.StopClient()`.

Example

This script accesses the designated output document and sets each of the pivot tables as selected.

```
import SpssClient
SpssClient.StartClient()

OutputDoc = SpssClient.GetDesignatedOutputDoc()
OutputItems = OutputDoc.GetOutputItems()

for index in range(OutputItems.Size()):
   OutputItem = OutputItems.GetItemAt(index)
   if OutputItem.GetType() == SpssClient.OutputItemType.PIVOT:
         OutputItem.SetSelected(True)
SpssClient.StopClient()
```

Target for Standard output

The Python `print` statement writes output to Python's standard output. When you run a Python script from Utilities>Run Script, Python's standard output is directed to a log item in the IBM SPSS Statistics Viewer.

Getting Started with Autoscripts in Python

Autoscripts are scripts that run automatically when triggered by the creation of specific pieces of output from selected procedures and typically require a reference to the object that triggered the script. They may also require a reference to the associated output document and possibly the index of the output item in the output document. These values are obtained from the `SpssScriptContext` object, as shown in this example of an autoscript that transposes the rows and columns of a pivot table.

```
import SpssClient
SpssClient.StartClient()

SpssScriptContext = SpssClient.GetScriptContext()
SpssOutputItem = SpssScriptContext.GetOutputItem()
SpssPivotTable = SpssOutputItem.GetSpecificType()
SpssPivotMgr = SpssPivotTable.PivotManager()
SpssPivotMgr.TransposeRowsWithColumns()

SpssClient.StopClient()
```

- `SpssClient.GetScriptContext` returns an `SpssScriptContext` object that provides values for use by the autoscript.
- The `GetOutputItem` method of the `SpssScriptContext` object returns the output item that triggered the current autoscript--in this example, the pivot table whose rows and columns are to be transposed.

Although not used in this example, the `GetOutputDoc` method of the `SpssScriptContext` object returns the associated output document, and the `GetOutputItemIndex` method returns the index (in the associated output document) of the output item that triggered the autoscript.

General information on autoscripts is available from Core System>Scripting Facility in the Help system.

Detecting When a Script is Run as an Autoscript

Using the GetScriptContext method, you can detect when a script is being run as an autoscript. This allows you to code a script so that it functions in either context (autoscript or not). This trivial script illustrates the approach.

```
import SpssClient
SpssClient.StartClient()

SpssScriptContext = SpssClient.GetScriptContext()
if SpssScriptContext == None:
   print "I'm not an autoscript"
else:
   print "I'm an autoscript"

SpssClient.StopClient()
```

- When a script is not run as an autoscript, the GetScriptContext method will return a value of None.
- Given the if-else logic in this example, you would include your autoscript-specific code in the else clause. Any code that is not to be run in the context of an autoscript would be included in the if clause. Of course you can also include code that is to be run in either context.

Running Python Scripts from Python Programs

You can run Python scripts from Python programs and you can call Python script methods from within a Python program. This allows you to write Python programs that operate on user interface and output objects.

- This feature is only available when running a Python program from the IBM SPSS Statistics client--within a BEGIN PROGRAM-END PROGRAM block in command syntax or within an extension command. It is not available when running a Python program from an external Python process.
- When running Python scripting code from a Python program in distributed mode, you may need to configure your firewall to allow access from the remote server to which you are connected.

Example: Calling a Python Script from a Python Program

This example shows a Python program that creates a custom pivot table and calls a Python script to make the column labels of the table bold.

```
BEGIN PROGRAM.
import spss, MakeColsBold
spss.StartProcedure("Demo")
table = spss.BasePivotTable("Sample Table","OMS subtype")
table.SimplePivotTable(rowlabels = ["1","2"],
                       collabels = ["A","B"],
                       cells = ["1A","1B","2A","2B"])
spss.EndProcedure()
MakeColsBold.Run("Sample Table")
END PROGRAM.
```

- Python programs use the interface exposed by the Python spss module, so the first line of the program contains an import statement for that module. The Python script is assumed to be contained in a Python module named MakeColsBold, so the import statement also includes that module.
- The code from spss.StartProcedure to spss.EndProcedure creates a pivot table titled "Sample Table".
- MakeColsBold.Run("Sample Table") calls the Run function in the MakeColsBold module and passes the value "Sample Table" as the argument. The Run function implements the Python script to make the column labels of the specified table bold.

The content of the MakeColsBold module is as follows:

```
import SpssClient

def Run(tableName):
   SpssClient.StartClient()
   OutputDoc = SpssClient.GetDesignatedOutputDoc()
   OutputItems = OutputDoc.GetOutputItems()
   for index in range(OutputItems.Size()):
      OutputItem = OutputItems.GetItemAt(index)
      if OutputItem.GetType() == SpssClient.OutputItemType.PIVOT \
        and OutputItem.GetDescription() == tableName:
         PivotTable = OutputItem.GetSpecificType()
         ColumnLabels = PivotTable.ColumnLabelArray()
```

```
        for i in range(ColumnLabels.GetNumColumns()):
           ColumnLabels.SelectLabelAt(1,i)
        PivotTable.SetTextStyle(SpssClient.SpssTextStyleTypes.SpssTSBold)
SpssClient.StopClient()
```

- The `import SpssClient` statement is needed to access the classes and methods available in the Python scripting interface.
- The module contains a single function named `Run`, which implements the script. It takes a single argument that specifies the name of the table to modify. There is nothing special about the name *Run* and the module is not limited to a single function. You can create a module that contains many functions, each of which implements a different script.
- The `Run` function calls `SpssClient.StartClient()` to provide a connection to the associated IBM SPSS Statistics client and `SpssClient.StopClient()` to terminate the connection at the completion of the script.

Example: Calling Python Scripting Methods Directly from a Python Program

This example shows a Python program that creates a custom pivot table and makes direct calls to Python scripting methods to make the title of the table italic.

```
BEGIN PROGRAM.
import spss, SpssClient
spss.StartProcedure("Demo")
table = spss.BasePivotTable("Sample Table","OMS subtype")
table.SimplePivotTable(cells = ["A","B","C","D"])
spss.EndProcedure()

SpssClient.StartClient()
OutputDoc = SpssClient.GetDesignatedOutputDoc()
OutputItems = OutputDoc.GetOutputItems()
OutputItem = OutputItems.GetItemAt(OutputItems.Size()-1)
PivotTable = OutputItem.GetSpecificType()
PivotTable.SelectTitle()
PivotTable.SetTextStyle(SpssClient.SpssTextStyleTypes.SpssTSItalic)
SpssClient.StopClient()
END PROGRAM.
```

- The `import spss, SpssClient` statement provides access to the classes and methods available for Python programs (`spss`) as well as those for Python scripts (`SpssClient`).
- The code from `spss.StartProcedure` to `spss.EndProcedure` is the Python program code that creates the pivot table.
- The code from `SpssClient.StartClient()` to `SpssClient.StopClient()` is the Python script code that makes the title italic.

SpssClient Class

The `SpssClient` class is the top level class for the IBM SPSS Statistics Python scripting interface. From an `SpssClient` object you can:

- Access the current data, syntax, or output document.
- Open and access a saved data, syntax, or output document.
- Create and access a new data, syntax, or output document.
- Obtain a list of all open data, syntax, or output documents.
- Run command syntax.
- Get and set options available from Edit>Options in the user interface.
- Get and set export options for exporting output.
- Get values pertinent to an autoscript, such as the output item that triggered the autoscript.
- Obtain information about configured instances of IBM SPSS Statistics Server and configure new instances.

The SpssClient object is always available to a script, but you must call SpssClient.StartClient to establish a connection to the IBM SPSS Statistics client before you can use any of the other methods in the class. See the topic "Getting Started with Python Scripts" on page 96 for more information.

CreateNewServer Method

Creates a new server configuration and returns an SpssServerConf object. To add this server to the list of configured servers, use the Add method in the SpssServerConfList class. You can obtain an instance of SpssServerConfList from the GetConfiguredServers method in the SpssClient class.

Syntax

```
SpssServerConf=SpssClient.CreateNewServer(serverName,port,desc)
```

Parameters

serverName. The machine name or IP address of the IBM SPSS Statistics Server machine

port. Port number for IBM SPSS Statistics Server

desc. Textual description of the server

Exit Method

Terminates the instance of the IBM SPSS Statistics client associated with the current script. This method is intended for use when running a script from an external Python process (such as a Python IDE or the Python interpreter), and will terminate the instance of the IBM SPSS Statistics client associated with the script. The method has no effect when called from a script that is run from within the IBM SPSS Statistics client, either through Utilities>Run Script or from a Python IDE launched from File>Open>Script or File>New>Script.

Syntax

```
SpssClient.Exit()
```

GetActiveDataDoc Method

Returns the active dataset as an SpssDataDoc object.

Syntax

```
SpssDataDoc=SpssClient.GetActiveDataDoc()
```

GetConfiguredServers Method

Returns the list of configured servers as an SpssServerConfList object. The list consists of SpssServerConf objects for each of the configured servers, including the local computer.

Syntax

```
SpssServerConfList=SpssClient.GetConfiguredServers()
```

GetCurrentDirectory Method

Returns the current working directory of the IBM SPSS Statistics client.

Syntax

```
SpssClient.GetCurrentDirectory()
```

GetCurrentServer Method

Returns an `SpssServerConf` object representing the current server (may be an instance of IBM SPSS Statistics Server or the local computer).

Syntax

```
SpssServerConf=SpssClient.GetCurrentServer()
```

GetDataDocuments Method

Returns the list of open datasets as a DataDocsList object. Each item in the list is an `SpssDataDoc` object.

Syntax

```
DataDocsList=SpssClient.GetDataDocuments()
```

GetDefaultJCVersion Method

Returns a string specifying the default version of the IBM SPSS Statistics - Integration Plug-in for Python used for Python scripts--for example, `"SpssClient170"` for version 17.0. This method is useful when working with multiple versions of the plug-in on a given machine (see Note below). You can change the default using the SetDefaultJCVersion method.

Syntax

```
SpssClient.GetDefaultJCVersion()
```

Note: The methods for managing multiple versions of the plug-in (`SpssClient.GetDefaultJCVersion`, and `SpssClient.SetDefaultJCVersion`) operate within a given Python version, not across Python versions. For example, if you are driving IBM SPSS Statistics from a Python IDE installed for Python 2.6 then you can view and control the versions of the plug-in installed for Python 2.6.

See the topic "Working with Multiple Versions of IBM SPSS Statistics" on page 94 for more information.

GetDefaultServer Method

Returns an `SpssServerConf` object representing the default server (may be an instance of IBM SPSS Statistics Server or the local computer).

Syntax

```
SpssServerConf=SpssClient.GetDefaultServer()
```

GetDesignatedOutputDoc Method

Returns an `SpssOutputDoc` object representing the designated output document.

- If you have more than one open output document, output is routed to the designated one.

Syntax

```
SpssOutputDoc=SpssClient.GetDesignatedOutputDoc()
```

GetDesignatedSyntaxDoc Method

Returns an SpssSyntaxDoc object representing the designated syntax document.

- If you have more than one open syntax document, command syntax is pasted into the designated one.

Syntax

```
SpssSyntaxDoc=SpssClient.GetDesignatedSyntaxDoc()
```

GetExportOption Method

Returns the value of the specified export option, as a string.

Syntax

```
SpssClient.GetExportOption(option)
```

Parameters

The value of *option* is the identifier SpssClient.ExportOptions, followed by a period (.) and the name of the option--for example, SpssClient.ExportOptions.GraphExportType. See Export Options for the available list of options.

GetLocale Method

Returns a string specifying the current locale. The locale consists of the language, country and char set information.

Syntax

```
SpssClient.GetLocale()
```

GetLocalServer Method

Returns an SpssServerConf object representing the local computer.

Syntax

```
SpssServerConf=SpssClient.GetLocalServer()
```

GetOutputDocuments Method

Returns the list of open output documents as an OutputDocsList object. Each item in the list is an SpssOutputDoc object.

Syntax

```
OutputDocsList=SpssClient.GetOutputDocuments()
```

GetPreference Method

Returns the value of the specified preference option, as a string.

Syntax

```
SpssClient.GetPreference(option)
```

The value of *option* is the identifier `SpssClient.PreferenceOptions`, followed by a period (.) and the name of the option--for example, `SpssClient.PreferenceOptions.VariableListDisplay`. See Preference Options for the available list of options.

GetScriptContext Method

Returns an SpssScriptContext object that allows you to determine the context in which a script is being run--as an autoscript, or not.

- When the script is being run as an autoscript, the returned `SpssScriptContext` object provides access to the output item that triggered the autoscript as well as the associated output document.
- When the script is not being run as an autoscript, `GetScriptContext` returns `None`.

Syntax

```
SpssScriptContext=SpssClient.GetScriptContext()
```

GetSPSSOptions Method

Returns a string which is a concatenation of three-letter abbreviations for each of the licensed options. You can determine if a specified option is available from the IsOptionAvailable method.

Syntax

```
options=SpssClient.GetSPSSOptions()
```

The options and associated three-letter abbreviations are as follows:

Bas. Base

Pro. Regression

Adv. Advanced Statistics

Cyt. Exact Test

Cat. Categories

Mva. Missing Values

Con. Conjoint

Msa. Custom Tables

Csp. Complex Samples

Tre. Decision Trees

Vld. Data Preparation

Trd. Forecasting

Pes. Statistics Adaptor

Neu. Neural Networks

Rfm. RFM

GetSPSSPath Method

Returns a string specifying the path to the IBM SPSS Statistics installation directory.

Syntax

```
path=SpssClient.GetSPSSPath()
```

GetSPSSVersion Method

Returns a string specifying the IBM SPSS Statistics version.

Syntax

```
version=SpssClient.GetSPSSVersion()
```

GetSyntaxDocuments Method

Returns the list of open syntax documents as a SyntaxDocsList object. Each item in the list is an `SpssSyntaxDoc` object.

Syntax

```
SyntaxDocsList=SpssClient.GetSyntaxDocuments()
```

GetUIAlerts Method

Returns the current setting of UI alerts for the client. The result is Boolean.

Syntax

```
SpssClient.GetUIAlerts()
```

Returns

True. Alerts are displayed in the UI

False. UI alerts are suppressed

IsDataDocInUse Method

Indicates whether a specified data file is in use by another instance of IBM SPSS Statistics. The result is Boolean. The argument is a string specifying the path to the data file. *Note*: This method is not supported on Mac and Linux.

Syntax

```
SpssClient.IsDataDocInUse(fileName)
```

On Windows, it is recommended to use raw strings for file paths, or replace backslashes with forward slashes (IBM SPSS Statistics accepts a forward slash for any backslash in a file specification). Raw strings are specified by prefacing the string with r, as in r'c:\examples\mydata.sav'. In raw mode, Python treats all backslashes in the string as the backslash character and not as the start of an escape sequence.

IsDistributedMode

Indicates whether the scripting process is being run in distributed mode. The result is Boolean.

Syntax

```
SpssClient.IsDistributedMode()
```

IsOptionAvailable Method

Checks if the IBM SPSS Statistics client is licensed for a specified optional component. The result is Boolean. You can obtain a list of all available options from the GetSPSSOptions method.

Syntax

```
SpssClient.IsOptionAvailable(licOption)
```

Parameters

The parameter *licOption* specifies the option.

Table 8. Option values

Value	Description
SpssClient.LicenseOption.BASE	Base
SpssClient.LicenseOption.PRO_STATS	Regression
SpssClient.LicenseOption.ADVANCED_STATS	Advanced Statistics
SpssClient.LicenseOption.CYTEL	Exact Test
SpssClient.LicenseOption.MARKET_RESEARCH	Categories
SpssClient.LicenseOption.MISSING_VALUES	Missing Values
SpssClient.LicenseOption.CONJOINT	Conjoint
SpssClient.LicenseOption.CUSTOM_TABLES	Custom Tables
SpssClient.LicenseOption.COMPLEX_SAMPLE	Complex Samples
SpssClient.LicenseOption.TREEVIEW	Decision Trees
SpssClient.LicenseOption.VALIDATEDATA	Data Preparation
SpssClient.LicenseOption.TRENDS	Forecasting
SpssClient.LicenseOption.PES	Statistics Adaptor
SpssClient.LicenseOption.NEURAL_NETWORK	Neural Networks
SpssClient.LicenseOption.RFM	RFM

Returns

True. The option is available.

False. The option is not available or the license for the option has expired.

LogToViewer Method

Writes the specified content to the designated output document as a log item. The content is appended to the last log item in the output document.

Syntax

```
SpssClient.LogToViewer(content)
```

Parameters

content. A string

NewDataDoc Method

Creates a new dataset and makes it the active dataset. The method returns an SpssDataDoc object associated with the new dataset.

Syntax

```
SpssDataDoc=SpssClient.NewDataDoc()
```

NewOutputDoc Method

Creates a new output document and makes it the designated output document. The method returns an SpssOutputDoc object associated with the new output document.

Syntax

```
SpssOutputDoc=SpssClient.NewOutputDoc()
```

NewSyntaxDoc Method

Creates a new syntax document and makes it the designated syntax document. The method returns an SpssSyntaxDoc object associated with the new syntax document.

Syntax

```
SpssSyntaxDoc=SpssClient.NewSyntaxDoc()
```

OpenDataDoc Method

Opens the specified data document and makes it the active dataset. The method returns an SpssDataDoc object.

- This method is not available when called from a Python program in distributed mode (Python programs make use of the interface exposed by the Python `spss` module).

Syntax

```
SpssDataDoc=SpssClient.OpenDataDoc(fileName,password=None)
```

Parameters

fileName. The path and file name of the data document, as a string.

password. A string specifying the password required to open the file. Only applies to encrypted data files. The password can be specified as encrypted or unencrypted. Encrypted passwords are created when pasting command syntax, for an encrypted file, from the Save Data As dialog.

On Windows, it is recommended to use raw strings for file paths, or replace backslashes with forward slashes (IBM SPSS Statistics accepts a forward slash for any backslash in a file specification). Raw strings are specified by prefacing the string with r, as in r'c:\examples\mydata.sav'. In raw mode, Python treats all backslashes in the string as the backslash character and not as the start of an escape sequence.

OpenOutputDoc Method

Opens the specified output document and makes it the designated output document. The method returns an SpssOutputDoc object. By default, the associated Viewer window is invisible. Use the SetVisible method from the `SpssOutputUI` class to make the Viewer window visible. You get an `SpssOutputUI` object using the GetOutputUI method of the `SpssOutputDoc` object.

Syntax

```
SpssOutputDoc=SpssClient.OpenOutputDoc(fileName,password=None)
```

Parameters

fileName. The path and file name of the output document, as a string.

password. A string specifying the password required to open the file. Only applies to encrypted output files. The password can be specified as encrypted or unencrypted. Encrypted passwords are created when pasting command syntax, for an encrypted file, from the Save Output As dialog.

On Windows, it is recommended to use raw strings for file paths, or replace backslashes with forward slashes (IBM SPSS Statistics accepts a forward slash for any backslash in a file specification). Raw strings are specified by prefacing the string with r, as in r'c:\examples\mydata.sav'. In raw mode, Python treats all backslashes in the string as the backslash character and not as the start of an escape sequence.

OpenSyntaxDoc Method

Opens the specified syntax document and makes it the designated syntax document. The method returns an `SpssSyntaxDoc` object. By default, the associated Syntax Editor window is invisible. Use the SetVisible method from the `SpssSyntaxUI` class to make the Syntax Editor window visible. You get an `SpssSyntaxUI` object using the GetSyntaxUI method of the `SpssSyntaxDoc` object.

Syntax

```
SpssSyntaxDoc=SpssClient.OpenSyntaxDoc(fileName,password=None)
```

Parameters

fileName. The path and file name of the syntax document, as a string.

password. A string that specifies the password that is required to open the file. This setting applies only to encrypted syntax files. The password can be specified as encrypted or unencrypted. For reference, passwords are always encrypted in pasted syntax.

On Windows, it is recommended to use raw strings for file paths, or replace backslashes with forward slashes (IBM SPSS Statistics accepts a forward slash for any backslash in a file specification). Raw strings are specified by prefacing the string with r, as in r'c:\examples\mydata.sav'. In raw mode, Python treats all backslashes in the string as the backslash character and not as the start of an escape sequence.

RunSyntax Method

Executes a set of syntax commands.

- The submitted commands are executed synchronously with any other submitted command syntax.
- This method cannot be called within a script that is run from the `SCRIPT` command. It is also not available when called from a Python program in distributed mode (Python programs make use of the interface exposed by the Python `spss` module).

Syntax

```
SpssClient.RunSyntax(syntaxCommands)
```

Parameters

syntaxCommands. A string specifying command syntax. If the string is empty, no error is returned and the script continues. Commands must end in a period (command terminator).

Example

```
SpssClient.RunSyntax("GET FILE='/examples/data/Employee data.sav'.")
```

To specify multiple commands, separate each command by the escape sequence for a linefeed, "\n", or enclose the set of commands in a triple-quoted string, as in:

```
SpssClient.RunSyntax(r"""
GET FILE='/examples/data/Employee data.sav'.
SORT CASES BY gender.
SPLIT FILE
  LAYERED BY gender.
DESCRIPTIVES
  VARIABLES=salary salbegin jobtime prevexp
  /STATISTICS=MEAN STDDEV MIN MAX.
SPLIT FILE OFF.
""")
```

- The triple double quotes enclose a block of command syntax that is submitted for processing, retaining the line breaks. You can use either triple single quotes or triple double quotes, but you must use the same type (single or double) on both sides of the command syntax block.
- Notice that the triple-quoted expression is prefixed with the letter r. The r prefix to a string specifies Python's raw mode. In raw mode, Python treats all backslashes in the string as the backslash character and not as the start of an escape sequence.

SaveServers Method

Saves the set of configured servers so that new server configurations added during the current session will persist across sessions.

Syntax

```
SpssClient.SaveServers()
```

ScriptParameter Method

Retrieves a parameter passed to the script when calling the script from a SCRIPT command within command syntax. Only a single parameter can be passed and it must be a quoted string.

Syntax

```
SpssClient.ScriptParameter(0)
```

SetCurrentDirectory Method

Sets the current working directory of the IBM SPSS Statistics client to a specified value.

Syntax

```
SpssClient.SetCurrentDirectory(newDir)
```

Parameters

newDir. The absolute path to the new working directory, as a string.

On Windows, it is recommended to use raw strings for file paths, or replace backslashes with forward slashes (IBM SPSS Statistics accepts a forward slash for any backslash in a file specification). Raw strings are specified by prefacing the string with r, as in r'c:\examples\mydata.sav'. In raw mode, Python treats all backslashes in the string as the backslash character and not as the start of an escape sequence.

SetDefaultJCVersion Method

Sets the default version of the IBM SPSS Statistics - Integration Plug-in for Python used for Python scripts. This method is useful when working with multiple versions of the plug-in on a given machine (see Note below). The value of the argument is a quoted string or an integer specifying a plug-in version--for example, `"SpssClient160"` or `160` for version 16.0. You can view the default using the GetDefaultJCVersion method.

- `SetDefaultJCVersion` also sets the default version of the IBM SPSS Statistics - Integration Plug-in for Python used for Python programs (Python code that utilizes the `spss` module).

Syntax

```
SpssClient.SetDefaultJCVersion(version)
```

Note: The methods for managing multiple versions of the plug-in (`SpssClient.GetDefaultJCVersion`, and `SpssClient.SetDefaultJCVersion`) operate within a given Python version, not across Python versions. For example, if you are driving IBM SPSS Statistics from a Python IDE installed for Python 2.6 then you can view and control the versions of the plug-in installed for Python 2.6.

See the topic "Working with Multiple Versions of IBM SPSS Statistics" on page 94 for more information.

SetExportOption Method

Sets the value of the specified export option to the value provided.

Syntax

```
SpssClient.SetExportOption(option,value)
```

Parameters

value. A string

For a list of the available export options and associated settings, see . The value of *option* is the identifier `SpssClient.ExportOptions`, followed by a period (.) and the name of the option--for example, `SpssClient.ExportOptions.GraphExportType`.

SetPreference Method

Sets the value of the specified preference option to the value provided.

Syntax

```
SpssClient.SetPreference(option,value)
```

Parameters

value. A string

For a list of the available preference options and settings, see Appendix E, "Preference Options," on page 255. The value of *option* is the identifier `SpssClient.PreferenceOptions`, followed by a period (.) and the name of the option--for example, `SpssClient.PreferenceOptions.VariableListDisplay`.

SetUIAlerts Method

Specifies the setting of UI alerts for the IBM SPSS Statistics client.

Syntax

```
SpssClient.SetUIAlerts(showUIAlerts)
```

Parameters

showUIAlerts. True if alerts are to be displayed in the UI and False if UI alerts are to be suppressed.

If False is specified, any alerts triggered by script operations are propagated to the script as an exception.

StartClient Method

Establishes a connection to the IBM SPSS Statistics client and is required for every Python script.

- If the script is run from an external Python process (such as a Python IDE or the Python interpreter), an attempt is made to connect to an existing IBM SPSS Statistics client. If more than one client is found, a connection is made to the most recently launched one. If an existing client is not found, a new and invisible instance of the IBM SPSS Statistics client is started and a connection to it is established. You can make the associated Data Editor window visible using the `SetVisible` method from the `SpssDataUI` class.
- `SpssClient.StopClient()` should be called at the completion of the script. To ensure that `StopClient()` is called, it is recommended to include the call in the `finally` clause of a `try` statement--for example, by including the body of the script in a `try` statement. If the script is being run from an external Python process that starts up a new client, call `SpssClient.Exit()` before `SpssClient.StopClient()`.

Syntax

```
SpssClient.StartClient()
```

StopClient Method

Terminates the connection to the IBM SPSS Statistics client. This method should be called at the completion of each Python script. To ensure that `StopClient()` is called, it is recommended to include the call in the `finally` clause of a `try` statement--for example, by including the body of the script in a `try` statement.

Syntax

```
SpssClient.StopClient()
```

_heartBeat Method

The `_heartBeat` method is a utility function for use with thread-aware debuggers that pause all threads at a breakpoint. If you are using such a debugger, then you will need to disable the `SpssClient` heartbeat function (which is enabled by default) during debugging; otherwise the scripting session may terminate at a breakpoint due to a failed heartbeat.

Syntax

To set the heartbeat status, use:

```
SpssClient._heartBeat(status)
```

To get the heartbeat status, use:

```
SpssClient._heartBeat()
```

Parameters

status. True to enable the heartbeat function, False to disable the heartbeat function.

Returns

True. The heartbeat function is enabled.

False. The heartbeat function is disabled.

Datasets and Data Editor Windows

SpssDataDoc Class

The SpssDataDoc class represents an open dataset.

Example: Obtaining the Active Dataset

```
import SpssClient
SpssClient.StartClient()
ActiveDataDoc = SpssClient.GetActiveDataDoc()
```

- The variable *ActiveDataDoc* is an SpssDataDoc object for the active dataset.

Example: Obtaining the First Opened Dataset

```
import SpssClient
SpssClient.StartClient()
DataDocsList = SpssClient.GetDataDocuments()
FirstDataDoc = DataDocsList.GetItemAt(0)
```

- SpssClient.GetDataDocuments() returns a DataDocsList object, which provides access to all open datasets.
- The GetItemAt method from the DataDocsList class is used to get the dataset with index 0 (the first opened dataset) from the list of open datasets. The variable *FirstDataDoc* is an SpssDataDoc object for this dataset.

CloseDocument Method

Closes the dataset. If the dataset is the last open dataset then the instance of the IBM SPSS Statistics client associated with the current script is terminated.

Syntax

```
SpssDataDoc.CloseDocument()
```

GetCaseCount Method

Returns the number of cases in the dataset.

Syntax

```
SpssDataDoc.GetCaseCount()
```

GetDatasetName Method

Returns the dataset name. If the dataset is unnamed, an empty string is returned.

Syntax

```
SpssDataDoc.GetDatasetName()
```

GetDataUI Method

Returns an SpssDataUI object representing the Data Editor window of the associated dataset, if one exists.

Syntax

```
SpssDataUI=SpssDataDoc.GetDataUI()
```

GetDocumentPath Method

Returns the path and file name of the data file associated with this dataset object, or the empty string if the dataset is not associated with a file.

Syntax

```
SpssDataDoc.GetDocumentPath()
```

Note: If you reopen a data file that is currently open, the `GetDocumentPath` method will return the empty string when called on the `SpssDataDoc` object associated with the reopened instance of the file.

GetMenuTable Method

Returns a MenuTableList object containing the list of available menu items for the data document.

Syntax

```
MenuTableList = SpssDataDoc.GetMenuTable()
```

GetVariableCount Method

Returns the number of variables in the associated dataset.

Syntax

```
SpssDataDoc.GetVariableCount()
```

IsActiveDataDoc Method

Indicates if this dataset is the active one. The result is Boolean--*True* if the dataset is the active one, *False* otherwise.

Syntax

```
SpssDataDoc.IsActiveDataDoc()
```

IsEqualTo Method

Indicates if this dataset object is the same object as a specified dataset object. The result is Boolean--*True* if the two objects are identical, *False* otherwise.

Syntax

```
SpssDataDoc.IsEqualTo(dataDoc)
```

Parameters

dataDoc. An `SpssDataDoc` object

IsModified Method

Indicates whether the dataset has been modified. The result is Boolean--*True* if the dataset has been modified, *False* otherwise.

Syntax

```
SpssDataDoc.IsModified()
```

IsPromptToSave Method

Indicates if the 'prompt to save' flag is set for this dataset object. The result is Boolean--*True* if the 'prompt to save' flag has been set, *False* otherwise.

Syntax

```
SpssDataDoc.IsPromptToSave()
```

SaveAs Method

Saves the dataset to the specified file.

Syntax

```
SpssDataDoc.SaveAs(fileName,password=None)
```

Parameters

fileName. The path and file name of the data file, as a string.

password. An optional string specifying the password that will be required to open the file. Only applies if you want to encrypt the data file. Passwords are limited to 10 characters and are case-sensitive. All spaces, including leading and trailing spaces, are retained.

Note: The save operation is carried out asynchronously, which means that execution continues without waiting for the save operation to complete. If you require subsequent access to the saved file using the Python `open` function, you can attempt to open the file from a `try` block within a `while` loop, continuing to loop until the open operation succeeds.

Creating strong passwords

- Use eight or more characters.
- Include numbers, symbols and even punctuation in your password.
- Avoid sequences of numbers or characters, such as `"123"` and `"abc"`, and avoid repetition, such as `"111aaa"`.
- Do not create passwords that use personal information such as birthdays or nicknames.
- Periodically change the password.

Warning: Passwords cannot be recovered if they are lost. If the password is lost the file cannot be opened.

Note: Encrypted data files and output documents cannot be opened in versions of IBM SPSS Statistics prior to version 21. Encrypted syntax files cannot be opened in versions prior to version 22.

SetAsActiveDataDoc Method

Sets this dataset as the active one.

Syntax

```
SpssDataDoc.SetAsActiveDataDoc()
```

SetDatasetName Method

Sets the dataset name. The argument is a string.

Syntax

```
SpssDataDoc.SetDatasetName(name)
```

SetModified Method
Sets the modified status of the dataset.

Syntax

```
SpssDataDoc.SetModified(modified)
```

Parameters

modified. True to set the status to modified, False otherwise.

SetPromptToSave Method
Sets the 'prompt to save' flag for this dataset object.

Syntax

```
SpssDataDoc.SetPromptToSave(promptToSave)
```

Parameters

promptToSave. True to set the prompt to save flag, False otherwise.

DataDocsList Class
The `DataDocsList` class provides access to the list of open datasets. You obtain a `DataDocsList` object from the GetDataDocuments method of the `SpssClient` class.

A `DataDocsList` object is not an iterable Python object. In order to iterate over the items in the list, use a `for` loop, as in:

```
for index in range(DataDocsList.Size()):
```

For an example that uses the `DataDocsList` class, see the examples for the SpssDataDoc class.

GetItemAt Method
Returns an `SpssDataDoc` object representing the dataset with the specified index. The index corresponds to the order in which the datasets were opened, with the first opened document having an index of 0.

Syntax

```
SpssDataDoc=DataDocsList.GetItemAt(index)
```

Size Method
Returns the number of open datasets.

Syntax

```
DataDocsList.Size()
```

SpssDataUI Class
The `SpssDataUI` class represents the Data Editor window associated with an open dataset. You obtain an `SpssDataUI` object from the GetDataUI method of an `SpssDataDoc` object.

Example: Get the SpssDataUI Object Associated with the Active Dataset

```
import SpssClient
SpssClient.StartClient()
ActiveDataDoc = SpssClient.GetActiveDataDoc()
DataUI = ActiveDataDoc.GetDataUI()
```

- The variable *DataUI* is an `SpssDataUI` object for the Data Editor window associated with the active dataset.

GetHeight Method
Returns the height of the associated Data Editor window in units of pixels.

Syntax

```
SpssDataUI.GetHeight()
```

GetLeft Method
Returns the horizontal screen position of the associated Data Editor window's upper left corner. The result is in units of pixels.

Syntax

```
SpssDataUI.GetLeft()
```

GetShowGridLines Method
Return the setting for showing grid lines in the associated Data Editor window. The result is Boolean--*True* if grid lines are visible, *False* otherwise.

Syntax

```
SpssDataUI.GetShowGridLines()
```

GetShowValueLabels Method
Return the setting for displaying value labels in the associated Data Editor window. The result is Boolean--*True* if value labels are displayed, *False* otherwise.

Syntax

```
SpssDataUI.GetShowValueLabels()
```

GetTitleText Method
Returns the title bar text of the associated Data Editor window.

Syntax

```
SpssDataUI.GetTitleText()
```

GetTop Method
Returns the vertical screen position of the associated Data Editor window's upper left corner. The result is in units of pixels.

Syntax

```
SpssDataUI.GetTop()
```

GetVisible Method
Indicates if the associated Data Editor window is visible. The result is Boolean--*True* if the Data Editor window is visible, *False* otherwise.

Syntax

```
SpssDataUI.GetVisible()
```

GetWidth Method

Returns the width of the associated Data Editor window in units of pixels.

Syntax

```
SpssDataUI.GetWidth()
```

GetWindowState Method

Returns the state of the associated Data Editor window.

Syntax

```
SpssDataUI.GetWindowState()
```

Returns

Table 9. Window states

Value	Description
SpssClient.SpssWindowStates.SpssMinimized	Minimized
SpssClient.SpssWindowStates.SpssMaximized	Maximized
SpssClient.SpssWindowStates.SpssNormal	Normal

InvokeDialog Method

Invokes a dialog and returns the syntax generated from that dialog, if any.

Syntax

```
syntax = SpssDataUI.InvokeDialog(menuItemPath,desktopParent)
```

Parameters

menuItemPath. Menu or menu item with path of the dialog to invoke. See below for detailed description.

desktopParent. True specifies that the dialog is parented off the desktop. False specifies that the dialog is parented off an IBM SPSS Statistics window.

Note: For release 19.0.0.2 and higher, the *bSync* parameter (available in previous releases) is deprecated. The `InvokeDialog` method always runs synchronously, meaning that the scripting process waits until the dialog has been dismissed. Older scripts containing the *bSync* parameter will continue to function in release 19.0.0.2 and higher, but the value of the parameter will be ignored.

Specifying The Menu Item Path

The value of the *menuItemPath* parameter is a string specifying the menu path to the desired dialog--for example "analyze>survival>life tables". The greater-than sign (>) is used to separate a menu, its submenus and the menu item. The menu string must correspond exactly to the text on the menus, submenus, and menu items, and is language specific.

PrintDataDoc Method

Prints the document.

Syntax

```
SpssDataUI.PrintDataDoc()
```

SetHeight Method

Sets the height of the associated Data Editor window.

Syntax

```
SpssDataUI.SetHeight(height)
```

Parameters

height. An integer representing the height in pixels.

SetLeft Method

Sets the horizontal screen position of the associated Data Editor window's upper left corner.

Syntax

```
SpssDataUI.SetLeft(leftPosition)
```

Parameters

leftPosition. An integer representing the position in pixels.

SetShowGridLines Method

Specify the setting for showing grid lines in the associated Data Editor window.

Syntax

```
SpssDataUI.SetShowGridLines(isGridLines)
```

Parameters

isGridLines. True if grid lines are to be displayed, False otherwise.

SetShowValueLabels Method

Specify the setting for displaying value labels in the Data Editor window.

Syntax

```
SpssDataUI.SetShowValueLabels(isValueLabels)
```

Parameters

isValueLabels. True if value labels are to be displayed, False otherwise.

SetTop Method

Sets the vertical screen position of the associated Data Editor window's upper left corner.

Syntax

```
SpssDataUI.SetTop(topPosition)
```

Parameters

topPosition. An integer representing the position in pixels.

SetVisible Method
Sets the visibility of the associated Data Editor window.

Syntax

```
SpssDataUI.SetVisible(isVisible)
```

Parameters

isVisible. True to set the Data Editor window as visible, False otherwise.

SetWidth Method
Sets the width of the associated Data Editor window.

Syntax

```
SpssDataUI.SetWidth(width)
```

Parameters

width. An integer representing the width in pixels.

SetWindowState Method
Set the state of the associated Data Editor window.

Syntax

```
SpssDataUI.SetWindowState(newState)
```

Table 10. Window states

Value	Description
SpssClient.SpssWindowStates.SpssMinimized	Minimized
SpssClient.SpssWindowStates.SpssMaximized	Maximized
SpssClient.SpssWindowStates.SpssNormal	Normal

Output Documents and Viewer Windows

SpssOutputDoc Class
The `SpssOutputDoc` class represents an open output document.

Example: Obtaining the Designated Output Document

```
import SpssClient
SpssClient.StartClient()
DesignatedOutputDoc = SpssClient.GetDesignatedOutputDoc()
```

- The variable *DesignatedOutputDoc* is an `SpssOutputDoc` object for the designated output document.

Example: Obtaining the First Opened Output Document

```
import SpssClient
SpssClient.StartClient()
OutputDocsList = SpssClient.GetOutputDocuments()
FirstOutputDoc = OutputDocsList.GetItemAt(0)
```

- `SpssClient.GetOutputDocuments()` returns an `OutputDocsList` object, which provides access to all open output documents.

- The GetItemAt method from the OutputDocsList class is used to get the output document with index 0 (the first opened output document) from the list of open output documents. The variable *FirstOutputDoc* is an SpssOutputDoc object for this output document.

Example: Create a New Output Document and Set it as the Designated One

```
import SpssClient
SpssClient.StartClient()
NewOutputDoc = SpssClient.NewOutputDoc()
NewOutputDoc.SetAsDesignatedOutputDoc()
```

- The variable *NewOutputDoc* is an SpssOutputDoc object for the new output document.

Accessing Output Items in an Output Document

You access individual output items, within an output document, from an OutputItemList object. You obtain an OutputItemList object from the GetOutputItems method of the SpssOutputDoc class. See the topic "SpssOutputItem Class" on page 146 for more information.

ClearSelection Method

Deselects all selected output items or pivot table elements.

Syntax

```
SpssOutputDoc.ClearSelection()
```

CloseDocument Method

Closes the output document.

Syntax

```
SpssOutputDoc.CloseDocument()
```

Copy Method

Copies selected items to the clipboard. Use this method with caution because it overwrites clipboard content. To improve performance when copying large pivot tables, consider using the CopySpecial method.

To select individual items, use the SetSelected method. You can also select all items of a given type, such as all tables using the SelectAllTables method.

Syntax

```
SpssOutputDoc.Copy()
```

CopySpecial Method

Copies selected items to the clipboard in a set of specified formats. Use this method with caution because it overwrites clipboard content. This method is especially useful when copying large pivot tables since you can limit the output to just the formats you need. In that regard, the Copy method generates output in all available formats.

To select individual items, use the SetSelected method. You can also select all items of a given type, such as all tables using the SelectAllTables method.

Syntax

```
SpssOutputDoc.CopySpecial(formats)
```

Table 11. Format specifications

Format Identifier	Description
SpssClient.CopySpecialFormat.Text	plain text
SpssClient.CopySpecialFormat.Rtf	rich text format
SpssClient.CopySpecialFormat.Image	image
SpssClient.CopySpecialFormat.Emf	Windows enhanced metafile
SpssClient.CopySpecialFormat.Biff	excel worksheet in biff5 format

- The image format is a java raster image and is handled differently by different applications.
- The Windows enhanced metafile (emf) format is only supported when selecting a single output item to copy to the clipboard.

Table 12. Available formats for each output type

Type	Formats
SpssClient.OutputItemType.CHART	rich text, image or emf
SpssClient.OutputItemType.LOG	plain text, rich text or biff
SpssClient.OutputItemType.MODEL	image or emf
SpssClient.OutputItemType.NOTE	plain text, rich text, image, emf or biff
SpssClient.OutputItemType.PAGETITLE	plain text, rich text or biff
SpssClient.OutputItemType.PIVOT	plain text, rich text, image, emf or biff
SpssClient.OutputItemType.TEXT	plain text, rich text or biff
SpssClient.OutputItemType.TITLE	plain text, rich text or biff
SpssClient.OutputItemType.TREEMODEL	rich text or image
SpssClient.OutputItemType.WARNING	plain text, rich text, image, emf or biff

- If a specified format is not supported for a selected item then the format is ignored for that item. For example, you select a Log item and a Chart item and specify the plain text and image formats. The clipboard contains the Log item in plain text format and the Chart item as an image.

Example

The following copies a pivot table to the clipboard in rich text format only.

```
SpssOutputDoc.CopySpecial([SpssClient.CopySpecialFormat.Rtf])
```

CreateHeaderItem Method

Returns an `SpssOutputItem` object for a new header item. To insert the header item into the output document, use the InsertChildItem method in the `SpssHeaderItem` class.

Syntax

```
SpssOutputItem=SpssOutputDoc.CreateHeaderItem(label)
```

Parameters

label. A string specifying the label for the header item. The value can be specified as plain text, HTML, or rich text format. For HTML, embed markup in a <html></html> block. For rich text format, specify the string as a raw string to avoid unintentional escape sequences.

CreateImageChartItem Method

Returns an SpssOutputItem object for a new chart item associated with an external image. This allows you to insert an external image of type *png*, *jpg*, or *gif* into an output document. To insert the chart item into the output document, use the InsertChildItem method in the SpssHeaderItem class.

Syntax

```
SpssOutputItem=SpssOutputDoc.CreateImageChartItem(fileName,label)
```

Parameters

fileName. Full path to the image file.

label. A string specifying the label for the chart item. The value can be specified as plain text, HTML, or rich text format. For HTML, embed markup in a <html></html> block. For rich text format, specify the string as a raw string to avoid unintentional escape sequences.

On Windows, it is recommended to use raw strings for file paths, or replace backslashes with forward slashes (IBM SPSS Statistics accepts a forward slash for any backslash in a file specification). Raw strings are specified by prefacing the string with r, as in r'c:\examples\mydata.sav'. In raw mode, Python treats all backslashes in the string as the backslash character and not as the start of an escape sequence.

CreateTextItem Method

Returns an SpssOutputItem object for a new text item. To insert the text item into the output document, use the InsertChildItem method in the SpssHeaderItem class.

Syntax

```
SpssOutputItem=SpssOutputDoc.CreateTextItem(content)
```

Parameters

content. A string specifying the content of the text item. The value can be specified as plain text, HTML, or rich text format. For HTML, embed markup in a <html></html> block. For rich text format, specify the string as a raw string to avoid unintentional escape sequences.

CreateTitleItem Method

Returns an SpssOutputItem object for a new title item. To insert the title item into the output document, use the InsertChildItem method in the SpssHeaderItem class.

Syntax

```
SpssOutputItem=SpssOutputDoc.CreateTitleItem(title,pageBreak)
```

Parameters

title. A string specifying the title. The value can be specified as plain text, HTML, or rich text format. For HTML, embed markup in a <html></html> block. For rich text format, specify the string as a raw string to avoid unintentional escape sequences.

pageBreak. True if this title item is to be a page title item, False otherwise.

Cut Method

Removes the selected data or text and places them on the clipboard. Use this method with caution because it overwrites clipboard content.

Syntax

```
SpssOutputDoc.Cut()
```

Delete Method

Deletes the selected items.

Syntax

```
SpssOutputDoc.Delete()
```

Demote Method

Demotes selected output items down one level within the hierarchy of the output tree.

- You cannot demote an item that is at the deepest level in the output tree and you cannot demote an item if there are unselected items at the same level immediately preceding it in the output tree.
- If the item has children, the children are also demoted.
- You cannot demote the root item.

Syntax

```
SpssOutputDoc.Demote()
```

You can promote items up one level with the Promote method.

ExportCharts Method

Exports charts from this output document.

Syntax

```
SpssOutputDoc.ExportCharts(subSet,filePrefix,format)
```

Parameters

subSet. Specifies whether all charts, all visible charts, or all selected charts are exported. See available choices below.

filePrefix. Full path and file name prefix for the files containing the exported charts. Each chart is exported to a separate file.

format. Specifies the export format. See available choices below.

On Windows, it is recommended to use raw strings for file paths, or replace backslashes with forward slashes (IBM SPSS Statistics accepts a forward slash for any backslash in a file specification). Raw strings are specified by prefacing the string with r, as in r'c:\examples\mydata.sav'. In raw mode, Python treats all backslashes in the string as the backslash character and not as the start of an escape sequence.

Table 13. Subset specifications

Value	Description
SpssClient.SpssExportSubset.SpssSelected	All selected charts
SpssClient.SpssExportSubset.SpssVisible	All visible charts
SpssClient.SpssExportSubset.SpssAll	All charts

Table 14. Image formats

Format	Description
SpssClient.ChartExportFormat.bmp	Windows bitmap
SpssClient.ChartExportFormat.emf	Enhanced metafile
SpssClient.ChartExportFormat.eps	Enhanced postscript
SpssClient.ChartExportFormat.jpg	JPG file
SpssClient.ChartExportFormat.png	PNG file
SpssClient.ChartExportFormat.tiff	Tagged image file

ExportDocument Method

Exports items from this output document.

- If the items to be exported include charts, then they are exported in the last selected graphics format. The graph export type can be set from the `SetExportOption` method in the `SpssClient` class.
- Use the SetOutputOptions method to set export options for export to Word, Excel, or PowerPoint.

Syntax

```
SpssOutputDoc.ExportDocument(subSet,fileName,format)
```

Parameters

subSet. Specifies whether all items, all visible items, or all selected items are exported. See available choices below.

fileName. Full path and file name for the file containing the exported items.

format. Specifies the export format. See available choices below.

On Windows, it is recommended to use raw strings for file paths, or replace backslashes with forward slashes (IBM SPSS Statistics accepts a forward slash for any backslash in a file specification). Raw strings are specified by prefacing the string with r, as in r'c:\examples\mydata.sav'. In raw mode, Python treats all backslashes in the string as the backslash character and not as the start of an escape sequence.

Table 15. Specifications for subsets

Value	Description
SpssClient.SpssExportSubset.SpssSelected	All selected items
SpssClient.SpssExportSubset.SpssVisible	All visible items
SpssClient.SpssExportSubset.SpssAll	All items

Table 16. Export formats

Format	Description
SpssClient.DocExportFormat.SpssFormatHtml	Html
SpssClient.DocExportFormat.SpssFormatDoc	Word
SpssClient.DocExportFormat.SpssFormatXls	Excel
SpssClient.DocExportFormat.SpssFormatText	Text
SpssClient.DocExportFormat.SpssFormatPdf	PDF
SpssClient.DocExportFormat.SpssFormatPpt	PowerPoint

GetCurrentItem Method

Returns an SpssOutputItem object for the current output item--as indicated by a red arrow next to the item in the outline pane.

Syntax

```
SpssOutputItem=SpssOutputDoc.GetCurrentItem()
```

GetDocumentPath Method

Returns the path and file name of the output file associated with this output document object, or the empty string if the output document is not associated with a file.

Syntax

```
SpssOutputDoc.GetDocumentPath()
```

GetFooterText Method

Returns the footer text for printed pages. The value is returned as plain text.

Syntax

```
SpssOutputDoc.GetFooterText()
```

GetHeaderText Method

Returns the header text for printed pages. The value is returned as plain text.

Syntax

```
SpssOutputDoc.GetHeaderText()
```

GetMenuTable Method

Returns a MenuTableList object containing the list of available menu items for the output document.

Syntax

```
MenuTableList = SpssOutputDoc.GetMenuTable()
```

GetOutputItems Method

Returns a list of items in the output document as an OutputItemList object. Each item in the list is an SpssOutputItem object.

Syntax

```
OutputItemList=SpssOutputDoc.GetOutputItems()
```

GetOutputOptions Method

Returns the value of the specified export option for this output document, as a string.

Syntax

```
SpssOutputDoc.GetOutputOptions(option)
```

Parameters

The available values for the *option* parameter are (specify the value without quotes):

SpssClient.DocExportOption.ExcelSheetNames. Specifies the name of the sheet to which items will be exported. This option only applies when exporting to Excel.

SpssClient.DocExportOption.ExcelStartingCell. Specifies the starting cell for exporting to Excel. Applies when `SpssClient.DocExportOption.ExcelLocationOptions` is set to “OverwriteAtCellRef”.

SpssClient.DocExportOption.ExcelOperationOptions. Specifies whether a new workbook is created, a new worksheet is created, or an existing worksheet is modified. This option only applies when exporting to Excel.

- **"CreateWorkbook".** A new workbook is created. If the specified file exists, it is overwritten.
- **"CreateWorksheet".** A new worksheet is created within the specified workbook. The name of the sheet is given by the setting of `SpssClient.DocExportOption.ExcelSheetNames`. If a worksheet with the specified name already exists, that worksheet is overwritten. If the specified file does not exist, a new file is created with a worksheet with the specified name.
- **"ModifyWorksheet".** Modifies the contents of an existing worksheet. The name of the sheet is given by the setting of `SpssClient.DocExportOption.ExcelSheetNames`. Export of charts, model views, and tree diagrams is not supported with "ModifyWorksheet".

SpssClient.DocExportOption.ExcelLocationOptions. Specifies how items will be added to a worksheet. This option only applies when exporting to Excel.

- **"AddColumns".** Specifies that items will be added after the last column, starting in the first row, without modifying any existing contents.
- **"AddRows".** Specifies that items will be added after the last row, starting in the first column, without modifying any existing contents.
- **"OverwriteAtCellRef".** Specifies that items will be written to the location specified in `SpssClient.DocExportOption.ExcelStartingCell`. Any existing content in the area where the exported items are added will be overwritten.

SpssClient.DocExportOption.WideTablesOptions. Specifies the treatment of pivot tables that are too wide for the document width (the specified page width minus the left and right margins). This option only applies when exporting to Word or PowerPoint.

- **"WT_Wrap".** Specifies that tables are divided into sections that will fit within the defined document width. Row labels are repeated for each section of the table. If the row labels are too wide for the defined document width, the table is exported without wrapping and will appear truncated in the document.
- **"WT_Shrink".** Specifies that font size and column width are reduced so that tables fit within the document width.
- **"WT_Extend".** Specifies that tables that are too wide for the document width will appear truncated. All of the table content, however, is retained so expanding the document width will display additional table content.

The following options apply when exporting to Word or PowerPoint.

SpssClient.DocExportOption.ItemsPageHeight. A character representation of a positive number representing the page height, in units specified by `SpssClient.DocExportOption.ItemsMeasurementUnits`.

SpssClient.DocExportOption.ItemsPageWidth. A character representation of a positive number representing the page width, in units specified by `SpssClient.DocExportOption.ItemsMeasurementUnits`.

SpssClient.DocExportOption.ItemsTopMargin. A character representation of a positive number representing the top margin, in units specified by `SpssClient.DocExportOption.ItemsMeasurementUnits`.

SpssClient.DocExportOption.ItemsBottomMargin. A character representation of a positive number representing the bottom margin, in units specified by `SpssClient.DocExportOption.ItemsMeasurementUnits`.

SpssClient.DocExportOption.ItemsRightMargin. A character representation of a positive number representing the right margin, in units specified by `SpssClient.DocExportOption.ItemsMeasurementUnits`.

SpssClient.DocExportOption.ItemsLeftMargin. A character representation of a positive number representing the left margin, in units specified by `SpssClient.DocExportOption.ItemsMeasurementUnits`.

SpssClient.DocExportOption.ItemsMeasurementUnits. The units for specifying page dimensions and margins: "IExportOptions.MeasurementUnits.Inches", "IExportOptions.MeasurementUnits.Millimeters", "IExportOptions.MeasurementUnits.Centimeters", and "IExportOptions.MeasurementUnits.PrintPoints" (1/72 inch).

GetOutputUI Method

Returns an `SpssOutputUI` object representing the Viewer window associated with the output document, if one exists.

Syntax

```
SpssOutputUI=SpssOutputDoc.GetOutputUI()
```

GetPrintOptions Method

Returns the value of the specified print option, as a string.

Syntax

```
SpssOutputDoc.GetPrintOptions(printOption)
```

Table 17. Print options

Option	Description
SpssClient.PrintOptions.LeftMargin	Left margin
SpssClient.PrintOptions.TopMargin	Top margin
SpssClient.PrintOptions.RightMargin	Right margin
SpssClient.PrintOptions.BottomMargin	Bottom margin
SpssClient.PrintOptions.Orientation	Orientation (portrait or landscape)
SpssClient.PrintOptions.StartingPageNumber	Starting page number
SpssClient.PrintOptions.SpaceBetweenItems	Space between items
SpssClient.PrintOptions.PrintedChartSize	Printed chart size (as is, full page, half page, or quarter page)

- All margin settings and Space Between Items are in units of points (1/72 inch).
- For Orientation, 1 corresponds to Portrait and 2 corresponds to Landscape.
- For Printed Chart Size, 0 corresponds to As Is, 1 to Full Page, 2 to Half Page, and 3 to Quarter Page.

InsertTable Method

Inserts an empty pivot table after the item designated as the current item. The inserted table is populated with default row, column and layer labels, and becomes the current item. *Note*: You can use the SetCurrentItem method to designate an item as the current item.

Syntax

```
index=SpssOutputDoc.InsertTable(heading,nrows,ncolumns,nlayers)
```

Parameters

heading. A string specifying the heading for this table in the outline pane of the Viewer.

nrows. An integer specifying the number of rows in the table. Specifying zero will result in a table with one row.

ncolumns. An integer specifying the number of columns in the table. Specifying zero will result in a table with one column.

nlayers. An integer specifying the number of layers in the table.

Return Value

index. The index of the new table item. The index corresponds to the order of the items in the output document, starting with 0 for the root item.

Example

This example inserts a pivot table with four rows, three columns and no layers. The table is inserted after the root item.

```
import SpssClient
SpssClient.StartClient()
OutputDoc = SpssClient.GetDesignatedOutputDoc()
OutputItems = OutputDoc.GetOutputItems()
OutputItem = OutputItems.GetItemAt(0)
OutputItem.SetCurrentItem()
index = OutputDoc.InsertTable("Sample table",4,3,0)
```

IsDesignatedOutputDoc Method

Indicates if this output document is the designated one. The result is Boolean--*True* if the output document is the designated one, *False* otherwise.

- If you have more than one open output document, output is routed to the designated one.

Syntax

```
SpssOutputDoc.IsDesignatedOutputDoc()
```

IsEqualTo Method

Indicates if this output document object is the same object as a specified output document object. The result is Boolean--*True* if the two objects are identical, *False* otherwise.

Syntax

```
SpssOutputDoc.IsEqualTo(outputDoc)
```

Parameters

outputDoc. An `SpssOutputDoc` object

IsModified Method

Indicates whether the output document has been modified. The result is Boolean--*True* if the output document has been modified, *False* otherwise.

Syntax

```
SpssOutputDoc.IsModified()
```

IsPromptToSave Method

Indicates if the 'prompt to save' flag is set for this output document object. The result is Boolean--*True* if the 'prompt to save' flag has been set, *False* otherwise.

Syntax

```
SpssOutputDoc.IsPromptToSave()
```

Paste Method

Pastes the clipboard content after the current item.

Syntax

```
SpssOutputDoc.Paste()
```

PasteBefore Method

Pastes the clipboard content before the current item.

Syntax

```
SpssOutputDoc.PasteBefore()
```

PrintRange Method

Sets the print range for the output document.

Syntax

```
SpssOutputDoc.PrintRange(range)
```

Parameters

range. An integer specifying the print range: 0 for all expanded output, 1 for all selected items.

You can specify print options using the SetPrintOptions method. You print an output document using the PrintOutputDoc method from the `SpssOutputUI` class.

Promote Method

Promotes selected output items up one level within the hierarchy of the output tree.

- You cannot promote an item to the root level and you cannot promote an item if there are unselected items at the same level immediately following it in the output tree.
- If the item has children, the children are also promoted.

Syntax

```
SpssOutputDoc.Promote()
```

You can demote items down one level with the Demote method.

SaveAs Method

Saves the output document to the specified file.

Syntax

```
SpssOutputDoc.SaveAs(fileName,password=None)
```

Parameters

fileName. The path and file name of the output file, as a string.

password. An optional string specifying the password that will be required to open the file. Only applies if you want to encrypt the output file. Passwords are limited to 10 characters and are case-sensitive. All spaces, including leading and trailing spaces, are retained.

Creating strong passwords

- Use eight or more characters.
- Include numbers, symbols and even punctuation in your password.
- Avoid sequences of numbers or characters, such as `"123"` and `"abc"`, and avoid repetition, such as `"111aaa"`.
- Do not create passwords that use personal information such as birthdays or nicknames.
- Periodically change the password.

Warning: Passwords cannot be recovered if they are lost. If the password is lost the file cannot be opened.

Note: Encrypted data files and output documents cannot be opened in versions of IBM SPSS Statistics prior to version 21. Encrypted syntax files cannot be opened in versions prior to version 22.

SelectAll Method

Selects all items in the output document.

Syntax

```
SpssOutputDoc.SelectAll()
```

SelectAllCharts Method

Selects all chart items in the output document. This includes standard charts, graphboard charts, and R graphics.

Syntax

```
SpssOutputDoc.SelectAllCharts()
```

SelectAllLogs Method

Selects all log items in the output document.

Syntax

```
SpssOutputDoc.SelectAllLogs()
```

SelectAllModels Method

Selects all Model Viewer items in the output document.

Syntax

```
SpssOutputDoc.SelectAllModels()
```

SelectAllNotes Method

Selects all notes items in the output document.

Syntax

```
SpssOutputDoc.SelectAllNotes()
```

SelectAllNotesEx Method

Note: This method is deprecated for release 20 and higher. Please use the "SelectAllNotes Method" on page 129 instead.

Selects all notes items in the output document.

Syntax

```
SpssOutputDoc.SelectAllNotesEx()
```

SelectAllOther Method

Selects all non-IBM SPSS Statistics items in the output document.

Syntax

```
SpssOutputDoc.SelectAllOther()
```

SelectAllTables Method

Selects all pivot tables in the output document.

Syntax

```
SpssOutputDoc.SelectAllTables()
```

SelectAllTablesEx Method

Note: This method is deprecated for release 20 and higher. Please use the "SelectAllTables Method" instead.

Selects all pivot tables in the output document.

Syntax

```
SpssOutputDoc.SelectAllTablesEx()
```

SelectAllText Method

Selects all text items in the output document.

Syntax

```
SpssOutputDoc.SelectAllText()
```

SelectAllTitles Method

Selects all title items in the output document.

Syntax

```
SpssOutputDoc.SelectAllTitles()
```

SelectAllWarnings Method

Selects all warnings in the output document.

Syntax

```
SpssOutputDoc.SelectAllWarnings()
```

SelectAllWarningsEx Method

Note: This method is deprecated for release 20 and higher. Please use the "SelectAllWarnings Method" on page 130 instead.

Selects all warnings in the output document.

Syntax

```
SpssOutputDoc.SelectAllWarningsEx()
```

SelectLastOutput Method

Selects all items generated by the last executed procedure.

Syntax

```
SpssOutputDoc.SelectLastOutput()
```

SetAsDesignatedOutputDoc Method

Sets this output document as the designated output document.

- If you have more than one open output document, output is routed to the designated one.

Syntax

```
SpssOutputDoc.SetAsDesignatedOutputDoc()
```

SetFooterText Method

Sets the footer text for printed pages. The value can be specified as plain text, HTML, or rich text format. For HTML, embed markup in a <html></html> block. For rich text format, specify the string as a raw string to avoid unintentional escape sequences.

Syntax

```
SpssOutputDoc.SetFooterText(text)
```

SetHeaderText Method

Sets the header text for printed pages. The value can be specified as plain text, HTML, or rich text format. For HTML, embed markup in a <html></html> block. For rich text format, specify the string as a raw string to avoid unintentional escape sequences.

Syntax

```
SpssOutputDoc.SetHeaderText(text)
```

SetModified Method

Sets the modified status of the output document.

Syntax

```
SpssOutputDoc.SetModified(modified)
```

Parameters

modified. True to set the status to modified, False otherwise.

SetOutputOptions Method

Sets export options for this output document. These options apply when exporting with the `ExportDocument` method from the `SpssOutputDoc` class as well as the `ExportToDocument` method from the `SpssOutputItem` class.

Syntax

```
SpssOutputDoc.SetOutputOptions(option,setting)
```

Parameters

The available values for the *option* parameter as well as the allowed values of the associated *setting* are (specify the name of the option without quotes):

SpssClient.DocExportOption.ExcelSheetNames. The setting is a string specifying the name of the sheet to which items will be exported. Sheet names cannot exceed 31 characters and cannot contain forward or back slashes, square brackets, question marks, or asterisks. If the specified worksheet doesn't exist in the Excel file, a new worksheet with that name will be created. This option only applies when exporting to Excel. The default worksheet is "Sheet1".

SpssClient.DocExportOption.ExcelStartingCell. The setting is a string specifying the starting cell, as in "B3". Applies when `SpssClient.DocExportOption.ExcelLocationOptions` is set to "OverwriteAtCellRef". Only applies when exporting to Excel. The default starting cell is A1.

SpssClient.DocExportOption.ExcelOperationOptions. The setting is a string specifying whether a new workbook is created, a new worksheet is created, or an existing worksheet is modified. This option only applies when exporting to Excel.

- **"CreateWorkbook".** A new workbook is created. This is the default. If the specified file exists, it is overwritten.
- **"CreateWorksheet".** A new worksheet is created within the specified workbook. The name of the sheet is given by the setting of `SpssClient.DocExportOption.ExcelSheetNames`. If a worksheet with the specified name already exists, that worksheet is overwritten. If the specified file does not exist, a new file is created with a worksheet with the specified name.
- **"ModifyWorksheet".** Modifies the contents of an existing worksheet. The name of the sheet is given by the setting of `SpssClient.DocExportOption.ExcelSheetNames`. Use `SpssClient.DocExportOption.ExcelLocationOptions` and `SpssClient.DocExportOption.ExcelStartingCell` to specify location in the sheet. Export of charts, model views, and tree diagrams is not supported with "ModifyWorksheet".

SpssClient.DocExportOption.ExcelLocationOptions. A string specifying how items will be added to a worksheet. This option only applies when exporting to Excel.

- **"AddColumns".** Specifies that items will be added after the last column, starting in the first row, without modifying any existing contents. This is the default.
- **"AddRows".** Specifies that items will be added after the last row, starting in the first column, without modifying any existing contents.
- **"OverwriteAtCellRef".** Specifies that items will be written to the location specified in `SpssClient.DocExportOption.ExcelStartingCell`. Any existing content in the area where the exported items are added will be overwritten.

SpssClient.DocExportOption.WideTablesOptions. A string specifying the treatment of pivot tables that are too wide for the document width (the specified page width minus the left and right margins). This option only applies when exporting to Word or PowerPoint.

- **"WT_Wrap".** Specifies that tables are divided into sections that will fit within the defined document width. This is the default. Row labels are repeated for each section of the table. If the row labels are too wide for the defined document width, the table is exported without wrapping and will appear truncated in the document.
- **"WT_Shrink".** Specifies that font size and column width are reduced so that tables fit within the document width.
- **"WT_Extend".** Specifies that tables that are too wide for the document width will appear truncated. All of the table content, however, is retained so expanding the document width will display additional table content.

The following options apply when exporting to Word or PowerPoint.

SpssClient.DocExportOption.ItemsPageHeight. A character representation of a positive number representing the page height, in units specified by `SpssClient.DocExportOption.ItemsMeasurementUnits`.

SpssClient.DocExportOption.ItemsPageWidth. A character representation of a positive number representing the page width, in units specified by `SpssClient.DocExportOption.ItemsMeasurementUnits`.

SpssClient.DocExportOption.ItemsTopMargin. A character representation of a positive number representing the top margin, in units specified by `SpssClient.DocExportOption.ItemsMeasurementUnits`.

SpssClient.DocExportOption.ItemsBottomMargin. A character representation of a positive number representing the bottom margin, in units specified by `SpssClient.DocExportOption.ItemsMeasurementUnits`.

SpssClient.DocExportOption.ItemsRightMargin. A character representation of a positive number representing the right margin, in units specified by `SpssClient.DocExportOption.ItemsMeasurementUnits`.

SpssClient.DocExportOption.ItemsLeftMargin. A character representation of a positive number representing the left margin, in units specified by `SpssClient.DocExportOption.ItemsMeasurementUnits`.

SpssClient.DocExportOption.ItemsMeasurementUnits. A string specifying the units for page dimensions and margins: "IExportOptions.MeasurementUnits.Inches", "IExportOptions.MeasurementUnits.Millimeters", "IExportOptions.MeasurementUnits.Centimeters", and "IExportOptions.MeasurementUnits.PrintPoints" (1/72 inch). The default is "IExportOptions.MeasurementUnits.Inches".

Example

This example assumes that `OutputDoc` is an `SpssOutputDoc` object and exports all pivot tables to an existing Excel worksheet beginning at a specified location.

```
OutputDoc = SpssClient.GetDesignatedOutputDoc()
OutputDoc.SetOutputOptions(SpssClient.DocExportOption.ExcelSheetNames,"mysheet")
OutputDoc.SetOutputOptions(SpssClient.DocExportOption.ExcelStartingCell,"B6")
OutputDoc.SetOutputOptions(SpssClient.DocExportOption.ExcelLocationOptions,
                           "OverwriteAtCellRef")
OutputDoc.SetOutputOptions(SpssClient.DocExportOption.ExcelOperationOptions,
                           "ModifyWorksheet")
OutputItems = OutputDoc.GetOutputItems()
for index in range(OutputItems.Size()):
    OutputItem = OutputItems.GetItemAt(index)
    if OutputItem.GetType() == SpssClient.OutputItemType.PIVOT:
        OutputItem.SetSelected(True)
OutputDoc.ExportDocument(SpssClient.SpssExportSubset.SpssSelected,
                         "/output/myexport.xls",
                         SpssClient.DocExportFormat.SpssFormatXls)
```

SetPrintOptions Method

Sets the value of the specified print option.

Syntax

```
SpssOutputDoc.SetPrintOptions(printOption,value)
```

Table 18. Print options

Option	Description
SpssClient.PrintOptions.LeftMargin	Left margin
SpssClient.PrintOptions.TopMargin	Top margin
SpssClient.PrintOptions.RightMargin	Right margin
SpssClient.PrintOptions.BottomMargin	Bottom margin
SpssClient.PrintOptions.Orientation	Orientation (portrait or landscape)
SpssClient.PrintOptions.StartingPageNumber	Starting page number
SpssClient.PrintOptions.SpaceBetweenItems	Space between items
SpssClient.PrintOptions.PrintedChartSize	Printed chart size (as is, full page, half page, or quarter page)

The parameter *value* is a string. Following are the available settings:

- All margin settings and Space Between Items are in units of points (1/72 inch).
- For Orientation, 1 corresponds to Portrait and 2 corresponds to Landscape.
- For Printed Chart Size, 0 corresponds to As Is, 1 to Full Page, 2 to Half Page, and 3 to Quarter Page.

You can specify the print range using the PrintRange method. You print an output document using the PrintOutputDoc method from the `SpssOutputUI` class.

SetPromptToSave Method

Sets the 'prompt to save' flag for this output document object.

Syntax

```
SpssOutputDoc.SetPromptToSave(promptToSave)
```

Parameters

promptToSave. True to set the prompt to save flag, False otherwise.

OutputDocsList Class

The `OutputDocsList` class provides access to the list of open output documents. You obtain an `OutputDocsList` object from the GetOutputDocuments method of the `SpssClient` class.

An `OutputDocsList` object is not an iterable Python object. In order to iterate over the items in the list, use a `for` loop, as in:

```
for index in range(OutputDocsList.Size()):
```

For an example that uses the `OutputDocsList` class, see the examples for the SpssOutputDoc class.

GetItemAt Method

Returns an SpssOutputDoc object representing the output document with the specified index. The index corresponds to the order in which the output documents were opened, with the first opened document having an index of 0.

Syntax

```
SpssOutputDoc=OutputDocsList.GetItemAt(index)
```

Size Method

Returns the number of open output documents.

Syntax

```
OutputDocsList.Size()
```

OutputItemList Class

The OutputItemList class provides access to the list of items in an open output document. You obtain an OutputItemList object from the GetOutputItems method of an SpssOutputDoc object.

An OutputItemList object is not an iterable Python object. In order to iterate over the items in the list, use a for loop, as in:

```
for index in range(OutputItemList.Size()):
```

For an example that uses the OutputItemList class, see the example for the SpssOutputItem class.

GetItemAt Method

Returns an SpssOutputItem object corresponding to the output item with the specified index. The index corresponds to the order of the items in the output document, starting with 0 for the root item.

Syntax

```
SpssOutputItem=OutputItemList.GetItemAt(index)
```

Size Method

Returns the number of items in the associated output document.

Syntax

```
OutputItemList.Size()
```

SpssOutputUI Class

The SpssOutputUI class represents the Viewer window associated with an open output document. You obtain an SpssOutputUI object from the GetOutputUI method of an SpssOutputDoc object.

Example: Get the SpssOutputUI Object Associated with the Designated Output Document

```
import SpssClient
SpssClient.StartClient()
DesignatedOutputDoc = SpssClient.GetDesignatedOutputDoc()
OutputUI = DesignatedOutputDoc.GetOutputUI()
```

- The variable *OutputUI* is an SpssOutputUI object for the Viewer window associated with the designated output document.

GetHeight Method
Returns the height of the associated Viewer window in units of pixels.

Syntax

```
SpssOutputUI.GetHeight()
```

GetLeft Method
Returns the horizontal screen position of the associated Viewer window's upper left corner. The result is in units of pixels.

Syntax

```
SpssOutputUI.GetLeft()
```

GetSplitterPosition Method
Returns the position of the splitter bar in the associated Viewer window. The result is in units of pixels. The splitter bar determines how large the outline area is.

Syntax

```
SpssOutputUI.GetSplitterPosition()
```

GetTitleText Method
Returns the title bar text of the associated Viewer window.

Syntax

```
SpssOutputUI.GetTitleText()
```

GetTop Method
Returns the vertical screen position of the associated Viewer window's upper left corner. The result is in units of pixels.

Syntax

```
SpssOutputUI.GetTop()
```

GetVisible Method
Indicates if the associated Viewer window is visible. The result is Boolean--*True* if the Viewer window is visible, *False* otherwise.

Syntax

```
SpssOutputUI.GetVisible()
```

GetWidth Method
Returns the width of the associated Viewer window in units of pixels.

Syntax

```
SpssOutputUI.GetWidth()
```

GetWindowState Method

Returns the state of the associated Viewer window.

Syntax

```
SpssOutputUI.GetWindowState()
```

Returns

Table 19. Window states

Value	Description
SpssClient.SpssWindowStates.SpssMinimized	Minimized
SpssClient.SpssWindowStates.SpssMaximized	Maximized
SpssClient.SpssWindowStates.SpssNormal	Normal

InvokeDialog Method

Invokes a dialog and returns the syntax generated from that dialog, if any.

Syntax

```
syntax = SpssOutputUI.InvokeDialog(menuItemPath,desktopParent)
```

Parameters

menuItemPath. Menu or menu item with path of the dialog to invoke. See below for detailed description.

desktopParent. True specifies that the dialog is parented off the desktop. False specifies that the dialog is parented off an IBM SPSS Statistics window.

Note: For release 19.0.0.2 and higher, the *bSync* parameter (available in previous releases) is deprecated. The `InvokeDialog` method always runs synchronously, meaning that the scripting process waits until the dialog has been dismissed. Older scripts containing the *bSync* parameter will continue to function in release 19.0.0.2 and higher, but the value of the parameter will be ignored.

Specifying The Menu Item Path

The value of the *menuItemPath* parameter is a string specifying the menu path to the desired dialog--for example "analyze>survival>life tables". The greater-than sign (>) is used to separate a menu, its submenus and the menu item. The menu string must correspond exactly to the text on the menus, submenus, and menu items, and is language specific.

PrintOutputDoc Method

Prints the document.

Syntax

```
SpssOutputUI.PrintOutputDoc()
```

You can specify the print range using the PrintRange method. You can specify print options using the SetPrintOptions method.

SetHeight Method

Sets the height of the associated Viewer window.

Syntax

```
SpssOutputUI.SetHeight(height)
```

Parameters

height. An integer representing the height in pixels.

SetLeft Method

Sets the horizontal screen position of the associated Viewer window's upper left corner.

Syntax

```
SpssOutputUI.SetLeft(leftPosition)
```

Parameters

leftPosition. An integer representing the position in pixels.

SetSplitterPosition Method

Sets the position of the splitter bar in the associated Viewer window. The splitter bar determines how large the outline area is.

Syntax

```
SpssOutputUI.SetSplitterPosition(position)
```

Parameters

position. An integer representing the position in pixels.

SetTop Method

Sets the vertical screen position of the associated Viewer window's upper left corner.

Syntax

```
SpssOutputUI.SetTop(topPosition)
```

Parameters

topPosition. An integer representing the position in pixels.

SetVisible Method

Sets the visibility of the associated Viewer window.

Syntax

```
SpssOutputUI.SetVisible(isVisible)
```

Parameters

isVisible. True to set the Viewer window as visible, False otherwise.

SetWidth Method

Sets the width of the associated Viewer window.

Syntax

```
SpssOutputUI.SetWidth(width)
```

Parameters

width. An integer representing the width in pixels.

SetWindowState Method

Sets the state of the associated Viewer window.

Syntax

```
SpssOutputUI.SetWindowState(newState)
```

Table 20. Window states

Value	Description
SpssClient.SpssWindowStates.SpssMinimized	Minimized
SpssClient.SpssWindowStates.SpssMaximized	Maximized
SpssClient.SpssWindowStates.SpssNormal	Normal

Syntax Documents and Syntax Editor Windows

SpssSyntaxDoc Class

The `SpssSyntaxDoc` class represents an open syntax document.

Example: Obtaining the Designated Syntax Document

```
import SpssClient
SpssClient.StartClient()
DesignatedSyntaxDoc = SpssClient.GetDesignatedSyntaxDoc()
```

- The variable *DesignatedSyntaxDoc* is an `SpssSyntaxDoc` object for the designated syntax document.

Example: Obtaining the First Opened Syntax Document

```
import SpssClient
SpssClient.StartClient()
SyntaxDocsList = SpssClient.GetSyntaxDocuments()
FirstSyntaxDoc = SyntaxDocsList.GetItemAt(0)
```

- `SpssClient.GetSyntaxDocuments()` returns a `SyntaxDocsList` object, which provides access to all open syntax documents.
- The `GetItemAt` method from the `SyntaxDocsList` class is used to get the syntax document with index 0 (the first opened syntax document) from the list of open syntax documents. The variable *FirstSyntaxDoc* is an `SpssSyntaxDoc` object for this syntax document.

Example: Create a New Syntax Document and Set it as the Designated One

```
import SpssClient
SpssClient.StartClient()
NewSyntaxDoc = SpssClient.NewSyntaxDoc()
NewSyntaxDoc.SetAsDesignatedSyntaxDoc()
```

- The variable *NewSyntaxDoc* is an `SpssSyntaxDoc` object for the new syntax document.

CloseDocument Method

Closes the syntax document.

Syntax

```
SpssSyntaxDoc.CloseDocument()
```

GetDocumentPath Method

Returns the path and file name of the syntax file associated with this syntax document object, or the empty string if the syntax document is not associated with a file.

Syntax

```
SpssSyntaxDoc.GetDocumentPath()
```

GetMenuTable Method

Returns a MenuTableList object containing the list of available menu items for the syntax document.

Syntax

```
MenuTableList = SpssSyntaxDoc.GetMenuTable()
```

GetSyntax Method

Returns the syntax contained in the associated syntax document, as a unicode string.

Syntax

```
SpssSyntaxDoc.GetSyntax()
```

GetSyntaxUI Method

Returns an `SpssSyntaxUI` object representing the syntax window associated with the syntax document.

Syntax

```
SpssSyntaxUI=SpssSyntaxDoc.GetSyntaxUI()
```

IsDesignatedSyntaxDoc Method

Indicates if this syntax document is the designated one. The result is Boolean--*True* if the syntax document is the designated one, *False* otherwise.

- If you have more than one open syntax document, command syntax is pasted into the designated one.

Syntax

```
SpssSyntaxDoc.IsDesignatedSyntaxDoc()
```

IsEqualTo Method

Indicates if this syntax document object is the same object as a specified syntax document object. The result is Boolean--*True* if the two objects are identical, *False* otherwise.

Syntax

```
SpssSyntaxDoc.IsEqualTo(syntaxDoc)
```

Parameters

syntaxDoc. An `SpssSyntaxDoc` object

IsModified Method

Indicates whether the syntax document has been modified. The result is Boolean--*True* if the syntax document has been modified, *False* otherwise.

Syntax

```
SpssSyntaxDoc.IsModified()
```

IsPromptToSave Method

Indicates if the 'prompt to save' flag is set for this syntax document object. The result is Boolean--*True* if the 'prompt to save' flag has been set, *False* otherwise.

Syntax

```
SpssSyntaxDoc.IsPromptToSave()
```

RunSyntax Method

Runs all of the syntax in the associated syntax document.

- The submitted commands are executed synchronously with any other submitted command syntax.
- This method cannot be called within a script that is run from the `SCRIPT` command. It is also not available when called from a Python program in distributed mode (Python programs make use of the interface exposed by the Python `spss` module).

Syntax

```
SpssSyntaxDoc.RunSyntax()
```

SaveAs Method

Saves the syntax document to the specified file.

Syntax

```
SpssSyntaxDoc.SaveAs(fileName,password=None)
```

Parameters

fileName. The path and file name of the syntax file, as a string. If you are encrypting the file with a password then specify .spsx as the file extension.

password. An optional string that specifies the password that is required to open the file. This setting applies only if you want to encrypt the syntax file. Passwords are limited to 10 characters and are case-sensitive. All spaces, including leading and trailing spaces, are retained.

Creating strong passwords

- Use eight or more characters.
- Include numbers, symbols and even punctuation in your password.
- Avoid sequences of numbers or characters, such as `"123"` and `"abc"`, and avoid repetition, such as `"111aaa"`.
- Do not create passwords that use personal information such as birthdays or nicknames.
- Periodically change the password.

Warning: Passwords cannot be recovered if they are lost. If the password is lost the file cannot be opened.

Note: Encrypted data files and output documents cannot be opened in versions of IBM SPSS Statistics prior to version 21. Encrypted syntax files cannot be opened in versions prior to version 22.

SetAsDesignatedSyntaxDoc Method

Sets this syntax document as the designated syntax document.

- If you have more than one open syntax document, command syntax is pasted into the designated one.

Syntax

```
SpssSyntaxDoc.SetAsDesignatedSyntaxDoc()
```

SetModified Method

Sets the modified status of the syntax document.

Syntax

```
SpssSyntaxDoc.SetModified(modified)
```

Parameters

modified. True to set the status to modified, False otherwise.

SetPromptToSave Method

Sets the 'prompt to save' flag for this syntax document object.

Syntax

```
SpssSyntaxDoc.SetPromptToSave(promptToSave)
```

Parameters

promptToSave. True to set the prompt to save flag, False otherwise.

SetSyntax Method

Specifies the content of the associated syntax document and replaces any existing content.

Syntax

```
SpssSyntaxDoc.SetSyntax(syntax)
```

Parameters

syntax. A string specifying the syntax. You can include line breaks using the escape sequence "\n", and you can use a triple-quoted string as shown in the example.

Example

```
SpssSyntaxDoc.SetSyntax(r"""DESCRIPTIVES
  VARIABLES=salary salbegin jobtime prevexp
  /STATISTICS=MEAN STDDEV MIN MAX.""")
```

- Using a triple-quoted string--as in this example--allows you to specify a block of syntax commands on multiple lines. You can use either triple single quotes or triple double quotes, but you must use the same type (single or double) on both sides of the specified string.
- Notice that the triple-quoted expression is prefixed with the letter `r`. The `r` prefix to a string specifies Python's raw mode. In raw mode, Python treats all backslashes in the string as the backslash character and not as the start of an escape sequence.

SyntaxDocsList Class

The SyntaxDocsList class provides access to the list of open syntax documents. You obtain a SyntaxDocsList object from the GetSyntaxDocuments method of the SpssClient class.

A SyntaxDocsList object is not an iterable Python object. In order to iterate over the items in the list, use a for loop, as in:

```
for index in range(SyntaxDocsList.Size()):
```

For an example that uses the SyntaxDocsList class, see the examples for the SpssSyntaxDoc class.

GetItemAt Method

Returns an SpssSyntaxDoc object representing the syntax document with the specified index. The index corresponds to the order in which the syntax documents were opened, with the first opened document having an index of 0.

Syntax

```
SpssSyntaxDoc=SyntaxDocsList.GetItemAt(index)
```

Size Method

Returns the number of open syntax documents.

Syntax

```
SyntaxDocsList.Size()
```

SpssSyntaxUI Class

The SpssSyntaxUI class represents the Syntax Editor window associated with an open syntax document. You obtain an SpssSyntaxUI object from the GetSyntaxUI method of an SpssSyntaxDoc object.

Example: Get the SpssSyntaxUI Object Associated with the Designated Syntax Document

```
import SpssClient
SpssClient.StartClient()
DesignatedSyntaxDoc = SpssClient.GetDesignatedSyntaxDoc()
SyntaxUI = DesignatedSyntaxDoc.GetSyntaxUI()
```

- The variable *SyntaxUI* is an SpssSyntaxUI object for the Syntax Editor window associated with the designated syntax document.

GetHeight Method

Returns the height of the associated Syntax Editor window in units of pixels.

Syntax

```
SpssSyntaxUI.GetHeight()
```

GetLeft Method

Returns the horizontal screen position of the associated Syntax Editor window's upper left corner. The result is in units of pixels.

Syntax

```
SpssSyntaxUI.GetLeft()
```

GetTitleText Method

Returns the title bar text of the associated Syntax Editor window.

Syntax

```
SpssSyntaxUI.GetTitleText()
```

GetTop Method

Returns the vertical screen position of the associated Syntax Editor window's upper left corner. The result is in units of pixels.

Syntax

```
SpssSyntaxUI.GetTop()
```

GetVisible Method

Indicates if the associated Syntax Editor window is visible. The result is Boolean--*True* if the Syntax Editor window is visible, *False* otherwise.

Syntax

```
SpssSyntaxUI.GetVisible()
```

GetWidth Method

Returns the width of the associated Syntax Editor window in units of pixels.

Syntax

```
SpssSyntaxUI.GetWidth()
```

GetWindowState Method

Returns the state of the associated Syntax Editor window.

Syntax

```
SpssSyntaxUI.GetWindowState()
```

Table 21. Window states

Value	Description
SpssClient.SpssWindowStates.SpssMinimized	Minimized
SpssClient.SpssWindowStates.SpssMaximized	Maximized
SpssClient.SpssWindowStates.SpssNormal	Normal

InvokeDialog Method

Invokes a dialog and returns the syntax generated from that dialog, if any.

Syntax

```
syntax = SpssSyntaxUI.InvokeDialog(menuItemPath,desktopParent)
```

Parameters

menuItemPath. Menu or menu item with path of the dialog to invoke. See below for detailed description.

desktopParent. True specifies that the dialog is parented off the desktop. False specifies that the dialog is parented off an IBM SPSS Statistics window.

Note: For release 19.0.0.2 and higher, the *bSync* parameter (available in previous releases) is deprecated. The `InvokeDialog` method always runs synchronously, meaning that the scripting process waits until the dialog has been dismissed. Older scripts containing the *bSync* parameter will continue to function in release 19.0.0.2 and higher, but the value of the parameter will be ignored.

Specifying The Menu Item Path

The value of the *menuItemPath* parameter is a string specifying the menu path to the desired dialog--for example "analyze>survival>life tables". The greater-than sign (>) is used to separate a menu, its submenus and the menu item. The menu string must correspond exactly to the text on the menus, submenus, and menu items, and is language specific.

PrintSyntaxDoc Method

Prints the document.

Syntax

```
SpssSyntaxUI.PrintSyntaxDoc()
```

SetHeight Method

Sets the height of the associated Syntax Editor window.

Syntax

```
SpssSyntaxUI.SetHeight(height)
```

Parameters

height. An integer representing the height in pixels.

SetLeft Method

Sets the horizontal screen position of the associated Syntax Editor window's upper left corner.

Syntax

```
SpssSyntaxUI.SetLeft(leftPosition)
```

Parameters

leftPosition. An integer representing the position in pixels.

SetTop Method

Sets the vertical screen position of the associated Syntax Editor window's upper left corner.

Syntax

```
SpssSyntaxUI.SetTop(topPosition)
```

Parameters

topPosition. An integer representing the position in pixels.

SetVisible Method
Sets the visibility of the associated Syntax Editor window.

Syntax

```
SpssSyntaxUI.SetVisible(isVisible)
```

Parameters

isVisible. True to set the Syntax Editor window as visible, False otherwise.

SetWidth Method
Sets the width of the associated Syntax Editor window.

Syntax

```
SpssSyntaxUI.SetWidth(width)
```

Parameters

width. An integer representing the width in pixels.

SetWindowState Method
Set the state of the associated Syntax Editor window.

Syntax

```
SpssSyntaxUI.SetWindowState(newState)
```

Table 22. Window states

Value	Description
SpssClient.SpssWindowStates.SpssMinimized	Minimized
SpssClient.SpssWindowStates.SpssMaximized	Maximized
SpssClient.SpssWindowStates.SpssNormal	Normal

Output Items

SpssOutputItem Class
The `SpssOutputItem` class represents any item in an output document. You get an `SpssOutputItem` object from an `OutputItemList` object.

Example: Exporting the First Pivot Table to HTML

```
import SpssClient
SpssClient.StartClient()

OutputDoc = SpssClient.GetDesignatedOutputDoc()
OutputItemList = OutputDoc.GetOutputItems()

for index in range(OutputItemList.Size()):
   OutputItem = OutputItemList.GetItemAt(index)
   if OutputItem.GetType() == SpssClient.OutputItemType.PIVOT:
        OutputItem.ExportToDocument("/myfiles/mypivot",
                        SpssClient.DocExportFormat.SpssFormatHtml)
        break
SpssClient.StopClient()
```

- `SpssClient.GetDesignatedOutputDoc()` gets an `SpssOutputDoc` object for the designated output document. In this example, the variable *OutputDoc* is an `SpssOutputDoc` object.

- The `GetOutputItems` method of an `SpssOutputDoc` object returns an `OutputItemList` object. In this example, the variable *OutputItemList* is an `OutputItemList` object.
- The `for` loop iterates through all of the objects in the `OutputItemList` object--one object for each item in the output document. On each iteration of the loop, the variable *OutputItem* is an `SpssOutputItem` object.
- The `GetType` method from the `SpssOutputItem` class returns the type of the output item. Pivot tables have an output item type of `SpssClient.OutputItemType.PIVOT`.
- You export the pivot table using the `ExportToDocument` method from the `SpssOutputItem` class.
- The `break` statement terminates the loop if a pivot table is found.

ConvertToStdTable Method

This method is obsolete for release 20 and higher. *Note*: Lightweight tables created in IBM SPSS Statistics release 19 automatically have full support for pivoting and editing in release 20 or later.

ExportToDocument Method

Exports this output item in the specified document format.

- This method cannot be used for exporting chart items. To export a chart item, use the `ExportToImage` method.
- See SetOutputOptions for options available when exporting to Word, Excel, or PowerPoint.

Syntax

```
SpssOutputItem.ExportToDocument(fileName,format)
```

Table 23. Export formats

Format	Description
SpssClient.DocExportFormat.SpssFormatHtml	Html
SpssClient.DocExportFormat.SpssFormatDoc	Word
SpssClient.DocExportFormat.SpssFormatXls	Excel
SpssClient.DocExportFormat.SpssFormatText	Text
SpssClient.DocExportFormat.SpssFormatPdf	PDF
SpssClient.DocExportFormat.SpssFormatPpt	PowerPoint

ExportToImage Method

Exports this output item in the specified image format.

- This method can only be used for exporting charts, trees, and Model Viewer items. To export other item types, use the `ExportToDocument` method.
- When used for a Model Viewer item, the method exports the view displayed in the Viewer. You can export all views using the "ExportAllViews Method" on page 155.

Syntax

```
SpssOutputItem.ExportToImage(fileName,format)
```

Table 24. Image formats

Format	Description
SpssClient.ChartExportFormat.bmp	Windows bitmap
SpssClient.ChartExportFormat.emf	Enhanced metafile
SpssClient.ChartExportFormat.eps	Enhanced postscript
SpssClient.ChartExportFormat.jpg	JPG file

Table 24. Image formats (continued)

Format	Description
SpssClient.ChartExportFormat.png	PNG file
SpssClient.ChartExportFormat.tiff	Tagged image file

GetAlignment Method

Returns the alignment for this output item.

Syntax

```
SpssOutputItem.GetAlignment()
```

Returns

Table 25. Alignment types

Type	Description
SpssClient.OutputItemAlignment.Left	Left
SpssClient.OutputItemAlignment.Center	Center
SpssClient.OutputItemAlignment.Right	Right

When testing for a particular alignment type with the return value from `GetAlignment`, you can also use the following integer type codes: 0 (Left), 1 (Center), 2 (Right). For example:

```
if SpssOutputItem.GetAlignment() == 0:
```

GetDescription Method

Returns the name of this output item as it appears in the outline pane of the Viewer.

Syntax

```
SpssOutputItem.GetDescription()
```

GetHeight Method

Returns the height of this output item in units of points (1/72 inch).

- This method is not available for header items or root items.

Syntax

```
SpssOutputItem.GetHeight()
```

GetPageBreak Method

Indicates whether a page break is set before this item. The result is Boolean--*True* if the page break is set, *False* otherwise.

Syntax

```
SpssOutputItem.GetPageBreak()
```

GetParentItem Method

Returns an `SpssOutputItem` object representing the parent item of this output item.

Syntax

```
SpssOutputItem.GetParentItem()
```

GetProcedureName Method

Returns the name of the IBM SPSS Statistics procedure that generated this output item. The value is the OMS command identifier associated with the procedure.

Syntax

```
SpssOutputItem.GetProcedureName()
```

GetSpecificType Method

Returns an object of a specific output type, such as a pivot table or header item. You will need to call this method before using methods specific to that type of output item. For example, before you can use the methods available for a pivot table, you must call `GetSpecificType` on the associated `SpssOutputItem` object. The set of output types is listed in the description of the GetType method.

Syntax

```
object=SpssOutputItem.GetSpecificType()
```

For an example of using the `GetSpecificType` method, see "SpssPivotTable Class" on page 162.

GetSubType Method

Returns the OMS (Output Management System) sub type identifier, if any, of this output item.

Syntax

```
SpssOutputItem.GetSubType()
```

GetTreeLevel Method

Returns the level of this item within the hierarchy of the output tree. For instance, the root item is at level 0, and header items beneath the root are at level 1.

Syntax

```
SpssOutputItem.GetTreeLevel()
```

GetType Method

Returns the type associated with this output item.

Syntax

```
SpssOutputItem.GetType()
```

Table 26. Type codes

Type	Type Code
SpssClient.OutputItemType.UNKNOWN	0
SpssClient.OutputItemType.CHART	1
SpssClient.OutputItemType.HEAD	2
SpssClient.OutputItemType.LOG	3
SpssClient.OutputItemType.NOTE	4
SpssClient.OutputItemType.PIVOT	5
SpssClient.OutputItemType.ROOT	6

Table 26. Type codes (continued)

Type	Type Code
SpssClient.OutputItemType.TEXT	7
SpssClient.OutputItemType.WARNING	8
SpssClient.OutputItemType.TITLE	9
SpssClient.OutputItemType.PAGETITLE	11
SpssClient.OutputItemType.TREEMODEL	13
SpssClient.OutputItemType.GENERIC	14
SpssClient.OutputItemType.MODEL	15
SpssClient.OutputItemType.LIGHTNOTE	18
SpssClient.OutputItemType.LIGHTPIVOT	19
SpssClient.OutputItemType.LIGHTWARNING	20

When testing for a particular output type with the return value from `GetType`, you can use the integer type code or the textual specification of the type. For an example of using the `GetType` method, see "SpssPivotTable Class" on page 162.

Notes

- Standard charts, graphboard charts, and R graphics all have the type `SpssClient.OutputItemType.CHART`. To distinguish between these chart types, use the SPSSSubtype method on the associated `SpssChartItem` object.
- Objects of type *SpssClient.OutputItemType.ROOT* (the root object in an output document) or *SpssClient.OutputItemType.HEAD* are `SpssHeaderItem` objects.
- Objects of type *SpssClient.OutputItemType.TREEMODEL* are `SpssChartItem` objects.
- The object types *SpssClient.OutputItemType.LIGHTNOTE*, *SpssClient.OutputItemType.LIGHTPIVOT*, and *SpssClient.OutputItemType.LIGHTWARNING* are obsolete for release 20 and higher. Lightweight Notes items, lightweight Pivot Table items, and lightweight Warnings items created in release 19 will have the output types *SpssClient.OutputItemType.NOTE*, *SpssClient.OutputItemType.PIVOT* and *SpssClient.OutputItemType.WARNING* respectively when accessed in release 20 or higher.

GetTypeString Method

Returns the type string of this output item. The returned value is not translated.

Syntax

```
SpssOutputItem.GetTypeString()
```

Returns

Returns one of the following strings: "Chart", "Log", "Notes", "Table", "Text", "Warning", "Title", "PageTitle", "TreeDiagram", "Model".

Notes

- Standard charts, graphboard charts, and R graphics all have the type string 'Chart'. To distinguish between these chart types, use the SPSSSubtype method on the associated `SpssChartItem` object.
- Lightweight Notes items, lightweight Pivot Table items, and lightweight Warnings items created in release 19 will have the type strings *Notes*, *Table*, and *Warning* respectively when accessed in release 20 or higher.

GetWidth Method

Returns the width of this output item in units of points (1/72 inch).

- This method is not available for header items or root items.

Syntax

```
SpssOutputItem.GetWidth()
```

GetXML Method

Returns the XML representation for an `SpssChartItem` or `SpssModelItem`, as a Unicode string.

Syntax

```
SpssOutputItem.GetXML()
```

You can set the XML for a chart item using the SetXML method from the `SpssChartItem` class. You can set the XML for a Model Viewer item using the SetXML method from the `SpssModelItem` class.

IsCurrentItem Method

Indicates if this output item is the current item--as indicated by a red arrow next to the item in the outline pane. The result is Boolean--*True* if the item is the current item, *False* otherwise.

Syntax

```
SpssOutputItem.IsCurrentItem()
```

IsEditable Method

Indicates whether this output item can be edited. The result is Boolean--*True* if the item can be edited, *False* otherwise.

Syntax

```
SpssOutputItem.IsEditable()
```

IsEqualTo Method

Indicates if this output item object is the same object as a specified output item object. The result is Boolean--*True* if the two objects are identical, *False* otherwise.

Syntax

```
SpssOutputItem.IsEqualTo(outputItem)
```

Parameters

outputItem. An `SpssOutputItem` object

IsSelected Method

Indicates whether the current output item is selected. The result is Boolean--*True* if the item is currently selected, *False* otherwise. Use the SetSelected method to select an item.

Syntax

```
SpssOutputItem.IsSelected()
```

IsVisible Method

Indicates if this output item is visible. The result is Boolean--*True* if the item is visible, *False* if it is hidden.

Syntax

```
SpssOutputItem.IsVisible()
```

SetAlignment Method

Sets the alignment for this output item.

Syntax

```
SpssOutputItem.SetAlignment(alignment)
```

Table 27. Alignment types

Type	Description
SpssClient.OutputItemAlignment.Left	Left
SpssClient.OutputItemAlignment.Center	Center
SpssClient.OutputItemAlignment.Right	Right

SetCurrentItem Method

Sets the item as the current item--as indicated by a red arrow next to the item in the outline pane.

Syntax

```
SpssOutputItem.SetCurrentItem()
```

SetDescription Method

Sets the name of this output item. This is the name that is displayed in the outline pane of the Viewer. The value can be specified as plain text, HTML, or rich text format. For HTML, embed markup in a <html></html> block. For rich text format, specify the string as a raw string to avoid unintentional escape sequences.

Syntax

```
SpssOutputItem.SetDescription(desc)
```

SetHeight Method

Sets the height of this output item in units of points (1/72 inch).

- This method is not available for pivot tables, header items, or the root item.

Syntax

```
SpssOutputItem.SetHeight(height)
```

SetPageBreak Method

Sets or clears a page break before this item.

Syntax

```
SpssOutputItem.SetPageBreak(pageBreak)
```

Parameters

pageBreak. True to set a page break, False to clear a page break.

SetProcedureName Method

Sets the procedure name associated with this output item. The argument is a string and is not translated.

Syntax

```
SpssOutputItem.SetProcedureName(procName)
```

SetSelected Method

Specifies whether the current output item is set as selected. You can use the IsSelected method to determine if a given item is already selected.

Syntax

```
SpssOutputItem.SetSelected(selected)
```

Parameters

selected. True to set this item as selected, False to set it as not selected.

SetSubType Method

Sets the OMS (Output Management System) sub-type identifier of this output item.

Syntax

```
SpssOutputItem.SetSubType(subType)
```

Parameters

subType. A string

SetTreeLevel Method

Sets the level of this item within the hierarchy of the output tree. For instance, the root item is at level 0, and header items beneath the root are at level 1.

Syntax

```
SpssOutputItem.SetTreeLevel(level)
```

Parameters

level. An integer

SetVisible Method

Specifies whether this output item is visible.

Syntax

```
SpssOutputItem.SetVisible(visible)
```

Parameters

visible. True to set the item as visible, False to set it as hidden.

SetWidth Method

Sets the width of this output item in units of points (1/72 inch).

- This method is not available for pivot tables, header items, or the root item.

Syntax

```
SpssOutputItem.SetWidth(width)
```

SpssChartItem Class

The SpssChartItem class represents a chart item in an output document. You get an SpssChartItem object from the collection of output items in an output document.

Example: Getting Chart Items

```
import SpssClient
SpssClient.StartClient()

OutputDoc = SpssClient.GetDesignatedOutputDoc()
OutputItems = OutputDoc.GetOutputItems()

for index in range(OutputItems.Size()):
   OutputItem = OutputItems.GetItemAt(index)
   if OutputItem.GetType() == SpssClient.OutputItemType.CHART:
        ChartItem = OutputItem.GetSpecificType()
```

- Chart items have an output item type of SpssClient.OutputItemType.CHART.
- Once an output item has been identified as a chart item, you get an SpssChartItem object by calling the GetSpecificType method on the output item object. In this example, *ChartItem* is an SpssChartItem object.

SetXML Method

Sets the chart XML from a UTF-8 (Unicode Transformation Format, 8 bit) string.

Syntax

```
SpssChartItem.SetXML(xml)
```

You can get the XML for a chart item using the GetXML method from the SpssOutputItem class. You can also use chart XML (OXML) created by the OMS command as the source for SetXML. To do so, extract the visualization element from the OXML, decode the resulting string to "UTF-8" (e.g., with the Python decode string method), and use the decoded string as the argument to SetXML.

SPSSSubtype Method

Returns a string specifying the type of chart.

Syntax

```
SpssChartItem.SPSSSubtype()
```

Table 28. Chart types

Type	Description
CHART	Standard chart
GRAPHBOARD	Graphboard chart
IMAGE	R graphic
TREEMODEL	Tree model

SpssModelItem Class

The SpssModelItem class represents a Model Viewer item in an output document. You get an SpssModelItem object from the collection of output items in an output document.

Example: Getting Model Viewer Items

```
import SpssClient
SpssClient.StartClient()

OutputDoc = SpssClient.GetDesignatedOutputDoc()
OutputItems = OutputDoc.GetOutputItems()

for index in range(OutputItems.Size()):
   OutputItem = OutputItems.GetItemAt(index)
   if OutputItem.GetType() == SpssClient.OutputItemType.MODEL:
        ModelItem = OutputItem.GetSpecificType()
```

- Model Viewer items have an output item type of SpssClient.OutputItemType.MODEL.
- Once an output item has been identified as a Model Viewer item, you get an SpssModelItem object by calling the GetSpecificType method on the output item object. In this example, *ModelItem* is an SpssModelItem object.

ExportAllViews Method

Exports all views of this Model Viewer item in the specified image format.

Syntax

```
SpssModelItem.ExportAllViews(filePrefix,format)
```

Parameters

The argument *filePrefix* is the full path and file name prefix for the files containing the exported views. Each view is exported to a separate file.

On Windows, it is recommended to use raw strings for file paths, or replace backslashes with forward slashes (IBM SPSS Statistics accepts a forward slash for any backslash in a file specification). Raw strings are specified by prefacing the string with r, as in r'c:\examples\mydata.sav'. In raw mode, Python treats all backslashes in the string as the backslash character and not as the start of an escape sequence.

Table 29. Image formats

Format	Description
SpssClient.ChartExportFormat.bmp	Windows bitmap
SpssClient.ChartExportFormat.emf	Enhanced metafile
SpssClient.ChartExportFormat.eps	Enhanced postscript
SpssClient.ChartExportFormat.jpg	JPG file
SpssClient.ChartExportFormat.png	PNG file
SpssClient.ChartExportFormat.tiff	Tagged image file

You can export the view displayed in the Viewer using the “ExportToImage Method” on page 147.

SetXML Method

Sets the XML for the Model Viewer item from a UTF-8 (Unicode Transformation Format, 8 bit) string.

Syntax

```
SpssModelItem.SetXML(xml)
```

You can get the XML for a Model Viewer item using the GetXML method from the SpssOutputItem class. You can also use model XML (OXML) created by the OMS command as the source for SetXML.

SpssHeaderItem Class

The SpssHeaderItem class represents a header item in an output document. You get an SpssHeaderItem object from the collection of output items in an output document.

Example: Getting Header Items

```
import SpssClient
SpssClient.StartClient()

OutputDoc = SpssClient.GetDesignatedOutputDoc()
OutputItems = OutputDoc.GetOutputItems()

for index in range(OutputItems.Size()):
   OutputItem = OutputItems.GetItemAt(index)
   if OutputItem.GetType() == SpssClient.OutputItemType.HEAD:
        HeaderItem = OutputItem.GetSpecificType()
```

- Header items have an output item type of SpssClient.OutputItemType.HEAD.
- Once an output item has been identified as a header item, you get an SpssHeaderItem object by calling the GetSpecificType method on the output item object. In this example, *HeaderItem* is an SpssHeaderItem object.

Note: The root item is an SpssHeaderItem object.

GetChildCount Method

Returns the child item count for this header item.

Syntax

```
SpssHeaderItem.GetChildCount()
```

GetChildItem Method

Returns an SpssOutputItem object for the child item at the specified index. Index values start from 0.

Syntax

```
SpssOutputItem=SpssHeaderItem.GetChildItem(index)
```

InsertChildItem Method

Inserts a child item--at the specified index--under the current header item in the associated output document. Index values start from 0 and are relative to the current header item.

- Use this method to insert header items, text items, and title items created with the CreateHeaderItem, CreateTextItem, and CreateTitleItem methods from the SpssOutputDoc class.

Syntax

```
SpssHeaderItem.InsertChildItem(item,index)
```

Parameters

item. An SpssOutputItem object

index. The index position of the new child item in the header item's child list. Index values start from 0. To append an item to the end of the child list, use an index value equal to the current child count.

Example: Appending a new header item containing a child text item

This example appends a new header item under the root item. A text item is added under the new header item.

```
import SpssClient
SpssClient.StartClient()
doc = SpssClient.GetDesignatedOutputDoc()
itemlist = doc.GetOutputItems()
# Get the root header item
root = itemlist.GetItemAt(0).GetSpecificType()
# Create a new header item
newHeader = doc.CreateHeaderItem("New header")
# Append the new header to the root item
root.InsertChildItem(newHeader,root.GetChildCount())
# Get the new header item
newHeaderItem = root.GetChildItem(root.GetChildCount()-1).GetSpecificType()
# Create a new text item
newText = doc.CreateTextItem("New text")
# Append the new text item to the new header item
newHeaderItem.InsertChildItem(newText,0)
SpssClient.StopClient()
```

Example: Inserting a text item under an existing header item

This example inserts a text item at index position 1, under a header item identified by the description string "Demo".

```
import SpssClient
SpssClient.StartClient()
doc = SpssClient.GetDesignatedOutputDoc()
OutputItems = doc.GetOutputItems()
for index in range(OutputItems.Size()):
   OutputItem = OutputItems.GetItemAt(index)
   if OutputItem.GetType() == SpssClient.OutputItemType.HEAD \
         and OutputItem.GetDescription() == "Demo":
      HeaderItem = OutputItem.GetSpecificType()
      newText = doc.CreateTextItem("My inserted text")
      HeaderItem.InsertChildItem(newText,1)
SpssClient.StopClient()
```

IsExpanded Method

Indicates whether the associated header item is expanded. The result is Boolean--*True* if the header item is expanded, *False* otherwise.

Syntax

```
SpssHeaderItem.IsExpanded()
```

RemoveChildItem Method

Removes the child item at the specified index. Index values start from 0.

Syntax

```
SpssHeaderItem.RemoveChildItem(index)
```

SetExpanded Method

Sets whether the associated header item is expanded.

Syntax

```
SpssHeaderItem.SetExpanded(expand)
```

Parameters

expand. True to expand the item, False to collapse the item.

SpssLogItem Class

The SpssLogItem class represents a log item in an output document. You get an SpssLogItem object from the collection of output items in an output document.

Example: Getting Log Items

```
import SpssClient
SpssClient.StartClient()

OutputDoc = SpssClient.GetDesignatedOutputDoc()
OutputItems = OutputDoc.GetOutputItems()

for index in range(OutputItems.Size()):
   OutputItem = OutputItems.GetItemAt(index)
   if OutputItem.GetType() == SpssClient.OutputItemType.LOG:
        LogItem = OutputItem.GetSpecificType()
```

- Log items have an output item type of SpssClient.OutputItemType.LOG.
- Once an output item has been identified as a log item, you get an SpssLogItem object by calling the GetSpecificType method on the output item object. In this example, *LogItem* is an SpssLogItem object.

Append Method

Appends the specified text to the contents of the associated log output item.

Syntax

```
SpssLogItem.Append(text)
```

GetTextContents Method

Returns the contents of the associated log output item. The value is returned as plain text.

Syntax

```
SpssLogItem.GetTextContents()
```

SetTextContents Method

Sets the contents of the associated log output item, replacing any existing content. The value can be specified as plain text, HTML, or rich text format. For HTML, embed markup in a <html></html> block. For rich text format, specify the string as a raw string to avoid unintentional escape sequences. For multiple lines, use "\n" to specify line breaks.

Syntax

```
SpssLogItem.SetTextContents(contents)
```

SpssTextItem Class

The SpssTextItem class represents a text item in an output document. You get an SpssTextItem object from the collection of output items in an output document.

Example: Getting Text Items

```
import SpssClient
SpssClient.StartClient()

OutputDoc = SpssClient.GetDesignatedOutputDoc()
OutputItems = OutputDoc.GetOutputItems()

for index in range(OutputItems.Size()):
   OutputItem = OutputItems.GetItemAt(index)
   if OutputItem.GetType() == SpssClient.OutputItemType.TEXT:
        TextItem = OutputItem.GetSpecificType()
```

- Text items have an output item type of SpssClient.OutputItemType.TEXT.

- Once an output item has been identified as a text item, you get an SpssTextItem object by calling the GetSpecificType method on the output item object. In this example, *TextItem* is an SpssTextItem object.

GetTextContents Method

Returns the contents of the associated text output item. The value is returned as plain text.

Syntax

```
SpssTextItem.GetTextContents()
```

SetTextContents Method

Sets the contents of the associated text output item, replacing any existing content. The value can be specified as plain text, HTML, or rich text format. For HTML, embed markup in a <html></html> block. For rich text format, specify the string as a raw string to avoid unintentional escape sequences. For multiple lines, use "\n" to specify line breaks.

Syntax

```
SpssTextItem.SetTextContents(contents)
```

SpssTitleItem Class

The SpssTitleItem class represents a title item in an output document. You get an SpssTitleItem object from the collection of output items in an output document.

Example: Getting Title Items

```
import SpssClient
SpssClient.StartClient()

OutputDoc = SpssClient.GetDesignatedOutputDoc()
OutputItems = OutputDoc.GetOutputItems()

for index in range(OutputItems.Size()):
   OutputItem = OutputItems.GetItemAt(index)
   if OutputItem.GetType() == SpssClient.OutputItemType.TITLE:
        TitleItem = OutputItem.GetSpecificType()
```

- Title items have an output item type of SpssClient.OutputItemType.TITLE.
- Once an output item has been identified as a title item, you get an SpssTitleItem object by calling the GetSpecificType method on the output item object. In this example, *TitleItem* is an SpssTitleItem object.

GetTextContents Method

Returns the contents of the associated title output item. The value is returned as plain text.

Syntax

```
SpssTitleItem.GetTextContents()
```

SetTextContents Method

Sets the contents of the associated title output item, replacing any existing content. The value can be specified as plain text, HTML, or rich text format. For HTML, embed markup in a <html></html> block. For rich text format, specify the string as a raw string to avoid unintentional escape sequences.

Syntax

```
SpssTitleItem.SetTextContents(contents)
```

Menus

MenuTableList Class

The MenuTableList class provides access to the list of available menu items for a data, output, or syntax document. You obtain a MenuTableList object from the GetMenuTable method of an SpssDataDoc, SpssOutputDoc, or SpssSyntaxDoc object.

A MenuTableList object is not an iterable Python object. In order to iterate over the items in the list, use a for loop, as in:

```
for index in range(MenuTableList.Size()):
```

For an example that uses the MenuTableList class, see the example for the SpssMenuItem class.

GetItemAt Method

Returns an SpssMenuItem object representing the menu item with the specified index.

Syntax

```
SpssMenuItem=MenuTableList.GetItemAt(index)
```

Size Method

Returns the number of items in a MenuTableList object.

Syntax

```
MenuTableList.Size()
```

SpssMenuItem Class

The SpssMenuItem class represents a menu item in a data, output, or syntax document. You get an SpssMenuItem object from the GetItemAt method of a MenuTableList object.

Example: Getting a List of Menu Items

```
import SpssClient
SpssClient.StartClient()

OutputDoc = SpssClient.GetDesignatedOutputDoc()
MenuTableList = OutputDoc.GetMenuTable()
strMenuItemList = []
for i in range(MenuTableList.Size()):
    item = MenuTableList.GetItemAt(i)
    strMenuItemList.append(item.GetTextContents())
```

GetTextContents Method

Returns the name of the menu item (as a string) associated with an SpssMenuItem object.

Syntax

```
SpssMenuItem.GetTextContents()
```

Pivot Tables

Pivot Tables

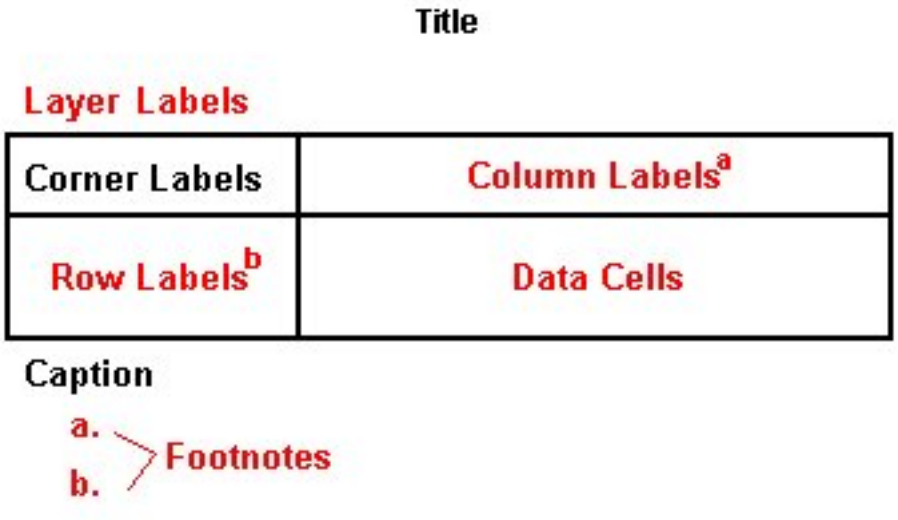
The scripting facility allows you to do most of the things you can do in the pivot table editor, through use of the SpssPivotTable class. There are two general approaches for working with pivot tables from scripting:

- Select groups of cells (results or labels) or other elements (such as footnotes) and apply methods that modify the entire selection. For example, you can change the foreground color for selected cells.
- Access a subset of the pivot table, such as its data cells or row labels, and modify a particular element in the subset. For example, you can access the data cells and call a method to set the foreground color for a specified cell.

Generally speaking, if you want to modify a number of elements in the same manner, the first approach is faster.

Areas of a Pivot Table

(Red labels indicate accessible objects.)



Available Objects

The `SpssPivotTable` object provides access to the following objects:

- SpssDataCells Provides access to the data cells.
- SpssLabels Provides access to the row and column labels.
- SpssFootnotes Provides access to all of the table's footnotes.
- SpssLayerLabels Provides access to labels in any layer dimensions.
- SpssPivotMgr Provides access to row, column, and layer dimensions.
- SpssDimension Provides access to the properties of a particular dimension.

Compatibility with previous releases

Legacy tables (referred to as full-featured tables in release 19) are tables that are fully compatible with IBM SPSS Statistics releases prior to 20. Legacy tables may render slowly and are only recommended if compatibility with releases prior to 20 is required. You can specify that tables are rendered as legacy tables by calling the `SpssClient.SetPreference` method with the `TableRender` option set to `"full"`. See the topic "SetPreference Method" on page 109 for more information. You can also specify legacy table creation with the command syntax `SET TABLERENDER = FULL`.

- Tables, other than legacy tables, created in IBM SPSS Statistics release 20 or later and lightweight tables in output documents that are modified in release 20 or later (but created in release 19) cannot be viewed or accessed through scripting in releases prior to 19.0.0.2. Such tables are viewable and accessible through scripting in release 19.0.0.2, where they are rendered as lightweight tables; however, they may not render the same as in release 20 or later.

- Lightweight tables created in IBM SPSS Statistics release 19 automatically have full support for pivoting and editing in release 20 or later.

SpssPivotTable Class

The SpssPivotTable class allows you to operate on the table as a whole as well as providing access to objects for working with the footnotes, data cells, row or column labels, and layer labels associated with the table. Pivot tables are output items and are accessed from the list of output items associated with an output document.

Example: Getting Pivot Tables

```
import SpssClient
SpssClient.StartClient()

OutputDoc = SpssClient.GetDesignatedOutputDoc()
OutputItems = OutputDoc.GetOutputItems()

for index in range(OutputItems.Size()):
   OutputItem = OutputItems.GetItemAt(index)
   if OutputItem.GetType() == SpssClient.OutputItemType.PIVOT:
        PivotTable = OutputItem.GetSpecificType()
```

- Pivot tables have an output item type of SpssClient.OutputItemType.PIVOT.
- Once an output item has been identified as a pivot table, you get an SpssPivotTable object by calling the GetSpecificType method on the output item object. In this example, *PivotTable* is an SpssPivotTable object.

Example: Getting the First Selected Pivot Table

```
import SpssClient
SpssClient.StartClient()

OutputDoc = SpssClient.GetDesignatedOutputDoc()
OutputItems = OutputDoc.GetOutputItems()

for index in range(OutputItems.Size()):
   OutputItem = OutputItems.GetItemAt(index)
   if OutputItem.GetType() == SpssClient.OutputItemType.PIVOT \
      and OutputItem.IsSelected():
        PivotTable = OutputItem.GetSpecificType()
```

Example: Getting the First Pivot Table Labeled "Statistics"

```
import SpssClient
SpssClient.StartClient()

OutputDoc = SpssClient.GetDesignatedOutputDoc()
OutputItems = OutputDoc.GetOutputItems()

for index in range(OutputItems.Size()):
   OutputItem = OutputItems.GetItemAt(index)
   if OutputItem.GetType() == SpssClient.OutputItemType.PIVOT \
      and OutputItem.GetDescription() == "Statistics":
        PivotTable = OutputItem.GetSpecificType()
```

Autofit Method

Recalculates the size of all cells in the entire table to accommodate label lengths or the lengths of both labels and data values.

- To specify how the cells are to be recalculated (labels only or labels and data), use the SetPreference method from the SpssClient class and specify SpssClient.PreferenceOptions.ColumnWidth as the option.

Syntax

```
SpssPivotTable.Autofit()
```

ClearSelection Method

Deselects all selected output items or pivot table elements.

- All Select methods add the current item(s)/element(s) to what has been previously selected. Always clear selections before you start selecting output items or table elements.

Syntax

```
SpssPivotTable.ClearSelection()
```

ColumnLabelArray Method

Returns an SpssLabels object representing the column labels.

Syntax

```
SpssLabels=SpssPivotTable.ColumnLabelArray()
```

DataCellArray Method

Returns an SpssDataCells object representing the data cells of the pivot table.

Syntax

```
SpssDataCells=SpssPivotTable.DataCellArray()
```

DisplayTableByRows Method

Specifies whether to display the table *n* rows at a time, where *n* is set with the SetRowsToDisplayRowCount method. The argument is Boolean--*True* to display the table *n* rows at a time, *False* otherwise.

Note: This method is only available for legacy tables. See the topic "Compatibility with previous releases" on page 161 for more information.

Syntax

```
SpssPivotTable.DisplayTableByRows(boolean)
```

FootnotesArray Method

Returns an SpssFootnotes object representing the table footnotes.

Syntax

```
SpssFootnotes=SpssPivotTable.FootnotesArray()
```

GetCaptionText Method

Returns the caption text for the current table.

Syntax

```
SpssPivotTable.GetCaptionText()
```

GetFootnoteMarkersPosition Method

Gets the current position--superscript or subscript--for footnote markers for the pivot table.

Syntax

```
SpssPivotTable.GetFootnoteMarkersPosition()
```

Table 30. Returned values

Value	Description
SpssClient.SpssFootnoteMarkerTypes.SpssFtSuperscript	Superscript
SpssClient.SpssFootnoteMarkerTypes.SpssFtSubscript	Subscript

GetFootnoteMarkersStyle Method

Gets the current style--alphabetic or numeric--for footnote markers for the pivot table.

Syntax

```
SpssPivotTable.GetFootnoteMarkersStyle()
```

Table 31. Returned values

Value	Description
SpssClient.SpssFootnoteMarkerTypes.SpssFtAlphabetic	Alphabetic
SpssClient.SpssFootnoteMarkerTypes.SpssFtNumeric	Numeric

GetHeight Method

Returns the height of the pivot table. The unit is the point (1/72 inch).

Syntax

```
SpssPivotTable.GetHeight()
```

GetRotateColumnLabels Method

Indicates if category labels closest to the data (that is, categories of the column dimension with the largest index) are rotated. The result is Boolean.

Syntax

```
SpssPivotTable.GetRotateColumnLabels()
```

Returns

True. The column labels closest to the data are displayed vertically

False. The column labels closest to the data are displayed horizontally

GetRotateRowLabels Method

Indicates if the labels of all but the last row dimension (that is, the row dimension with the largest index) are rotated. The result is Boolean.

Syntax

```
SpssPivotTable.GetRotateRowLabels()
```

Returns

True. The outer row labels are displayed vertically

False. The outer row labels are displayed horizontally

GetSigMarkersType Method

Gets the type of the significance indicators that are used for the pivot table. The method returns None when the table has no significance indicators. Significance indicators are created for custom tables (`CTABLES`) when pairwise comparisons of column proportions or column means are calculated. The `GetSigMarkersType` method returns the type only when the indicators are merged into the main table.

Syntax

```
SpssPivotTable.GetSigMarkersType()
```

Table 32. Returned values

Value	Description
SpssClient.SpssSigMarkerTypes.SpssSigSimple	Simple indicators
SpssClient.SpssSigMarkerTypes.SpssSigAPA	APA-style indicators
None	The table has no significance indicators

GetTitleText Method

Returns the text of the title for the table.

Syntax

```
SpssPivotTable.GetTitleText()
```

GetUpdateScreen Method

Returns whether changes in the pivot table are refreshed immediately. The result is Boolean--*True* if changes are refreshed immediately, *False* otherwise.

- By default, changes are refreshed immediately. Use the `SetUpdateScreen` method to specify that changes are not to be refreshed immediately.

Syntax

```
SpssPivotTable.GetUpdateScreen()
```

GetVarNamesDisplay Method

This method returns the setting for how variable names are displayed in the pivot table: as variable names, as variable labels, or both.

Note: This method is not supported for legacy tables.

Syntax

```
SpssPivotTable.GetVarNamesDisplay()
```

Table 33. Returned values.

Value	Description
SpssClient.VarNamesDisplay.Names	Names
SpssClient.VarNamesDisplay.Labels	Labels
SpssClient.VarNamesDisplay.Both	Names and labels

GetVarValuesDisplay Method

This method gets the setting for how variable values are displayed in the pivot table: as values, as value labels, or both.

Note: This method is not supported for legacy tables.

Syntax

```
SpssPivotTable.GetVarValuesDisplay()
```

Table 34. Returned values.

Value	Description
SpssClient.VarValuesDisplay.Values	Values
SpssClient.VarValuesDisplay.Labels	Labels
SpssClient.VarValuesDisplay.Both	Values and labels

GetWidowOrphanLines Method

Returns the number of allowable widow/orphan lines when pivot tables are printed.

- Widow lines are the last few lines of a paragraph printed at the top of the next page; orphan lines are the first few lines of a paragraph printed at the bottom of the previous page.

Syntax

```
SpssPivotTable.GetWidowOrphanLines()
```

GetWidth Method

Returns the width of the pivot table. The unit is the point (1/72 inch).

Syntax

```
SpssPivotTable.GetWidth()
```

Group Method

Groups selected category labels or group labels, creates a grouping level, and inserts a grouping label.

- The selection must be category or group labels.
- After the execution of this method, the inserted grouping label is selected and has the default label of *Group Label*.
- If a new group level is inserted, labels on the same and lower levels are demoted one level. (For column labels, the row index increases by one; for row labels, the column index increases by one.)

Syntax

```
SpssPivotTable.Group()
```

Example

This example assumes that *PivotTable* is an `SpssPivotTable` object, selects category column labels Clerical and Custodial, and groups them under the label of Non-Managerial.

```
ColumnLabels = PivotTable.ColumnLabelArray()

#Select the category column labels Clerical and Custodial:
PivotTable.ClearSelection()
for i in range(ColumnLabels.GetNumRows()):
   for j in range(ColumnLabels.GetNumColumns()):
     if ColumnLabels.GetValueAt(i,j) in ["Clerical","Custodial"]:
```

```
      ColumnLabels.SelectLabelAt(i,j)

#Group the categories and assign a group label:
PivotTable.Group()
for i in range(ColumnLabels.GetNumRows()):
   for j in range(ColumnLabels.GetNumColumns()):
     if ColumnLabels.GetValueAt(i,j)=="Group Label":
      ColumnLabels.SetValueAt(i,j,"Non-Managerial")
```

HideCaption Method

Hides the caption of the current table.

Syntax

```
SpssPivotTable.HideCaption()
```

HideFootnote Method

Hides the selected footnotes or all the footnotes referenced by the selected cell.

Syntax

```
SpssPivotTable.HideFootnote()
```

HideTitle Method

Hides the title of a pivot table.

Syntax

```
SpssPivotTable.HideTitle()
```

InsertFootnote Method

Inserts a footnote to the selected data or label cell.

- If multiple data cells or labels are selected, the footnote is attached to the first selected item.
- To set a footnote for corner text, first set the corner text with the `SpssPivotTable.SetCornerText` method.

Syntax

```
SpssPivotTable.InsertFootnote(string)
```

Parameters

string. Text of the footnote

IsDisplayTableByRows Method

Indicates whether the table is being displayed a fixed number of rows at a time. The result is Boolean--*True* if the table is being displayed a fixed number of rows at a time, *False* otherwise. Use the DisplayTableByRows method to change the setting.

Note: The feature to display a table a fixed number of rows at a time is only available for legacy tables. See the topic “Compatibility with previous releases” on page 161 for more information.

Syntax

```
SpssPivotTable.IsDisplayTableByRows()
```

IsLegacyTableCompatible Method

Indicates whether row and column labels are indexed in the same manner as for legacy tables. The result is Boolean--*True* if row and column labels are indexed in the same manner as for legacy tables, *False* otherwise.

- By default, row and column labels are indexed in the same manner as for legacy tables.
- `IsLegacyTableCompatible` always returns *True* for legacy tables.

Syntax

```
SpssPivotTable.IsLegacyTableCompatible()
```

LayerLabelArray Method

Returns an `SpssLayerLabels` object representing all layer labels.

Syntax

```
SpssLayerLabels=SpssPivotTable.LayerLabelArray()
```

NavigateToFirstRow Method

Displays the first block of rows of the table when displaying the table a fixed number of rows at a time. Use the DisplayTableByRows method to display the table a fixed number of rows at a time.

Note: This method is only available for legacy tables. See the topic "Compatibility with previous releases" on page 161 for more information.

Syntax

```
SpssPivotTable.NavigateToFirstRow()
```

NavigateToLastRow Method

Displays the last block of rows of the table when displaying the table a fixed number of rows at a time. Use the DisplayTableByRows method to display the table a fixed number of rows at a time.

Note: This method is only available for legacy tables. See the topic "Compatibility with previous releases" on page 161 for more information.

Syntax

```
SpssPivotTable.NavigateToLastRow()
```

NavigateToNextRows Method

Displays the next block of rows of the table when displaying the table a fixed number of rows at a time. Use the DisplayTableByRows method to display the table a fixed number of rows at a time.

Note: This method is only available for legacy tables. See the topic "Compatibility with previous releases" on page 161 for more information.

Syntax

```
SpssPivotTable.NavigateToNextRows()
```

NavigateToPreviousRows Method

Displays the previous block of rows of the table when displaying the table a fixed number of rows at a time. Use the DisplayTableByRows method to display the table a fixed number of rows at a time.

Note: This method is only available for legacy tables. See the topic "Compatibility with previous releases" on page 161 for more information.

Syntax

```
SpssPivotTable.NavigateToPreviousRows()
```

NumericFormat Method

Sets the display format for numeric values in the selected cells of the current table.

```
SpssPivotTable.NumericFormat(format,decimal)
```

Parameters

format. The string description of the format.

decimal. Number of decimal places.

For a listing of the format types, see Appendix D, "String Description of Numeric Formats," on page 253.

PivotManager Method

Returns an `SpssPivotMgr` object, providing access to the Pivot Manager.

Syntax

```
SpssPivotMgr=SpssPivotTable.PivotManager()
```

RowLabelArray Method

Returns an `SpssLabels` object representing the row labels.

Syntax

```
SpssLabels=SpssPivotTable.RowLabelArray()
```

SelectAllFootnotes Method

Selects all footnotes in the pivot table, in addition to what has already been selected.

Syntax

```
SpssPivotTable.SelectAllFootnotes()
```

SelectCaption Method

Selects the caption of the pivot table, in addition to all previously selected elements.

Syntax

```
SpssPivotTable.SelectCaption()
```

SelectCorner Method

Selects the corner of the pivot table, in addition to all previously selected elements.

Syntax

```
SpssPivotTable.SelectCorner()
```

SelectTable Method

Selects all the elements of a pivot table for modification.

Syntax

```
SpssPivotTable.SelectTable()
```

SelectTableBody Method

Selects the body of the pivot table (labels and data cells) for modification.

Syntax

```
SpssPivotTable.SelectTableBody()
```

SelectTitle Method

Selects the title of the pivot table for modification.

Syntax

```
SpssPivotTable.SelectTitle()
```

SetBackgroundColor Method

Sets the background color of the selected element(s) in the current pivot table.

Syntax

```
SpssPivotTable.SetBackgroundColor(color)
```

Parameters

color. Integer representation of the color

For information on setting color values, see Appendix B, "Setting Color Values," on page 249.

SetBottomMargin Method

Sets the bottom margin of the selected cells in the current pivot table.

Syntax

```
SpssPivotTable.SetBottomMargin(margin)
```

Parameters

margin. An integer. The unit is the point (1/72 inch). The maximum value is 36.

SetCaptionText Method

Sets the caption text for the current table.

Syntax

```
SpssPivotTable.SetCaptionText(string)
```

Parameters

string. Caption text

SetCornerText Method

Sets the corner text.

Syntax

```
SpssPivotTable.SetCornerText(string)
```

Parameters

string. Corner text

SetDataCellWidths Method

Sets the width of all data cells of the current table.

Syntax

```
SpssPivotTable.SetDataCellWidths(width)
```

Parameters

width. An integer. The unit is the point (1/72 inch).

SetFootnoteMarkers Method

Sets the style of footnote markers for the entire table.

Syntax

```
SpssPivotTable.SetFootnoteMarkers(type)
```

Table 35. Footnote marker types

Type	Description
SpssClient.SpssFootnoteMarkerTypes.SpssFtSuperscript	Superscript
SpssClient.SpssFootnoteMarkerTypes.SpssFtSubscript	Subscript
SpssClient.SpssFootnoteMarkerTypes.SpssFtAlphabetic	Alphabetic
SpssClient.SpssFootnoteMarkerTypes.SpssFtNumeric	Numeric

SetForegroundColor Method

This method is deprecated in release 17.0. Use the SetTextColor method instead.

SetHAlign Method

Sets the horizontal alignment of the selected elements in the current table.

Syntax

```
SpssPivotTable.SetHAlign(alignment)
```

Table 36. Horizontal alignment types

Type	Description
SpssClient.SpssHAlignTypes.SpssHAlLeft	Left
SpssClient.SpssHAlignTypes.SpssHAlRight	Right
SpssClient.SpssHAlignTypes.SpssHAlCenter	Center
SpssClient.SpssHAlignTypes.SpssHAlMixed	Mixed

Table 36. Horizontal alignment types (continued)

Type	Description
SpssClient.SpssHAlignTypes.SpssHAlDecimal	Decimal

SetHDecDigits Method

Sets the number of decimal digits for the selected cells of the pivot table.

Syntax

```
SpssPivotTable.SetHDecDigits(number)
```

Parameters

number. Number of decimal digits

SetLeftMargin Method

Sets the left margin of the selected cells in the pivot table.

Syntax

```
SpssPivotTable.SetLeftMargin(margin)
```

Parameters

margin. An integer. The unit is the point (1/72 inch). The maximum value is 36.

SetLegacyTableCompatible Method

Sets whether row and column labels are indexed in the same manner as for legacy tables. The argument is Boolean--*True* if row and column labels are indexed in the same manner as for legacy tables, *False* otherwise.

- By default, row and column labels are indexed in the same manner as for legacy tables.
- For tables with hidden rows or columns, you might need to specify `SetLegacyTableCompatible(False)`.
- `SetLegacyTableCompatible` has no effect on legacy tables.

Syntax

```
SpssPivotTable.SetLegacyTableCompatible(boolean)
```

SetRightMargin Method

Sets the right margin for the selected cells in the pivot table.

Syntax

```
SpssPivotTable.SetRightMargin(margin)
```

Parameters

margin. An integer. The unit is the point (1/72 inch). The maximum value is 36.

SetRotateColumnLabels Method

Rotates the category labels closest to the data (that is, categories of the column dimension with the largest index).

Syntax

```
SpssPivotTable.SetRotateColumnLabels(boolean)
```

Parameters

True. The column labels closest to the data are displayed vertically

False. The column labels closest to the data are displayed horizontally

SetRotateRowLabels Method

Rotates the labels of all but the last row dimension (that is, the row dimension with the largest index).

Syntax

```
SpssPivotTable.SetRotateRowLabels(boolean)
```

Parameters

True. The outer row labels are displayed vertically

False. The outer row labels are displayed horizontally

SetRowsToDisplayRowCount Method

Sets the number of rows to be displayed at a time for the current pivot table. Note that you must also call the DisplayTableByRows method with an argument of True to specify that the table is to be displayed a fixed number of rows at a time.

Note: This method is only available for legacy tables. See the topic "Compatibility with previous releases" on page 161 for more information.

Syntax

```
SpssPivotTable.SetRowsToDisplayRowCount(number)
```

Parameters

number. An integer specifying the number of rows to display at a time.

SetRowsToDisplayTolerance Method

Sets the widow/orphan tolerance to be used when displaying the table a fixed number of rows at a time (as set by the DisplayTableByRows method). The default is 0.

- If a break between blocks of rows leaves widow rows equal to or less than the specified tolerance, then the break point is shifted up in the table to display those rows in the next block.
- If a break between blocks of rows leaves orphan rows equal to or less than the specified tolerance, then the break point is shifted down in the table to display those rows in the previous block.
- If a break between blocks of rows leaves both widow and orphan rows equal to or less than the specified tolerance, then the break point is shifted up in the table to display the widow rows in the next block.

Note: This method is only available for legacy tables. See the topic "Compatibility with previous releases" on page 161 for more information.

Syntax

```
SpssPivotTable.SetRowsToDisplayTolerance(number)
```

Parameters

number. An integer specifying the widow/orphan tolerance.

SetTableLook Method

Applies a predefined table look.

Syntax

```
SpssPivotTable.SetTableLook(filename)
```

Parameters

filename. Path to the TableLook (.stt) file

On Windows, it is recommended to use raw strings for file paths, or replace backslashes with forward slashes (IBM SPSS Statistics accepts a forward slash for any backslash in a file specification). Raw strings are specified by prefacing the string with r, as in r'c:\examples\mydata.sav'. In raw mode, Python treats all backslashes in the string as the backslash character and not as the start of an escape sequence.

SetTextColor Method

Sets the color of the text in the selected cells of the pivot table.

Syntax

```
SpssPivotTable.SetTextColor(color)
```

Parameters

color. Integer representation of the color

For information on setting color values, see Appendix B, “Setting Color Values,” on page 249.

SetTextFont Method

Sets the font of the text in the selected cells of the pivot table.

Syntax

```
SpssPivotTable.SetTextFont(fontname)
```

Parameters

fontname. Name of the font family, as a string. Available fonts are accessed from Format>Table Properties in the pivot table editor.

SetTextHidden Method

Sets the hidden effect of the text in the selected cells of the pivot table.

Syntax

```
SpssPivotTable.SetTextHidden(boolean)
```

Parameters

True. Hidden

False. Not hidden

SetTextSize Method

Sets the font size of the text in the selected cells of the pivot table.

Syntax

```
SpssPivotTable.SetTextSize(size)
```

Parameters

size. Size in points (integer)

SetTextStyle Method

Sets the bold or italic style of the text in the selected cells of the pivot table.

Syntax

```
SpssPivotTable.SetTextStyle(style)
```

Table 37. Text style types

Type	Description
SpssClient.SpssTextStyleTypes.SpssTSRegular	Regular
SpssClient.SpssTextStyleTypes.SpssTSItalic	Italic
SpssClient.SpssTextStyleTypes.SpssTSBold	Bold
SpssClient.SpssTextStyleTypes.SpssTSBoldItalic	Bold Italic

SetTextUnderlined Method

Sets the underlined effect of the text in the selected cells of the pivot table.

Syntax

```
SpssPivotTable.SetTextUnderlined(boolean)
```

Parameters

True. Underlined

False. Not underlined

SetTitleText Method

Sets the text of the title for the pivot table.

Syntax

```
SpssPivotTable.SetTitleText(title)
```

Parameters

title. Text of the title

SetTopMargin Method

Sets the top margin of the selected cells in the pivot table.

Syntax

```
SpssPivotTable.SetTopMargin(margin)
```

Parameters

margin. An integer. The unit is the point (1/72 inch). The maximum value is 36.

SetUpdateScreen Method

Sets whether changes in the pivot table are refreshed immediately. The argument is Boolean--*True* if changes are refreshed immediately, *False* otherwise.

- By default, changes are refreshed immediately.

Syntax

```
SpssPivotTable.SetUpdateScreen(boolean)
```

Example

This example assumes that *PivotTable* is an `SpssPivotTable` object, and stops refreshing while looping through the row labels and making changes.

```
PivotTable.SetUpdateScreen(False)
rowlabels = PivotTable.RowLabelArray()
for i in range(rowlabels.GetNumRows()):
   for j in range(rowlabels.GetNumColumns()):
      if rowlabels.GetValueAt(i,j)=="Female":
         rowlabels.SetValueAt(i,j,"Women")
PivotTable.SetUpdateScreen(True)
```

Note: Setting the immediate refresh off (parameter set to *False*) prevents flashing when you make changes to individual cells in a loop (in internal scripting), but it may also prevent you assessing the results immediately. A better way is to avoid making changes cell by cell but select the cells and change the selection using a method on the pivot table object. This is also a faster way.

SetVAlign Method

Sets the vertical alignment of the text in the selected cells of the pivot table.

Syntax

```
SpssPivotTable.SetVAlign(alignment)
```

Table 38. Vertical alignment types

Type	Description
SpssClient.SpssVAlignTypes.SpssVAlTop	Top
SpssClient.SpssVAlignTypes.SpssVAlBottom	Bottom
SpssClient.SpssVAlignTypes.SpssVAlCenter	Center

SetVarNamesDisplay Method

This method sets how variable names are displayed in the pivot table: as variable names, as variable labels, or both.

Note: This method is not supported for legacy tables.

Syntax

```
SpssPivotTable.SetVarNamesDisplay(display)
```

Table 39. Settings for display.

Value	Description
SpssClient.VarNamesDisplay.Names	Names
SpssClient.VarNamesDisplay.Labels	Labels
SpssClient.VarNamesDisplay.Both	Names and labels

SetVarValuesDisplay Method

This method sets how variable values are displayed in the pivot table: as values, as value labels, or both.

Note: This method is not supported for legacy tables.

Syntax

```
SpssPivotTable.SetVarValuesDisplay(display)
```

Table 40. Settings for display.

Value	Description
SpssClient.VarValuesDisplay.Values	Values
SpssClient.VarValuesDisplay.Labels	Labels
SpssClient.VarValuesDisplay.Both	Values and labels

SetWidowOrphanLines Method

Sets the number of allowable widow/orphan lines when pivot tables are printed.

- Widow lines are the last few lines of a paragraph printed at the top of the next page; orphan lines are the first few lines of a paragraph printed at the bottom of the previous page.

Syntax

```
SpssPivotTable.SetWidowOrphanLines(number)
```

Parameters

number. Line limit (integer). The valid range is 1 to 100.

ShowAll Method

Shows all labels and data.

Syntax

```
SpssPivotTable.ShowAll()
```

ShowAllFootnotes Method

Shows all footnotes associated with the pivot table.

Syntax

```
SpssPivotTable.ShowAllFootnotes()
```

ShowCaption Method
Shows the caption of the pivot table.

Syntax

```
SpssPivotTable.ShowCaption()
```

ShowFootnote Method
Shows the hidden footnote(s) referenced by the selected label(s), data cell(s) or title.

- Ignored if no hidden footnote is referenced.

Syntax

```
SpssPivotTable.ShowFootnote()
```

ShowTitle Method
Shows the title of the pivot table.

Syntax

```
SpssPivotTable.ShowTitle()
```

Ungroup Method
Deletes selected group labels and ungroups the category or group labels in the deleted group(s).

- If all group labels on one level are removed, labels on the lower levels are promoted one level. (For column labels, the row index increases by one; for row labels, the column index increases by one.)
- Selection must be group labels.

Syntax

```
SpssPivotTable.Ungroup()
```

SpssDataCells Class
The `SpssDataCells` object provides access to the data cells of a pivot table. In most pivot tables, the data cells contain the results of the statistical analysis. You need to use the `SpssDataCells` object if you want to highlight significant values in the output (for example, making bold all correlation coefficients that are greater than a specified value) or to retrieve specific statistics from the output (for example, the means and standard deviations of each group or variable).

The `SpssDataCells` object represents a 2-dimensional array of the data cells you can view in a pivot table. If there are no layer dimensions, all of the cells will be accessible; otherwise, the table must be pivoted in order to fully access the data currently in layer dimensions.

The data cells array has the same number of rows as the row labels array and the same number of columns as the column labels array. That is to say, row indexes for the row labels and column indexes for the column labels respectively correspond to the row and column indexes for the data cells.

Note: If the current table has been set to display blocks of rows--either using `SET ROWSBREAK` or by checking **Display the table as blocks of rows** on the Pivot Tables tab of the Options dialog box--then methods of the `SpssDataCells` class that access specific cells, such as `GetTextColorAt`, will only have access to the first block of rows. Exceptions to this behavior are the `GetValueAt`, `SetValueAt`, and `GetUnformattedValueAt` methods, which can access all rows of the pivot table, regardless of whether the table is displayed in blocks of rows.

You get an `SpssDataCells` object from the DataCellArray method of an `SpssPivotTable` object, as in:

```
SpssDataCells = SpssPivotTable.DataCellArray()
```

Example: Modifying Specific Cells

This example assumes that `PivotTable` is an `SpssPivotTable` object, and sets the background color to red for all data cells containing a value below 0.01.

```
DataCells = PivotTable.DataCellArray()
for i in range(DataCells.GetNumRows()):
   for j in range(DataCells.GetNumColumns()):
      try:
         val = float(DataCells.GetValueAt(i,j))
         if  val < 0.01:
            DataCells.SetBackgroundColorAt(i,j,255)
      except:
         pass
```

- The value returned from `GetValueAt` is a unicode string. If the value is a representation of a numeric value, it is converted to a float, otherwise an exception is raised and control passes to the `except` clause. Since the `except` clause only contains a `pass` statement, execution continues.

GetBackgroundColorAt Method

Returns the background color of the specified data cell.

Syntax

```
SpssDataCells.GetBackgroundColorAt(row,column)
```

Parameters

row. Row index

column. Column index

Returns

The color is returned as an integer. See the topic Appendix B, “Setting Color Values,” on page 249 for more information.

GetBottomMarginAt Method

Returns the bottom margin of the specified data cell. The unit is the point (1/72 inch).

Syntax

```
SpssDataCells.GetBottomMarginAt(row,column)
```

Parameters

row. Row index

column. Column index

GetForegroundColorAt Method

This method is deprecated in release 17.0. Use the GetTextColorAt method instead.

GetHAlignAt Method

Returns the horizontal alignment of the specified data cell.

Syntax

```
SpssDataCells.GetHAlignAt(row,column)
```

Parameters

row. Row index

column. Column index

Returns

Table 41. Horizontal alignment types

Type	Description
SpssClient.SpssHAlignTypes.SpssHAlLeft	Left
SpssClient.SpssHAlignTypes.SpssHAlRight	Right
SpssClient.SpssHAlignTypes.SpssHAlCenter	Center
SpssClient.SpssHAlignTypes.SpssHAlMixed	Mixed
SpssClient.SpssHAlignTypes.SpssHAlDecimal	Decimal

GetHDecDigitsAt Method

Returns the number of decimal digits allowed in decimal alignment for the specified data cell.

Syntax

```
SpssDataCells.GetHDecDigitsAt(row,column)
```

Parameters

row. Row index

column. Column index

GetLeftMarginAt Method

Returns the left margin of the specified data cell. The unit is the point (1/72 inch).

Syntax

```
SpssDataCells.GetLeftMarginAt(row,column)
```

Parameters

row. Row index

column. Column index

GetNumColumns Method

Returns the number of columns in the `SpssDataCells` object.

Syntax

```
SpssDataCells.GetNumColumns()
```

GetNumericFormatAt method

Returns the display format for the numeric value in the specified data cell.

```
SpssDataCells.GetNumericFormatAt(row,column)
```

Parameters

row. Row index

column. Column index

Return Value

The string description of the format. For a listing of the format types, see Appendix D, "String Description of Numeric Formats," on page 253.

Note: To obtain detailed format information for custom currency formats use the `GetNumericFormatAtEx` method.

GetNumericFormatAtEx method

Returns an `SpssNumericFormat` object from which you can obtain detailed formatting information for a specified data cell, such as the prefix, separator, and suffix for a cell with a custom currency format.

```
SpssNumericFormat=SpssDataCells.GetNumericFormatAtEx(row,column)
```

Parameters

row. Row index

column. Column index

The `SpssNumericFormat` object supports two methods. `GetFormatListSize` indicates the number of format items available for retrieval--3 if the current cell has a custom currency format, and 1 otherwise. `GetFormatStringAt` retrieves a specified format item. It takes an integer (zero based) that specifies the index of the format item to retrieve.

GetFormatStringAt(0). If the list size is 3 then the returned value is the prefix of the value in the associated data cell; otherwise the returned value is the same as that returned from the GetNumericFormatAt method.

GetFormatStringAt(1). Returns the separator character of the format for the value in the associated data cell. Only available when the list size is greater than 1.

GetFormatStringAt(2). Returns the suffix of the value in the associated data cell. Only available when the list size is greater than 2.

GetNumRows Method

Returns the number of rows in the `SpssDataCells` object.

Syntax

```
SpssDataCells.GetNumRows()
```

GetReferredFootnotesAt Method

Returns an `SpssFootnotes` object, which allows access to all the footnotes referred to by the specified data cell.

- The footnotes array is a subset of the Footnotes object you can get from the pivot table. You can manipulate the subset using the same properties and methods, but the index of a footnote in this array is in no way related to the index of the same footnote when accessed from the pivot table.

Syntax

```
SpssFootnotes=SpssDataCells.GetReferredFootnotesAt(row,column)
```

Parameters

row. Row index

column. Column index

Example

This example gets the footnotes associated with the cell in the first row and first column of the data cell array and sets the text color and text style of the first footnote (index value 0) to red and bold respectively. It assumes that `PivotTable` is an `SpssPivotTable` object.

```
DataCells = PivotTable.DataCellArray()
Footnotes = DataCells.GetReferredFootnotesAt(0,0)
Footnotes.SetTextStyleAt(0,SpssClient.SpssTextStyleTypes.SpssTSBold)
Footnotes.SetTextColorAt(0,255)
```

GetRightMarginAt Method

Returns the right margin of the specified data cell. The unit is the point (1/72 inch).

Syntax

```
SpssDataCells.GetRightMarginAt(row,column)
```

Parameters

row. Row index

column. Column index

GetSigMarkersAt method

Returns the significance indicators, if any, for the specified data cell. Significance indicators are created for custom tables (`CTABLES`) when pairwise comparisons of column proportions or column means are calculated. The `GetSigMarkersAt` method returns significance indicators only when the indicators are merged into the main table.

```
SpssDataCells.GetSigMarkersAt(row,column)
```

Parameters

Row Row index.

Column
Column index.

Return Value

Simple indicators
When simple indicators are used, the returned value is a Python dictionary. The keys of the dictionary are the case-preserved values of the indicators for the specified cell. The value for each key is the number of the column, in the column label array, whose alphabetic identifier matches the key.

- When multiple statistics are specified for the custom table, the column number for each key is the column for the statistic that was used for the comparison.

- If two significance levels are specified for the pairwise comparisons, capital letters (for the indicators) are used to identify significance values less than or equal to the smaller level. Lowercase letters (for the indicators) are used to identify significance values less than or equal to the larger level.

APA-style indicators
When APA-style indicators are used, the returned value is a string that consists of the indicator for the specified cell.

You can get the indicator type from the `SpssPivotTable.GetSigMarkersType` method.

GetTextColorAt Method
Returns the color of the text in the specified data cell.

Syntax

```
SpssDataCells.GetTextColorAt(row,column)
```

Parameters

row. Row index

column. Column index

Returns

The color is returned as an integer. See the topic Appendix B, "Setting Color Values," on page 249 for more information.

GetTextFontAt Method
Returns the font of the text in the specified data cell, as a string.

Syntax

```
SpssDataCells.GetTextFontAt(row,column)
```

Parameters

row. Row index

column. Column index

GetTextHiddenAt Method
Returns the hidden effect of the text in the specified data cell. The result is Boolean.

Syntax

```
SpssDataCells.GetTextHiddenAt(row,column)
```

Parameters

row. Row index

column. Column index

Returns

True. Hidden

False. Not hidden

GetTextSizeAt Method

Returns the font size of the text in the specified data cell.

Syntax

```
SpssDataCells.GetTextSizeAt(row,column)
```

Parameters

row. Row index

column. Column index

GetTextStyleAt Method

Returns the bold or italic style of the text in the specified data cell.

Syntax

```
SpssDataCells.GetTextStyleAt(row,column)
```

Parameters

row. Row index

column. Column index

Returns

Table 42. Text style types

Type	Description
SpssClient.SpssTextStyleTypes.SpssTSRegular	Regular
SpssClient.SpssTextStyleTypes.SpssTSItalic	Italic
SpssClient.SpssTextStyleTypes.SpssTSBold	Bold
SpssClient.SpssTextStyleTypes.SpssTSBoldItalic	Bold Italic

GetTextUnderlinedAt Method

Returns the underlined effect of the text in the specified data cell. The result is Boolean.

Syntax

```
SpssDataCells.GetTextUnderlinedAt(row,column)
```

Parameters

row. Row index

column. Column index

Returns

True. Underlined

False. Not underlined

GetTopMarginAt Method

Returns the top margin of the specified data cell. The unit is the point (1/72 inch).

Syntax

```
SpssDataCells.GetTopMarginAt(row,column)
```

Parameters

row. Row index

column. Column index

GetVAlignAt Method

Returns the vertical alignment of the specified data cell.

Syntax

```
SpssDataCells.GetVAlignAt(row,column)
```

Parameters

row. Row index

column. Column index

Returns

Table 43. Vertical alignment types

Type	Description
SpssClient.SpssVAlignTypes.SpssVAlTop	Top
SpssClient.SpssVAlignTypes.SpssVAlBottom	Bottom
SpssClient.SpssVAlignTypes.SpssVAlCenter	Center

GetUnformattedValueAt Method

Returns the unformatted value of the specified data cell, as a unicode string. This allows you to obtain all available digits for a cell that contains a numeric value. In addition, any footnote markers associated with the cell are removed in the returned value. To obtain the value of the cell, formatted in the same manner as it appears in the pivot table, use the GetValueAt method.

Syntax

```
SpssDataCells.GetUnformattedValueAt(row,column)
```

Parameters

row. Row index

column. Column index

GetValueAt Method

Returns the value of the specified data cell, as a unicode string, and formatted in the same manner as it appears in the pivot table. To obtain an unformatted version of the cell, use the GetUnformattedValueAt method.

Syntax

```
SpssDataCells.GetValueAt(row,column,includeFootnotes)
```

Parameters

row. Row index

column. Column index

includeFootnotes. Optional Boolean specifying whether to include footnote markers in the returned value. The default is *True*.

HideFootnotesAt Method

Hides all footnotes referenced by the specified data cell.

Syntax

```
SpssDataCells.HideFootnotesAt(row,column)
```

Parameters

row. Row index

column. Column index

InsertNewFootnoteAt Method

Inserts a new footnote for the specified data cell.

Syntax

```
index=SpssDataCells.InsertNewFootnoteAt(row,column,string)
```

Parameters

row. Row index

column. Column index

string. New footnote text

Return Value

index. Integer (to be used to insert the footnote in other cells if it is a shared footnote)

Example

This example inserts a footnote for the cell in the first row and first column of the data cell array and inserts a shared footnote for each cell whose value is identical to this one. It assumes that `PivotTable` is an `SpssPivotTable` object.

```
DataCells = PivotTable.DataCellArray()
val = DataCells.GetUnformattedValueAt(0,0)
index = DataCells.InsertNewFootnoteAt(0,0,"My footnote")
for i in range(DataCells.GetNumRows()):
   for j in range(DataCells.GetNumColumns()):
      if DataCells.GetUnformattedValueAt(i,j) == val:
         DataCells.InsertSharedFootnoteAt(i,j,index)
```

InsertSharedFootnoteAt Method

Inserts a shared footnote (a footnote that applies to multiple data cells and/or labels) for the specified data cell.

Syntax

```
SpssDataCells.InsertSharedFootnoteAt(row,column,index)
```

Parameters

row. Row index.

column. Column index

index. The index (in the footnote array) of the desired footnote.

Note: When inserting a shared footnote along with a new footnote created with the `InsertNewFootnoteAt` method, you can use the index value returned by the `InsertNewFootnoteAt` method. See the topic "InsertNewFootnoteAt Method" on page 186 for more information.

ReSizeColumn Method

Resets the width of the current column.

Syntax

```
SpssDataCells.ReSizeColumn(column,width)
```

Parameters

column. Column index

width. An integer. The unit is the point (1/72 inch).

SelectCellAt Method

Selects the specified data cell, in addition to previously selected elements.

Syntax

```
SpssDataCells.SelectCellAt(row,column)
```

Parameters

row. Row index

column. Column index

SelectReferredFootnotesAt Method

Selects all the footnotes referenced by the specified data cell, in addition to previously selected elements.

Syntax

```
SpssDataCells.SelectReferredFootnotesAt(row,column)
```

Parameters

row. Row index

column. Column index

Note: This method is not available for legacy tables. To modify footnotes associated with a particular data cell in a legacy table, use the GetReferredFootnotesAt method to get an `SpssFootnotes` object containing the footnotes. You can then use the methods of the `SpssFootnotes` object to make the desired modifications.

SetBackgroundColorAt Method

Sets the background color of the specified data cell.

Syntax

```
SpssDataCells.SetBackgroundColorAt(row,column,color)
```

Parameters

row. Row index

column. Column index

color. Integer representation of the color

For information on setting color values, see Appendix B, "Setting Color Values," on page 249.

SetBottomMarginAt Method

Sets the bottom margin of the specified data cell.

Syntax

```
SpssDataCells.SetBottomMarginAt(row,column,margin)
```

Parameters

row. Row index

column. Column index

margin. An integer. The unit is the point (1/72 inch). The maximum value is 36.

SetForegroundColorAt Method

This method is deprecated in release 17.0. Use the SetTextColorAt method instead.

SetHAlignAt Method

Sets the horizontal alignment of the specified data cell.

Syntax

```
SpssDataCells.SetHAlignAt(row,column,alignment)
```

Parameters

row. Row index

column. Column index

Table 44. Horizontal alignment types

Type	Description
SpssClient.SpssHAlignTypes.SpssHAlLeft	Left
SpssClient.SpssHAlignTypes.SpssHAlRight	Right
SpssClient.SpssHAlignTypes.SpssHAlCenter	Center
SpssClient.SpssHAlignTypes.SpssHAlMixed	Mixed
SpssClient.SpssHAlignTypes.SpssHAlDecimal	Decimal

SetHDecDigitsAt Method

Sets the number of decimal digits for the specified data cell.

Syntax

```
SpssDataCells.SetHDecDigitsAt(row,column,number)
```

Parameters

row. Row index

column. Column index

number. Number of decimal digits

SetLeftMarginAt Method

Sets the left margin of the specified data cell.

Syntax

```
SpssDataCells.SetLeftMarginAt(row,column,margin)
```

Parameters

row. Row index

column. Column index

margin. An integer. The unit is the point (1/72 inch). The maximum value is 36.

SetNumericFormatAt method

Sets the display format for the numeric value in the current cell.

```
SpssDataCells.SetNumericFormatAt(row,column,format)
```

Parameters

row. Row index

column. Column index

format. The string description of the format.

For a listing of the format types, see Appendix D, “String Description of Numeric Formats,” on page 253.

SetNumericFormatAtWithDecimal method

Sets the display format for the numeric value in the current cell.

```
SpssDataCells.SetNumericFormatAtWithDecimal(row,column,format,decimal)
```

Parameters

row. Row index

column. Column index

format. The string description of the format.

decimal. Number of decimal places. The default is 0.

For a listing of the format types, see Appendix D, “String Description of Numeric Formats,” on page 253.

SetRightMarginAt Method

Sets the right margin of the specified data cell.

Syntax

```
SpssDataCells.SetRightMarginAt(row,column,margin)
```

Parameters

row. Row index

column. Column index

margin. An integer. The unit is the point (1/72 inch). The maximum value is 36.

SetTextColorAt Method

Sets the color of the text in the specified data cell.

Syntax

```
SpssDataCells.SetTextColorAt(row,column,color)
```

Parameters

row. Row index

column. Column index

color. Integer representation of the color

For information on setting color values, see Appendix B, “Setting Color Values,” on page 249.

SetTextFontAt Method

Sets the font of the text in the specified data cell.

Syntax

```
SpssDataCells.SetTextFontAt(row,column,fontname)
```

Parameters

row. Row index

column. Column index

fontname. Name of the font family, as a string. Available fonts are accessed from Format>Cell Properties in the pivot table editor.

SetTextHiddenAt Method

Sets the hidden effect of the text in the specified data cell.

Syntax

```
SpssDataCells.SetTextHiddenAt(row,column,boolean)
```

Parameters

row. Row index

column. Column index

boolean. True for hidden, False for not hidden

SetTextSizeAt Method

Sets the font size of the text in the specified data cell.

Syntax

```
SpssDataCells.SetTextSizeAt(row,column,size)
```

Parameters

row. Row index

column. Column index

size. Size in points (integer)

SetTextStyleAt Method

Sets the bold or italic style of the text in the specified data cell.

Syntax

```
SpssDataCells.SetTextStyleAt(row,column,style)
```

Parameters

row. Row index

column. Column index

Table 45. Text style types

Type	Description
SpssClient.SpssTextStyleTypes.SpssTSRegular	Regular
SpssClient.SpssTextStyleTypes.SpssTSItalic	Italic
SpssClient.SpssTextStyleTypes.SpssTSBold	Bold
SpssClient.SpssTextStyleTypes.SpssTSBoldItalic	Bold Italic

SetTextUnderlinedAt Method

Sets the underlined effect of the text in the specified data cell.

Syntax

```
SpssDataCells.SetTextUnderlinedAt(row,column,boolean)
```

Parameters

row. Row index

column. Column index

boolean. True for underlined, False for not underlined

SetTopMarginAt Method

Sets the top margin of the specified data cell.

Syntax

```
SpssDataCells.SetTopMarginAt(row,column,margin)
```

Parameters

row. Row index

column. Column index

margin. An integer. The unit is the point (1/72 inch). The maximum value is 36.

SetVAlignAt Method

Sets the vertical alignment of the specified data cell.

Syntax

```
SpssDataCells.SetVAlignAt(row,column,alignment)
```

Parameters

row. Row index

column. Column index

Table 46. Vertical alignment types

Type	Description
SpssClient.SpssVAlignTypes.SpssVAlTop	Top

Table 46. Vertical alignment types (continued)

Type	Description
SpssClient.SpssVAlignTypes.SpssVAlBottom	Bottom
SpssClient.SpssVAlignTypes.SpssVAlCenter	Center

SetValueAt Method

Sets the value of the specified data cell.

Syntax

```
SpssDataCells.SetValueAt(row,column,value)
```

Parameters

row. Row index

column. Column index

value. Value as a string

ShowFootnotesAt Method

Displays all the footnotes referenced by the specified data cell.

Syntax

```
SpssDataCells.ShowFootnotesAt(row,column)
```

Parameters

row. Row index

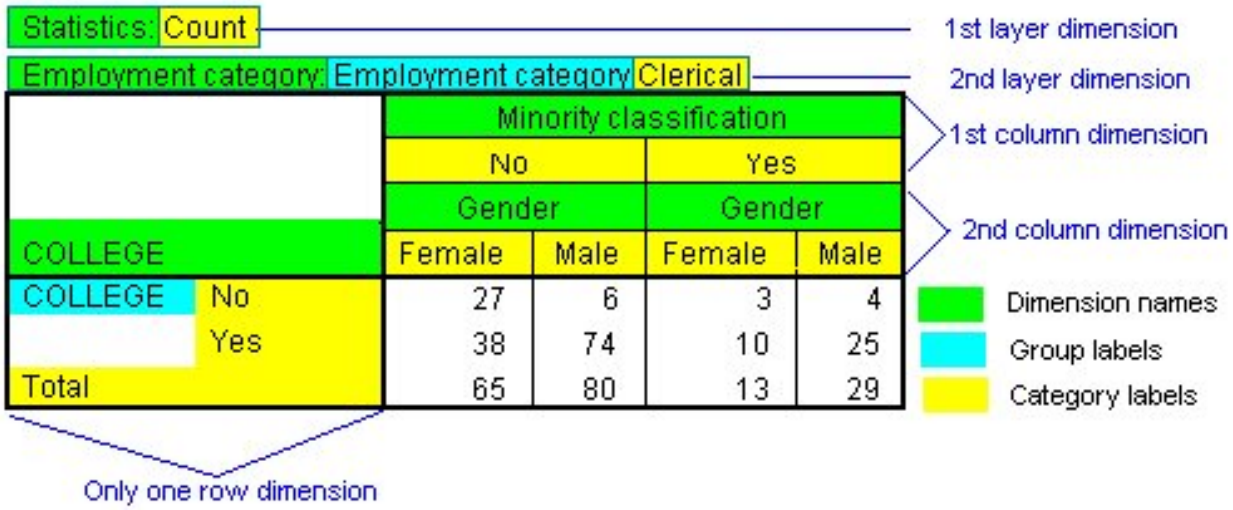
column. Column index

SpssDimension Class

The `SpssDimension` class provides access to a pivot table's dimensions. A pivot table can have three types of dimensions: column dimensions, row dimensions, and layer dimensions. Using an `SpssDimension` object you can obtain the name of a dimension, change the current category, or pivot the dimension.

Table Dimensions

The following is an illustration of the three types of dimension in a pivot table. To see the pivot table with all its labels (as shown in the figure), double-click it and select View>Show All from the menus in the pivot table editor.



To display the pivot trays (if they are not on the screen), select Pivot >Pivoting Trays from the menus.

Getting an SpssDimension Object

You get an `SpssDimension` object from the GetColumnDimension, GetLayerDimension, or GetRowDimension method of an `SpssPivotMgr` object. The `SpssPivotMgr` object is obtained from the `SpssPivotTable` object. For example, the following gets an `SpssDimension` object for the row dimension with index 1:

```
SpssPivotMgr = SpssPivotTable.PivotManager()
SpssDimension = SpssPivotMgr.GetRowDimension(1)
```

Example

This example assumes that `PivotTable` is an `SpssPivotTable` object and moves the "Statistics" row dimension to the first column dimension.

```
PivotManager = PivotTable.PivotManager()
# Search for the row dimension named "Statistics" and pivot it to
# the first column dimension.
for i in range(PivotManager.GetNumRowDimensions()):
    RowDim = PivotManager.GetRowDimension(i)
    if RowDim.GetDimensionName()  == "Statistics":
        RowDim.MoveToColumn(0)
        break
```

GetCategoryValueAt Method

Returns the label associated with the current category.

Syntax

```
SpssDimension.GetCategoryValueAt(index)
```

Parameters

index. Category index within the column or row dimension

GetCurrentCategory Method

Returns the index for the current category.

Syntax

```
SpssDimension.GetCurrentCategory()
```

GetDimensionName Method

Returns the dimension name.

Syntax

```
SpssDimension.GetDimensionName()
```

GetFullDimensionLabel Method

Returns the value of the label for the dimension, which is a concatenation of the dimension name, all the group labels (if any), and the label for the current category.

Syntax

```
SpssDimension.GetFullDimensionLabel()
```

GetNumCategories Method

Returns the number of categories in the dimension.

Syntax

```
SpssDimension.GetNumCategories()
```

HideLabel Method

Hides the dimension label.

Syntax

```
SpssDimension.HideLabel()
```

MoveToColumn Method

Pivots the dimension to the column, placing it before the specified column dimension.

Syntax

```
SpssDimension.MoveToColumn(index)
```

Parameters

index. Column dimension index

MoveToLayer Method

Pivots the dimension to the layer, placing it before the specified layer dimension.

Syntax

```
SpssDimension.MoveToLayer(index)
```

Parameters

index. Layer dimension index

MoveToRow Method

Pivots the dimension to the row, placing it before the specified row dimension.

Syntax

```
SpssDimension.MoveToRow(index)
```

Parameters

index. Row dimension index

SetCurrentCategory Method

Sets the specified category as current.

Syntax

```
SpssDimension.SetCurrentCategory(index)
```

Parameters

index. Category index

SetDimensionName Method

Sets the dimension name.

Syntax

```
SpssDimension.SetDimensionName(name)
```

SpssFootnotes Class

The SpssFootnotes class provides access to all of the footnotes contained in a pivot table. The index of a footnote does not correspond to the footnote marker but to the order of their references in the table. The index is returned when you insert a new footnote.

You get an SpssFootnotes object from the FootnotesArray method of an SpssPivotTable object, as in:

```
SpssFootnotes = SpssPivotTable.FootnotesArray()
```

An SpssFootnotes object is also returned by the GetReferredFootnotesAt method of an SpssDataCells or SpssLabels object. The footnote collection thus returned contains only the footnotes referred to by the specified data cell or label. The indexes for the returned collection are in the ordinal order of the references in the referring cell. IBM SPSS Statistics does not provide a method to go from a footnote referred to by a cell to the same footnote contained in the footnote array of the table.

Example

This example assumes that PivotTable is an SpssPivotTable object and sets the background color to yellow for all data cells that have footnotes.

```
Footnotes = PivotTable.FootnotesArray()
PivotTable.ClearSelection()
for i in range(Footnotes.GetCount()):
    Footnotes.SelectCellAt(i)
PivotTable.SetBackgroundColor(65535)
```

ChangeMarkerToRegular Method

Changes the marker of the current footnote to the regular marker. The marker is either alphabetic or numeric. The footnote marker type is set from the SetFootnoteMarkers method in the SpssPivotTable class.

Syntax

```
SpssFootnotes.ChangeMarkerToRegular(index)
```

Parameters

index. Index of the footnote

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

ChangeMarkerToSpecial Method

Changes the marker of the current footnote to the special marker.

Syntax

```
SpssFootnotes.ChangeMarkerToSpecial(index,newmarker)
```

Parameters

index. Index of the footnote

newmarker. Special marker for the footnote. The value is a string with a maximum length of two characters.

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

GetBackgroundColorAt Method

Returns the background color of the specified footnote.

Syntax

```
SpssFootnotes.GetBackgroundColorAt(index)
```

Parameters

index. Index of the footnote

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

Returns

The color is returned as an integer. See the topic Appendix B, “Setting Color Values,” on page 249 for more information.

Note: This method is not available for legacy tables.

GetBottomMarginAt Method

Returns the bottom margin of the specified footnote. The unit is the point (1/72 inch).

Syntax

```
SpssFootnotes.GetBottomMarginAt(index)
```

Parameters

index. Index of the footnote

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

Note: This method is not available for legacy tables.

GetCount Method

Returns the number of footnotes associated with the current pivot table.

Syntax

```
SpssFootnotes.GetCount()
```

GetForegroundColorAt Method

This method is deprecated in release 17.0 and obsolete for legacy tables in release 20 and higher. Use the GetTextColorAt method instead.

GetHAlignAt Method

Returns the horizontal alignment of the specified footnote.

Syntax

```
SpssFootnotes.GetHAlignAt(index)
```

Parameters

index. Index of the footnote

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

Returns

Table 47. Horizontal alignment types

Type	Description
SpssClient.SpssHAlignTypes.SpssHAlLeft	Left
SpssClient.SpssHAlignTypes.SpssHAlRight	Right
SpssClient.SpssHAlignTypes.SpssHAlCenter	Center
SpssClient.SpssHAlignTypes.SpssHAlMixed	Mixed
SpssClient.SpssHAlignTypes.SpssHAlDecimal	Decimal

Note: This method is not available for legacy tables.

GetLeftMarginAt Method

Returns the left margin for the specified footnote. The unit is the point (1/72 inch).

Syntax

```
SpssFootnotes.GetLeftMarginAt(index)
```

Parameters

index. Index of the footnote.

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

Note: This method is not available for legacy tables.

GetRightMarginAt Method

Returns the right margin of the specified footnote. The unit is the point (1/72 inch).

Syntax

```
SpssFootnotes.GetRightMarginAt(index)
```

Parameters

index. Index of the footnote

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

Note: This method is not available for legacy tables.

GetTextColorAt Method

Returns the color of the text of the specified footnote.

Syntax

```
SpssFootnotes.GetTextColorAt(index)
```

Parameters

index. Index of the footnote

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

Returns

The color is returned as an integer. See the topic Appendix B, “Setting Color Values,” on page 249 for more information.

GetTextFontAt Method

Returns the font of the text in the specified footnote, as a string.

Syntax

```
SpssFootnotes.GetTextFontAt(index)
```

Parameters

index. Index of the footnote

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

GetTextHiddenAt Method

Returns the hidden effect of the specified footnote. The result is a Boolean.

Syntax

```
SpssFootnotes.GetTextHiddenAt(index)
```

Parameters

index. Index of the footnote

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

Returns

True. Hidden

False. Not hidden

GetTextSizeAt Method

Returns the font size of the specified footnote.

Syntax

```
SpssFootnotes.GetTextSizeAt(index)
```

Parameters

index. Index of the footnote

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

GetTextStyleAt Method

Returns the bold or italic style of the text for specified footnote.

Syntax

```
SpssFootnotes.GetTextStyleAt(index)
```

Parameters

index. Index of the footnote

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

Returns

Table 48. Text style types

Type	Description
SpssClient.SpssTextStyleTypes.SpssTSRegular	Regular
SpssClient.SpssTextStyleTypes.SpssTSItalic	Italic
SpssClient.SpssTextStyleTypes.SpssTSBold	Bold

Table 48. Text style types (continued)

Type	Description
SpssClient.SpssTextStyleTypes.SpssTSBoldItalic	Bold Italic

GetTextUnderlinedAt Method

Returns the underlined effect of the specified footnote. The result is a Boolean.

Syntax

```
SpssFootnotes.GetTextUnderlinedAt(index)
```

Parameters

index. Index of the footnote

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

Returns

True. Underlined

False. Not underlined

Note: This method is not available for legacy tables.

GetTopMarginAt Method

Returns the top margin of the specified footnote. The unit is the point (1/72 inch).

Syntax

```
SpssFootnotes.GetTopMarginAt(index)
```

Parameters

index. Index of the footnote

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

Note: This method is not available for legacy tables.

GetVAlignAt Method

Returns the vertical alignment of the specified footnote.

Syntax

```
SpssFootnotes.GetVAlignAt(index)
```

Parameters

index. Index of the footnote

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

Returns

Table 49. Vertical alignment types

Type	Description
SpssClient.SpssVAlignTypes.SpssVAlTop	Top
SpssClient.SpssVAlignTypes.SpssVAlBottom	Bottom
SpssClient.SpssVAlignTypes.SpssVAlCenter	Center

Note: This method is not available for legacy tables.

GetValueAt Method

Returns the value associated with the specified footnote, as a unicode string.

Syntax

```
SpssFootnotes.GetValueAt(index)
```

Parameters

index. Index of the footnote

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

RenumberFootnotes Method

Renumbers all footnotes.

Syntax

```
SpssFootnotes.RenumberFootnotes()
```

SelectCellAt Method

Selects the data or label cell associated with the specified footnote, in addition to previously selected elements.

Syntax

```
SpssFootnotes.SelectCellAt(index)
```

Parameters

index. Index of the footnote

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

SetBackgroundColorAt Method

Sets the background color of the specified footnote.

Syntax

```
SpssFootnotes.SetBackgroundColorAt(index,color)
```

Parameters

index. Index of the footnote

color. Integer representation of the color

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

For information on setting color values, see Appendix B, "Setting Color Values," on page 249.

Note: This method is not available for legacy tables. You can set the background color of all footnotes in a legacy table by selecting all footnotes with the SpssPivotTable.SelectAllFootnotes method and then calling the SpssPivotTable.SetBackgroundColor method.

SetBottomMarginAt Method

Sets the bottom margin of the specified footnote.

Syntax

```
SpssFootnotes.SetBottomMarginAt(index,margin)
```

Parameters

index. Index of the footnote

margin. An integer. The unit is the point (1/72 inch). The maximum value is 36.

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

Note: This method is not available for legacy tables. You can set the bottom margin of the footnote area in a legacy table by selecting all footnotes with the SpssPivotTable.SelectAllFootnotes method and then calling the SpssPivotTable.SetBottomMargin method.

SetForegroundColorAt Method

This method is deprecated in release 17.0 and obsolete for legacy tables in release 20 and higher. Use the SetTextColorAt method instead.

SetHAlignAt Method

Sets the horizontal alignment of the specified footnote.

Syntax

```
SpssFootnotes.SetHAlignAt(index,alignment)
```

Parameters

index. Index of the footnote

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

Table 50. Horizontal alignment types

Type	Description
SpssClient.SpssHAlignTypes.SpssHAlLeft	Left
SpssClient.SpssHAlignTypes.SpssHAlRight	Right

Table 50. Horizontal alignment types (continued)

Type	Description
SpssClient.SpssHAlignTypes.SpssHAlCenter	Center
SpssClient.SpssHAlignTypes.SpssHAlMixed	Mixed
SpssClient.SpssHAlignTypes.SpssHAlDecimal	Decimal

Note: This method is not available for legacy tables. You can set the horizontal alignment of all footnotes in a legacy table by selecting all footnotes with the SpssPivotTable.SelectAllFootnotes method and then calling the SpssPivotTable.SetHAlign method.

SetLeftMarginAt Method

Sets the left margin for the specified footnote.

Syntax

```
SpssFootnotes.SetLeftMarginAt(index,margin)
```

Parameters

index. Index of the footnote.

margin. An integer. The unit is the point (1/72 inch). The maximum value is 36.

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

Note: This method is not available for legacy tables. You can set the left margin of the footnote area in a legacy table by selecting all footnotes with the SpssPivotTable.SelectAllFootnotes method and then calling the SpssPivotTable.SetLeftMargin method.

SetRightMarginAt Method

Sets the right margin of specified footnote.

Syntax

```
SpssFootnotes.SetRightMarginAt(index,margin)
```

Parameters

index. Index of the footnote

margin. An integer. The unit is the point (1/72 inch). The maximum value is 36.

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

Note: This method is not available for legacy tables. You can set the right margin of the footnote area in a legacy table by selecting all footnotes with the SpssPivotTable.SelectAllFootnotes method and then calling the SpssPivotTable.SetRightMargin method.

SetTextColorAt Method

Sets the color of the text of the specified footnote.

Syntax

```
SpssFootnotes.SetTextColorAt(index,color)
```

Parameters

index. Index of the footnote

color. Integer representation of the color

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

For information on setting color values, see Appendix B, "Setting Color Values," on page 249.

SetTextFontAt Method

Sets the font of the text in the specified footnote.

Syntax

```
SpssFootnotes.SetTextFontAt(index,fontname)
```

Parameters

index. Index of the footnote

fontname. Name of the font family, as a string. Available fonts are accessed from Format>Cell Properties in the pivot table editor.

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

SetTextHiddenAt Method

Sets the hidden effect of the specified footnote.

Syntax

```
SpssFootnotes.SetTextHiddenAt(index,boolean)
```

Parameters

index. Index of the footnote

boolean. True for hidden, False for not hidden

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

SetTextSizeAt Method

Sets the font size of the specified footnote.

Syntax

```
SpssFootnotes.SetTextSizeAt(index,size)
```

Parameters

index. Index of the footnote

size. Size in points (integer)

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

SetTextStyleAt Method

Sets the bold or italic style of the text for the specified footnote.

Syntax

```
SpssFootnotes.SetTextStyleAt(index,style)
```

Parameters

index. Index of the footnote

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

Table 51. Text style types

Type	Description
SpssClient.SpssTextStyleTypes.SpssTSRegular	Regular
SpssClient.SpssTextStyleTypes.SpssTSItalic	Italic
SpssClient.SpssTextStyleTypes.SpssTSBold	Bold
SpssClient.SpssTextStyleTypes.SpssTSBoldItalic	Bold Italic

SetTextUnderlinedAt Method

Sets the underlined effect of the specified footnote.

Syntax

```
SpssFootnotes.SetTextUnderlinedAt(index,boolean)
```

Parameters

index. Index of the footnote

boolean. True for underlined, False for not underlined

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

Note: This method is not available for legacy tables. You can set the underlined effect of all footnotes in a legacy table by selecting all footnotes with the SpssPivotTable.SelectAllFootnotes method and then calling the SpssPivotTable.SetTextUnderlined method.

SetTopMarginAt Method

Sets the top margin of the specified footnote.

Syntax

```
SpssFootnotes.SetTopMarginAt(index,margin)
```

Parameters

index. Index of the footnote

margin. An integer. The unit is the point (1/72 inch). The maximum value is 36.

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

Note: This method is not available for legacy tables. You can set the top margin of the footnote area in a legacy table by selecting all footnotes with the SpssPivotTable.SelectAllFootnotes method and then calling the SpssPivotTable.SetTopMargin method.

SetVAlignAt Method

Sets the vertical alignment of the specified footnote.

Syntax

```
SpssFootnotes.SetVAlignAt(index,alignment)
```

Parameters

index. Index of the footnote

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

Table 52. Vertical alignment types

Type	Description
SpssClient.SpssVAlignTypes.SpssVAlTop	Top
SpssClient.SpssVAlignTypes.SpssVAlBottom	Bottom
SpssClient.SpssVAlignTypes.SpssVAlCenter	Center

Note: This method is not available for legacy tables. You can set the vertical alignment of all footnotes in a legacy table by selecting all footnotes with the SpssPivotTable.SelectAllFootnotes method and then calling the SpssPivotTable.SetVAlign method.

SetValueAt Method

Sets the value associated with the specified footnote.

Syntax

```
SpssFootnotes.SetValueAt(index,value)
```

Parameters

index. Index of the footnote

value. String

The index of a footnote does not correspond to the footnote marker but to the order of their references in the table.

SpssLabels Class

The `SpssLabels` class provides access to the row labels and column labels contained in a pivot table. You need to use this object to format column or row labels (for example, making all "Total" labels bold) or to

change labels (for example, changing the "Column %", "Row %" or "Total %" label). Generally speaking, you need to get the specified column or row label in order to locate specific statistics in an `SpssDataCells` object.

The row and column labels are represented as 2-dimensional arrays, referred to as the **row labels array** and **column labels array**. The arrays contain all row and column labels for the pivot table, including hidden labels. The row labels array has the same number of rows as the data cells array and the column labels array has the same number of columns as the data cells array. Row indexes for the row labels and column indexes for the column labels respectively correspond to the row and column indexes for the data cells.

Column Labels Array

The following diagrams illustrate how the column labels array is indexed. Note that where you see only one label (such as the dimension name Statistics) in the pivot table, the label can be accessed in all cells corresponding to the categories under it. In the case of Statistics, you can access it using (0,0), (0,1), (0,2), (0,3), (0,4) or (0,5) and any change you make to one cell is reflected in all these cells.

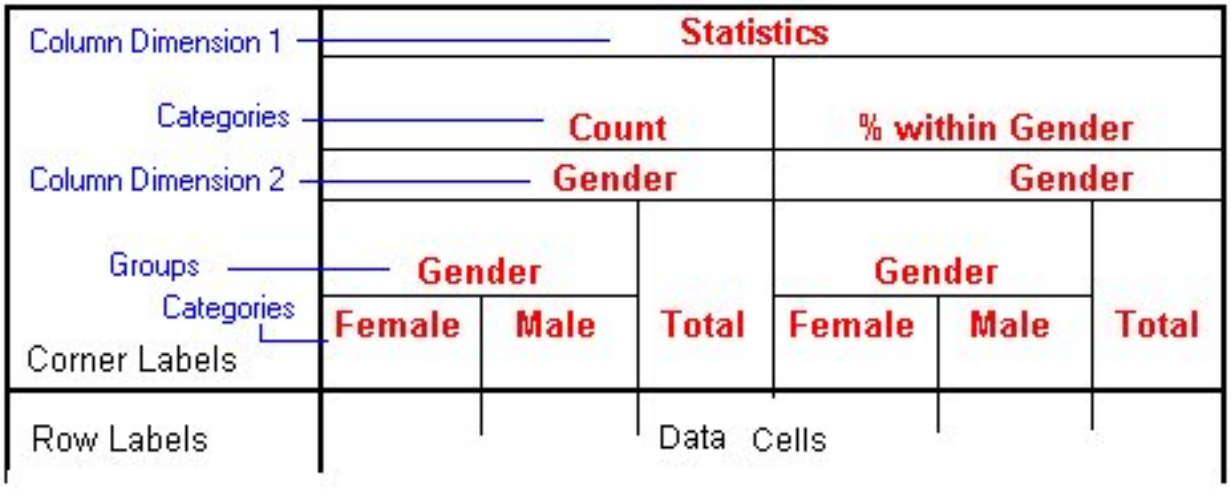


Figure 18. Column labels (after Showing All)

Column dimension 1	Statistics (0,0)	Statistics (0,1)	Statistics (0,2)	Statistics (0,3)	Statistics (0,4)	Statistics (0,5)
Categories	Count (1,0)	Count (1,1)	Count (1,2)	% Within Gender (1,3)	% Within Gender (1,4)	% Within Gender (1,5)
Column dimension 2	Gender (2,0)	Gender (2,1)	Gender (2,2)	Gender (2,3)	Gender (2,4)	Gender (2,5)
Groups	Gender (3,0)	Gender (3,1)	(3,2)	Gender (3,3)	Gender (3,4)	(3,5)
Categories	Female (4,0)	Male (4,1)	Total (4,2)	Female (4,3)	Male (4,4)	Total (4,5)
Corner Labels						
RowLabels	Data Cells					

Figure 19. Column labels array indexing

Each column dimension in the column labels array is represented by a set of **levels**. The first level is the dimension label, the last level contains the category labels, and all the levels in between (if any) contain group labels.

Row Labels Array

The following diagrams illustrate how the row labels array is indexed. Note that where you see only one label (such as the dimension name Statistics) in the pivot table, the label can be accessed in all cells corresponding to the categories under it. In the case of Statistics, you can access it using (0,0), (1,0), (2,0), (3,0), (4,0) and (5,0) and any change you make to one cell is reflected in all these cells.

Row dimension 1	Categories	Row dimension 2	Groups	Categories	Column Labels
Statistics	Count	Gender	Gender	Female	Data Cells
				Male	
				Total	
	% within Gender	Gender	Gender	Female	
				Male	
				Total	

Figure 20. Row labels (after showing all)

Row dimension 1	Categories	Row dimension 2	Groups	Categories	Column Labels
Statistics (0,0)	Count (0,1)	Gender (0,2)	Gender (0,3)	Female (0,4)	Data Cells
Statistics (1,0)	Count (1,1)	Gender (1,2)	Gender (1,3)	Male (1,4)	
Statistics (2,0)	Count (2,1)	Gender (2,2)	(2,3)	Total (2,4)	
Statistics (3,0)	% Within Gender (3,1)	Gender (3,2)	Gender (3,3)	Female (3,4)	
Statistics (4,0)	% Within Gender (4,1)	Gender (4,2)	Gender (4,3)	Male (4,4)	
Statistics (5,0)	% Within Gender (5,1)	Gender (5,2)	(5,3)	Total (5,4)	

Figure 21. Row labels array indexing

Each row dimension in the row labels array is represented by a set of **levels**. The first level is the dimension label, the last level contains the category labels, and all the levels in between (if any) contain group labels.

Notes

- To see all row and column labels in a pivot table, double-click on it and select View>Show All in the pivot table editor.
- Blank cells in the row label or column label arrays indicate that some categories (or subgroups) are not grouped.

Getting an SpssLabels Object

You get an `SpssLabels` object from the RowLabelArray or ColumnLabelArray method of an `SpssPivotTable` object, as in:

```
SpssLabels = SpssPivotTable.RowLabelArray()
```

Examples

This example assumes that `PivotTable` is an `SpssPivotTable` object, selects all row labels and bolds the text.

```
RowLabels = PivotTable.RowLabelArray()
for i in range(RowLabels.GetNumRows()):
   for j in range(1,RowLabels.GetNumColumns()):
      RowLabels.SetTextStyleAt(i,j,
         SpssClient.SpssTextStyleTypes.SpssTSBold)
```

This example assumes that `PivotTable` is an `SpssPivotTable` object, selects all column labels and bolds the text.

```
ColLabels = PivotTable.ColumnLabelArray()
for i in range(1,ColLabels.GetNumRows()):
   for j in range(ColLabels.GetNumColumns()):
      ColLabels.SetTextStyleAt(i,j,
         SpssClient.SpssTextStyleTypes.SpssTSBold)
```

BreakHere Method

Sets the break location for printing large pivot tables. The break occurs after a specified row or column label and only applies to the innermost row or column labels. Breaks are specified for a particular row or column label and do not apply to any repeated versions of the specified label.

Syntax

```
SpssLabels.BreakHere(index)
```

Parameters

index. For row labels, the index of the row (to break on) in the row labels array. For column labels, the index of the column (to break on) in the column labels array.

GetBackgroundColorAt Method

Returns the background color of the specified row/column label.

Syntax

```
SpssLabels.GetBackgroundColorAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

Returns

The color is returned as an integer. See the topic Appendix B, “Setting Color Values,” on page 249 for more information.

GetBottomMarginAt Method

Returns the bottom margin of the specified row/column label. The unit is the point (1/72 inch).

Syntax

```
SpssLabels.GetBottomMarginAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

GetColumnLabelWidthAt Method

Returns the width of the column labels for the level containing the current label. The unit is the point (1/72 inch).

Syntax

```
SpssLabels.GetColumnLabelWidthAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

GetForegroundColorAt Method

This method is deprecated in release 17.0. Use the GetTextColorAt method instead.

GetHAlignAt Method

Returns the horizontal alignment of the specified row/column label.

Syntax

```
SpssLabels.GetHAlignAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

Returns

Table 53. Horizontal alignment types

Type	Description
SpssClient.SpssHAlignTypes.SpssHAlLeft	Left
SpssClient.SpssHAlignTypes.SpssHAlRight	Right
SpssClient.SpssHAlignTypes.SpssHAlCenter	Center
SpssClient.SpssHAlignTypes.SpssHAlMixed	Mixed
SpssClient.SpssHAlignTypes.SpssHAlDecimal	Decimal

GetLeftMarginAt Method

Returns the left margin of the specified row/column label. The unit is the point (1/72 inch).

Syntax

```
SpssLabels.GetLeftMarginAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

GetNumColumns Method

Returns the number of columns in the row/column labels object.

Syntax

```
SpssLabels.GetNumColumns()
```

GetNumRows Method

Returns the number of rows in the row/column labels object.

Syntax

```
SpssLabels.GetNumRows()
```

GetReferredFootnotesAt Method

Returns an SpssFootnotes object, which allows access to all the footnotes referred to by the specified label cell.

- The returned footnotes array is a subset of the array returned by the FootnotesArray method of the SpssPivotTable class. You can manipulate the subset using the same properties and methods, but the index of a footnote in this array is in no way related to the index of the same footnote when accessed from the pivot table.

Syntax

```
SpssFootnotes=SpssLabels.GetReferredFootnotesAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

Example

This example gets the footnotes for the column label with row index 1 and column index 1 (in the column label array) and sets the text color and text style of the first footnote (index value 0) to red and bold respectively. It assumes that PivotTable is an SpssPivotTable object.

```
Labels = PivotTable.ColumnLabelArray()
Footnotes = Labels.GetReferredFootnotesAt(1,1)
Footnotes.SetTextStyleAt(0,SpssClient.SpssTextStyleTypes.SpssTSBold)
Footnotes.SetTextColorAt(0,255)
```

GetRightMarginAt Method

Returns the right margin of the specified row/column label. The unit is the point (1/72 inch).

Syntax

```
SpssLabels.GetRightMarginAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

GetRowLabelWidthAt Method

Returns the width of the row labels for the level containing the current label. The unit is the point (1/72 inch).

Syntax

```
SpssLabels.GetRowLabelWidthAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

GetTextColorAt Method

Returns the color of the text in the specified row/column label.

Syntax

```
SpssLabels.GetTextColorAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

Returns

The color is returned as an integer. See the topic Appendix B, "Setting Color Values," on page 249 for more information.

GetTextFontAt Method

Returns the font of the text in the specified row/column label, as a string.

Syntax

```
SpssLabels.GetTextFontAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

GetTextHiddenAt Method

Returns the hidden effect of the text for the specified row/column label. The result is a Boolean.

Syntax

```
SpssLabels.GetTextHiddenAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

Returns

True. Hidden

False. Not hidden

GetTextSizeAt Method

Returns the font size of the text for the specified row/column label.

Syntax

```
SpssLabels.GetTextSizeAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

GetTextStyleAt Method

Returns the bold or italic style of the text in the specified row/column label.

Syntax

```
SpssLabels.GetTextStyleAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

Returns

Table 54. Text style types

Type	Description
SpssClient.SpssTextStyleTypes.SpssTSRegular	Regular
SpssClient.SpssTextStyleTypes.SpssTSItalic	Italic
SpssClient.SpssTextStyleTypes.SpssTSBold	Bold
SpssClient.SpssTextStyleTypes.SpssTSBoldItalic	Bold Italic

GetTextUnderlinedAt Method

Returns the underlined effect of the text in the specified row/column label. The result is a Boolean.

Syntax

```
SpssLabels.GetTextUnderlinedAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

Returns

True. Underlined

False. Not underlined

GetTextWidthAt Method

Returns the width of the text in the indexed row/column label, as if the text were unwrapped. The unit is the point (1/72 inch).

Syntax

```
SpssLabels.GetTextWidthAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

GetTopMarginAt Method

Returns the top margin of the specified row/column label. The unit is the point (1/72 inch).

Syntax

```
SpssLabels.GetTopMarginAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

GetVAlignAt Method

Returns the vertical alignment of the specified row/column label.

Syntax

```
SpssLabels.GetVAlignAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

Returns

Table 55. Vertical alignment types

Type	Description
SpssClient.SpssVAlignTypes.SpssVAlTop	Top
SpssClient.SpssVAlignTypes.SpssVAlBottom	Bottom
SpssClient.SpssVAlignTypes.SpssVAlCenter	Center

GetValueAt Method

Returns the value of the specified row/column label, as a unicode string.

Syntax

```
SpssLabels.GetValueAt(row,column,includeFootnotes)
```

Parameters

row. Row index in the label array

column. Column index in the label array

includeFootnotes. Optional Boolean specifying whether to include footnote markers in the returned value. The default is *True*.

HideAllLabelsInDimensionAt Method

Hides all labels in the same dimension as the specified label.

Syntax

```
SpssLabels.HideAllLabelsInDimensionAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

HideFootnotesAt Method

Hides all footnotes referenced by the specified row/column label.

Syntax

```
SpssLabels.HideFootnotesAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

HideLabelsInDimensionAt Method

Hides all instances of the specified label within the dimension containing the label.

Syntax

```
SpssLabels.HideLabelsInDimensionAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

HideLabelsWithDataAt Method

Hides all instances of the specified label and all data associated with those instances. Only applies to the innermost labels.

Syntax

```
SpssLabels.HideLabelsWithDataAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

InsertBefore Method

Moves the selected column(s) or rows before a specified column or row. (The data are moved together with the labels.)

- The selected and specified labels must be in the same dimension and must be either category or group labels. (That is, they cannot be dimension names.)
- If no labels in the same dimension are selected, the method is ignored.

Syntax

```
SpssLabels.InsertBefore(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

InsertNewAfter Method

Inserts a new row or column, after the specified row or column, in the pivot table. To insert a new row, use this method with the row labels array. To insert a new column, use this method with the column labels array. The particular row or column is specified by providing the indexes of its associated label in the labels array. For example, to insert a column after the column whose label is "Mean", you provide the indexes of the label "Mean" in the column labels array.

- A plus sign "+" is inserted in the first cell of the new row or column to prevent the row or column from being automatically hidden because it is empty.
- In a table with nested or layered dimensions, a column or row is inserted at every corresponding dimension level.

Note: This method is not supported for legacy tables.

Syntax

```
SpssLabels.InsertNewAfter(row,column,label=None)
```

Parameters

row. Row index in the label array

column. Column index in the label array

label. An optional label for the new row or column. If omitted, a plus sign "+" is used for the label.

InsertNewBefore Method

Inserts a new row or column, before the specified row or column, in the pivot table. To insert a new row, use this method with the row labels array. To insert a new column, use this method with the column labels array. The particular row or column is specified by providing the indexes of its associated label in the labels array. For example, to insert a column before the column whose label is "Mean", you provide the indexes of the label "Mean" in the column labels array.

- A plus sign "+" is inserted in the first cell of the new row or column to prevent the row or column from being automatically hidden because it is empty.
- In a table with nested or layered dimensions, a column or row is inserted at every corresponding dimension level.

Note: This method is not supported for legacy tables.

Syntax

```
SpssLabels.InsertNewBefore(row,column,label=None)
```

Parameters

row. Row index in the label array

column. Column index in the label array

label. An optional label for the new row or column. If omitted, a plus sign "+" is used for the label.

InsertNewFootnoteAt Method

Inserts a new footnote for the specified row/column label.

Syntax

```
index=SpssLabels.InsertNewFootnoteAt(row,column,string)
```

Parameters

row. Row index in the label array

column. Column index in the label array

string. New footnote text

Return Value

index. Integer (to be used to insert the footnote in other cells if it is a shared footnote)

Example

This example inserts a footnote for the column label with row index 1 and column index 1 (in the column label array), and it also inserts a shared footnote for the column label with row index 1 and column index 2. It assumes that `PivotTable` is an `SpssPivotTable` object.

```
Labels = PivotTable.ColumnLabelArray()
index = Labels.InsertNewFootnoteAt(1,1,"My footnote")
Labels.InsertSharedFootnoteAt(1,2,index)
```

InsertSharedFootnoteAt Method

Inserts a shared footnote (a footnote that applies to multiple labels and/or data cells) for the specified row/column label.

Syntax

```
SpssLabels.InsertSharedFootnoteAt(row,column,index)
```

Parameters

row. Row index in the label array.

column. Column index in the label array

index. The index (in the footnote array) of the desired footnote.

Note: When inserting a shared footnote along with a new footnote created with the `InsertNewFootnoteAt` method, you can use the index value returned by the `InsertNewFootnoteAt` method. See the topic "InsertNewFootnoteAt Method" on page 219 for more information.

KeepTogether Method

Prevents a page break from occurring within the specified range when printing large pivot tables.

Syntax

```
SpssLabels.KeepTogether(from,to)
```

Parameters

from. For row labels, the index of the starting row in the row labels array. For column labels, the index of the starting column in the column labels array.

to. For row labels, the index of the ending row in the row labels array. For column labels, the index of the ending column in the column labels array.

RemoveBreakHere Method

Clears a previously set break location.

Syntax

```
SpssLabels.RemoveBreakHere(index)
```

Parameters

index. For row labels, the index of the row (in the row labels array) for which the break was set. For column labels, the index of the column (in the column labels array) for which the break was set.

RemoveKeepTogether Method

Negates the effects of a previous call to `KeepTogether`.

Syntax

```
SpssLabels.RemoveKeepTogether(from,to)
```

Parameters

from. For row labels, the index of the starting row in the row labels array. For column labels, the index of the starting column in the column labels array.

to. For row labels, the index of the ending row in the row labels array. For column labels, the index of the ending column in the column labels array.

SelectDataUnderLabelAt Method

Selects the data under the indexed label (but not the label), in addition to whatever has been selected previously.

Syntax

```
SpssLabels.SelectDataUnderLabelAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

SelectLabelAt Method

Selects the indexed label, in addition to previously selected elements.

Syntax

```
SpssLabels.SelectLabelAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

SelectLabelDataAt Method

Selects the indexed label and all corresponding data in the category, in addition to whatever has been selected previously.

Syntax

```
SpssLabels.SelectLabelDataAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

SelectReferredFootnotesAt Method

Selects all the footnotes referenced by the specified label cell, in addition to previously selected elements.

Syntax

```
SpssLabels.SelectReferredFootnotesAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

Note: This method is not available for legacy tables. To modify footnotes associated with a particular label in a legacy table, use the GetReferredFootnotesAt method to get an `SpssFootnotes` object containing the footnotes. You can then use the methods of the `SpssFootnotes` object to make the desired modifications.

SetBackgroundColorAt Method

Sets the background color of the specified row/column label.

Syntax

```
SpssLabels.SetBackgroundColorAt(row,column,color)
```

Parameters

row. Row index in the label array

column. Column index in the label array

color. Integer representation of the color

For information on setting color values, see Appendix B, "Setting Color Values," on page 249.

SetBottomMarginAt Method

Sets the bottom margin of the specified row/column label.

Syntax

```
SpssLabels.SetBottomMarginAt(row,column,margin)
```

Parameters

row. Row index in the label array

column. Column index in the label array

margin. An integer. The unit is the point (1/72 inch). The maximum value is 36.

SetColumnLabelWidthAt Method

Sets the width of the specified column label. This property will also set the widths of all column labels and data cells that are in the same column of the table as the specified label. To set column widths independently, use the `ReSizeColumn` method in the `SpssDataCells` class.

Syntax

```
SpssLabels.SetColumnLabelWidthAt(row,column,width)
```

Parameters

row. Row index in the label array

column. Column index in the label array

width. An integer. The unit is the point (1/72 inch).

SetForegroundColorAt Method

This method is deprecated in release 17.0. Use the SetTextColorAt method instead.

SetHAlignAt Method

Sets the horizontal alignment of the specified row/column label.

Syntax

```
SpssLabels.SetHAlignAt(row,column,alignment)
```

Parameters

row. Row index in the label array

column. Column index in the label array

Table 56. Horizontal alignment types

Type	Description
SpssClient.SpssHAlignTypes.SpssHAlLeft	Left
SpssClient.SpssHAlignTypes.SpssHAlRight	Right
SpssClient.SpssHAlignTypes.SpssHAlCenter	Center
SpssClient.SpssHAlignTypes.SpssHAlMixed	Mixed
SpssClient.SpssHAlignTypes.SpssHAlDecimal	Decimal

SetLeftMarginAt Method

Sets the left margin of the specified row/column label.

Syntax

```
SpssLabels.SetLeftMarginAt(row,column,margin)
```

Parameters

row. Row index in the label array

column. Column index in the label array

margin. An integer. The unit is the point (1/72 inch). The maximum value is 36.

SetRightMarginAt Method

Sets the right margin of the specified row/column label.

Syntax

```
SpssLabels.SetRightMarginAt(row,column,margin)
```

Parameters

row. Row index in the label array

column. Column index in the label array

margin. An integer. The unit is the point (1/72 inch). The maximum value is 36.

SetRowLabelWidthAt Method

Sets the width of the specified row label. This method will also set the widths of all row labels that are in the same column of the row label array as the specified label.

Syntax

```
SpssLabels.SetRowLabelWidthAt(row,column,width)
```

Parameters

row. Row index in the label array

column. Column index in the label array

width. An integer. The unit is the point (1/72 inch).

SetTextColorAt Method

Sets the color of the text in the specified row/column label.

Syntax

```
SpssLabels.SetTextColorAt(row,column,color)
```

Parameters

row. Row index in the label array

column. Column index in the label array

color. Integer representation of the color

For information on setting color values, see Appendix B, "Setting Color Values," on page 249.

SetTextFontAt Method

Sets the font of the text in the specified row/column label.

Syntax

```
SpssLabels.SetTextFontAt(row,column,fontname)
```

Parameters

row. Row index in the label array

column. Column index in the label array

fontname. Name of the font family, as a string. Available fonts are accessed from Format>Cell Properties in the pivot table editor.

SetTextHiddenAt Method

Sets the hidden effect of the text for the specified row/column label.

Syntax

```
SpssLabels.SetTextHiddenAt(row,column,boolean)
```

Parameters

row. Row index in the label array

column. Column index in the label array

boolean. True for hidden, False for not hidden. True hides the cell associated with the label.

SetTextSizeAt Method

Sets the font size of the text for the specified row/column label.

Syntax

```
SpssLabels.SetTextSizeAt(row,column,size)
```

Parameters

row. Row index in the label array

column. Column index in the label array

size. Size in points (integer)

SetTextStyleAt Method

Sets the bold or italic style of the text in the specified row/column label.

Syntax

```
SpssLabels.SetTextStyleAt(row,column,style)
```

Parameters

row. Row index in the label array

column. Column index in the label array

Table 57. Text style types

Type	Description
SpssClient.SpssTextStyleTypes.SpssTSRegular	Regular
SpssClient.SpssTextStyleTypes.SpssTSItalic	Italic
SpssClient.SpssTextStyleTypes.SpssTSBold	Bold
SpssClient.SpssTextStyleTypes.SpssTSBoldItalic	Bold Italic

SetTextUnderlinedAt Method

Sets the underlined effect of the text in the specified row/column label.

Syntax

```
SpssLabels.SetTextUnderlinedAt(row,column,boolean)
```

Parameters

row. Row index in the label array

column. Column index in the label array

boolean. True for underlined, False for not underlined.

SetTopMarginAt Method

Sets the top margin of the specified row/column label.

Syntax

```
SpssLabels.SetTopMarginAt(row,column,margin)
```

Parameters

row. Row index in the label array

column. Column index in the label array

margin. An integer. The unit is the point (1/72 inch). The maximum value is 36.

SetVAlignAt Method

Sets the vertical alignment of the specified row/column label.

Syntax

```
SpssLabels.SetVAlignAt(row,column,alignment)
```

Parameters

row. Row index in the label array

column. Column index in the label array

Table 58. Vertical alignment types

Type	Description
SpssClient.SpssVAlignTypes.SpssVAlTop	Top
SpssClient.SpssVAlignTypes.SpssVAlBottom	Bottom
SpssClient.SpssVAlignTypes.SpssVAlCenter	Center

SetValueAt Method

Sets the value of the specified row/column label.

Syntax

```
SpssLabels.SetValueAt(row,column,value)
```

Parameters

row. Row index in the label array

column. Column index in the label array

value. String

ShowAllLabelsAndDataInDimensionAt Method

Shows all labels and data in the dimension that contains the specified label.

Syntax

```
SpssLabels.ShowAllLabelsAndDataInDimensionAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

ShowAllLabelsInDimensionAt Method

Shows all labels in the dimension that contains the specified label.

Syntax

```
SpssLabels.ShowAllLabelsInDimensionAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

ShowFootnotesAt Method

Displays all the footnotes referenced by the specified row/column label.

Syntax

```
SpssLabels.ShowFootnotesAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

ShowHiddenDimensionLabelAt Method

Shows the hidden dimension label for the dimension that contains the specified label.

Syntax

```
SpssLabels.ShowHiddenDimensionLabelAt(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

Swap Method

Swaps the selected column(s) or rows with a specified column or row. The data are swapped together with the labels.

- The selected and specified labels must be in the same dimension and must be either category or group labels. (That is, they cannot be dimension names.)
- If no labels in the same dimenison are selected, the method is ignored.

Syntax

```
SpssLabels.Swap(row,column)
```

Parameters

row. Row index in the label array

column. Column index in the label array

SpssLayerLabels Class

The `SpssLayerLabels` class provides access to the layer labels contained in a pivot table. You need to use this object to format or change layer labels.

Layer Labels Array

The layer labels are represented as a 2-dimensional array, referred to as the **layer labels array**. Each row corresponds to the current category of one layer dimension. The first column is a concatenation of all the labels in the row, the second column is the dimension name, and the last column is the category label. Any columns between the second and last are group labels. You access other categories of the dimension from the `SpssDimension` object.

Minority classification: Yes

Gender: Total

Employment category: Employment category Clerical

		Educational level (years)						
		8	12	14	15	16	17	18
Statistics	Count	7	45	1	25	6	2	1

Figure 22. Layer labels displayed in a pivot table

Minority classification: Yes **(0,0)**	Minority classification **(0,1)**	**(0,2)**	Yes **(0,3)**
Gender: Total **(1,0)**	Gender **(1,1)**	**(1,2)**	Total **(1,3)**
Employment category: Employment category **(2,0)**	Employment category **(2,1)**	Employment category **(2,2)**	Clerical **(2,3)**
Displayed layer label	Dimension name	Group	Current category

Figure 23. Layer labels array indexing

Each layer dimension in the layer labels array is represented by a set of **levels**. The first level is the dimension name, the last level contains the current category label, and all the levels in between (if any) contain group labels. Blank cells in the layer labels array indicate that there are different numbers of levels in different layer dimensions.

Getting an SpssLayerLabels Object

You get an `SpssLayerLabels` object from the LayerLabelArray method of an `SpssPivotTable` object, as in:

```
SpssLayerLabels = SpssPivotTable.LayerLabelArray()
```

Example

This example assumes that `PivotTable` is an `SpssPivotTable` object and sets the background color of the label for the first layer dimension to yellow.

```
LayerLabels = PivotTable.LayerLabelArray()
LayerLabels.SetBackgroundColorAt(0,65535)
```

GetBackgroundColorAt Method

Returns the background color of the specified layer label.

Syntax

```
SpssLayerLabels.GetBackgroundColorAt(index)
```

Parameters

index. Index of the layer dimension

Returns

The color is returned as an integer. See the topic Appendix B, "Setting Color Values," on page 249 for more information.

GetBottomMarginAt Method

Returns the bottom margin of the specified layer label. The unit is the point (1/72 inch).

Syntax

```
SpssLayerLabels.GetBottomMarginAt(index)
```

Parameters

index. Index of the layer dimension

GetForegroundColorAt Method

This method is deprecated in release 17.0. Use the GetTextColorAt method instead.

GetHAlignAt Method

Returns the horizontal alignment of the specified layer label.

Syntax

```
SpssLayerLabels.GetHAlignAt(index)
```

Parameters

index. Index of the layer dimension

Returns

Table 59. Horizontal alignment types

Type	Description
SpssClient.SpssHAlignTypes.SpssHAlLeft	Left
SpssClient.SpssHAlignTypes.SpssHAlRight	Right
SpssClient.SpssHAlignTypes.SpssHAlCenter	Center
SpssClient.SpssHAlignTypes.SpssHAlMixed	Mixed
SpssClient.SpssHAlignTypes.SpssHAlDecimal	Decimal

GetLeftMarginAt Method

Returns the left margin of the label for the specified layer dimension. The unit is the point (1/72 inch).

Syntax

```
SpssLayerLabels.GetLeftMarginAt(index)
```

Parameters

index. Index of the layer dimension

GetNumDimensions Method

Returns the number of labels in the Layers (equal to the number of dimensions in the layers).

Syntax

```
SpssLayerLabels.GetNumDimensions()
```

GetNumLabelsWide Method

Returns the width (number of columns) of the Layer Labels array. The width equals the maximum depth of layer dimensions plus one.

Syntax

```
SpssLayerLabels.GetNumLabelsWide()
```

GetRightMarginAt Method

Returns the right margin of the label for the specified layer dimension. The unit is the point (1/72 inch).

Syntax

```
SpssLayerLabels.GetRightMarginAt(index)
```

Parameters

index. Index of the layer dimension

GetTextColorAt Method

Returns the color of the text in the label of the specified layer dimension.

Syntax

```
SpssLayerLabels.GetTextColorAt(index)
```

Parameters

index. Index of the layer dimension

Returns

The color is returned as an integer. See the topic Appendix B, “Setting Color Values,” on page 249 for more information.

GetTextFontAt Method

Returns the font of the text in the specified layer dimension, as a string.

Syntax

```
SpssLayerLabels.GetTextFontAt(index)
```

Parameters

index. Index of the layer dimension

GetTextHiddenAt Method

Returns the hidden effect of the specified layer dimension. The result is a Boolean.

Syntax

```
SpssLayerLabels.GetTextHiddenAt(index)
```

Parameters

index. Index of the layer dimension

Returns

True. Hidden

False. Not hidden

GetTextSizeAt Method

Returns the font size of the label for the specified layer dimension.

Syntax

```
SpssLayerLabels.GetTextSizeAt(index)
```

Parameters

index. Index of the layer dimension

GetTextStyleAt Method

Returns the bold or italic style of the text for the specified layer dimension.

Syntax

```
SpssLayerLabels.GetTextStyleAt(index)
```

Parameters

index. Index of the layer dimension

Returns

Table 60. Text style types

Type	Description
SpssClient.SpssTextStyleTypes.SpssTSRegular	Regular

Table 60. Text style types (continued)

Type	Description
SpssClient.SpssTextStyleTypes.SpssTSItalic	Italic
SpssClient.SpssTextStyleTypes.SpssTSBold	Bold
SpssClient.SpssTextStyleTypes.SpssTSBoldItalic	Bold Italic

GetTextUnderlinedAt Method

Returns the underlined effect of the specified layer dimension. The result is a Boolean.

Syntax

```
SpssLayerLabels.GetTextUnderlinedAt(index)
```

Parameters

index. Index of the layer dimension

Returns

True. Underlined

False. Not underlined

GetTopMarginAt Method

Returns the top margin of the label for the specified layer dimension. The unit is the point (1/72 inch).

Syntax

```
SpssLayerLabels.GetTopMarginAt(index)
```

Parameters

index. Index of the layer dimension

GetVAlignAt Method

Returns the vertical alignment of the label for the specified layer dimension.

Syntax

```
SpssLayerLabels.GetVAlignAt(index)
```

Parameters

index. Index of the layer dimension

Returns

Table 61. Vertical alignment types

Type	Description
SpssClient.SpssVAlignTypes.SpssVAlTop	Top
SpssClient.SpssVAlignTypes.SpssVAlBottom	Bottom
SpssClient.SpssVAlignTypes.SpssVAlCenter	Center

GetValueAt Method

Returns the value associated with the specified layer and column from the layer labels array, as a unicode string.

Syntax

```
SpssLayerLabels.GetValueAt(index,column)
```

Parameters

index. Index of the layer dimension

column. Column index of the cell in the layer labels array

HideFootnotesAt Method

Hides all footnotes referenced by the specified layer label.

Syntax

```
SpssLayerLabels.HideFootnotesAt(index)
```

Parameters

index. Index of the layer dimension

InsertNewFootnoteAt Method

Inserts a new footnote for the specified layer dimension.

Syntax

```
index=SpssLayerLabels.InsertNewFootnoteAt(index,string)
```

Parameters

index. Index of the layer dimension

string. New footnote text

Return Value

index. Integer (to be used to insert the footnote in other cells if it is a shared footnote)

Example

This example inserts a footnote for the layer dimension with index 0 (in the layer label array), and then inserts a shared footnote for the layer dimension with index 1. It assumes that `PivotTable` is an `SpssPivotTable` object.

```
Labels = PivotTable.LayerLabelArray()
index = Labels.InsertNewFootnoteAt(0,"My footnote")
Labels.InsertSharedFootnoteAt(1,index)
```

InsertSharedFootnoteAt Method

Inserts a shared footnote (a footnote that applies to multiple labels and/or data cells) for the specified layer dimension.

Syntax

```
SpssLayerLabels.InsertSharedFootnoteAt(dim,index)
```

Parameters

dim. Index of the layer dimension

index. The index (in the footnote array) of the desired footnote.

Note: When inserting a shared footnote along with a new footnote created with the `InsertNewFootnoteAt` method, you can use the index value returned by the `InsertNewFootnoteAt` method. See the topic "InsertNewFootnoteAt Method" on page 233 for more information.

SelectLabelAt Method

Selects the specified label, in addition to previously selected elements.

Syntax

```
SpssLayerLabels.SelectLabelAt(index)
```

Parameters

index. Index of the layer dimension

SelectReferredFootnotesAt Method

Selects all the footnotes referenced by the current layer label, in addition to previously selected elements.

Syntax

```
SpssLayerLabels.SelectReferredFootnotesAt(index)
```

Parameters

index. Index of the layer dimension

Note: This method is not available for legacy tables.

SetBackgroundColorAt Method

Sets the background color of the specified layer label.

Syntax

```
SpssLayerLabels.SetBackgroundColorAt(index,color)
```

Parameters

index. Index of the layer dimension

color. Integer representation of the color

For information on setting color values, see Appendix B, "Setting Color Values," on page 249.

SetBottomMarginAt Method

Sets the bottom margin of the specified layer label.

Syntax

```
SpssLayerLabels.SetBottomMarginAt(index,margin)
```

Parameters

index. Index of the layer dimension

margin. An integer. The unit is the point (1/72 inch). The maximum value is 36.

SetForegroundColorAt Method

This method is deprecated in release 17.0. Use the SetTextColorAt method instead.

SetHAlignAt Method

Sets the horizontal alignment of the specified layer label.

Syntax

```
SpssLayerLabels.SetHAlignAt(index,alignment)
```

Parameters

index. Index of the layer dimension

Table 62. Horizontal alignment types

Type	Description
SpssClient.SpssHAlignTypes.SpssHAlLeft	Left
SpssClient.SpssHAlignTypes.SpssHAlRight	Right
SpssClient.SpssHAlignTypes.SpssHAlCenter	Center
SpssClient.SpssHAlignTypes.SpssHAlMixed	Mixed
SpssClient.SpssHAlignTypes.SpssHAlDecimal	Decimal

SetLeftMarginAt Method

Sets the left margin of the label for the specified layer dimension.

Syntax

```
SpssLayerLabels.SetLeftMarginAt(index,margin)
```

Parameters

index. Index of the layer dimension

margin. An integer. The unit is the point (1/72 inch). The maximum value is 36.

SetRightMarginAt Method

Sets the right margin of the label for the specified layer dimension.

Syntax

```
SpssLayerLabels.SetRightMarginAt(index,margin)
```

Parameters

index. Index of the layer dimension

margin. An integer. The unit is the point (1/72 inch). The maximum value is 36.

SetTextColorAt Method

Sets the color of the text in the label of the specified layer dimension.

Syntax

```
SpssLayerLabels.SetTextColorAt(index,color)
```

Parameters

index. Index of the layer dimension

color. Integer representation of the color

For information on setting color values, see Appendix B, "Setting Color Values," on page 249.

SetTextFontAt Method

Sets the font of the text in the specified layer dimension.

Syntax

```
SpssLayerLabels.SetTextFontAt(index,fontname)
```

Parameters

index. Index of the layer dimension

fontname. Name of the font family, as a string. Available fonts are accessed from Format>Cell Properties in the pivot table editor.

SetTextHiddenAt Method

Sets the hidden effect of the label for the specified layer dimension.

Syntax

```
SpssLayerLabels.SetTextHiddenAt(index,boolean)
```

Parameters

index. Index of the layer dimension

boolean. True for hidden, False for not hidden

SetTextSizeAt Method

Sets the font size of the label of the specified layer dimension.

Syntax

```
SpssLayerLabels.SetTextSizeAt(index,size)
```

Parameters

index. Index of the layer dimension

size. Size in points (integer)

SetTextStyleAt Method

Sets the bold or italic style of the text for the specified layer dimension.

Syntax

```
SpssLayerLabels.SetTextStyleAt(index,style)
```

Parameters

index. Index of the layer dimension

Table 63. Text style types

Type	Description
SpssClient.SpssTextStyleTypes.SpssTSRegular	Regular
SpssClient.SpssTextStyleTypes.SpssTSItalic	Italic
SpssClient.SpssTextStyleTypes.SpssTSBold	Bold
SpssClient.SpssTextStyleTypes.SpssTSBoldItalic	Bold Italic

SetTextUnderlinedAt Method

Sets the underlined effect of the label for the specified layer dimension.

Syntax

```
SpssLayerLabels.SetTextUnderlinedAt(index,boolean)
```

Parameters

index. Index of the layer dimension

boolean. True for underlined, False for not underlined.

SetTopMarginAt Method

Sets the top margin of the label for the specified layer dimension.

Syntax

```
SpssLayerLabels.SetTopMarginAt(index,margin)
```

Parameters

index. Index of the layer dimension

margin. An integer. The unit is the point (1/72 inch). The maximum value is 36.

SetVAlignAt Method

Sets the vertical alignment of the label for the specified layer dimension.

Syntax

```
SpssLayerLabels.SetVAlignAt(index,alignment)
```

Parameters

index. Index of the layer dimension

Table 64. Vertical alignment types

Type	Description
SpssClient.SpssVAlignTypes.SpssVAlTop	Top
SpssClient.SpssVAlignTypes.SpssVAlBottom	Bottom
SpssClient.SpssVAlignTypes.SpssVAlCenter	Center

ShowFootnotesAt Method
Displays all the footnotes referenced by the label of the specified layer dimension.

Syntax

```
SpssLayerLabels.ShowFootnotesAt(index)
```

Parameters

index. Index of the layer dimension

SpssPivotMgr Class
The `SpssPivotMgr` class provides access to the row, column, and layer dimensions contained in a pivot table. By pivoting row dimensions to column dimensions, or column dimensions to layer dimensions, you can find the best way to present the results of the statistical analyses.

You get an `SpssPivotMgr` object from the PivotManager method of an `SpssPivotTable` object, as in:

```
SpssPivotMgr = SpssPivotTable.PivotManager()
```

For an example of using the `SpssPivotMgr` class, see "SpssDimension Class" on page 193.

GetColumnDimension Method
Returns an `SpssDimension` object for the specified column dimension.

Syntax

```
SpssDimension=SpssPivotMgr.GetColumnDimension(index)
```

Parameters

index. Index of the column dimension, where the value 0 refers to the innermost column dimension.

GetLayerDimension Method
Returns an `SpssDimension` object for the specified layer dimension.

Syntax

```
SpssDimension=SpssPivotMgr.LayerDimension(index)
```

Parameters

index. Index of the layer dimension

GetNumColumnDimensions Method
Returns the number of column dimensions.

Syntax

```
SpssPivotMgr.GetNumColumnDimensions()
```

GetNumLayerDimensions Method

Returns the number of layer dimensions.

Syntax

```
SpssPivotMgr.GetNumLayerDimensions()
```

GetNumRowDimensions Method

Returns the number of row dimensions.

Syntax

```
SpssPivotMgr.GetNumRowDimensions()
```

GetRowDimension Method

Returns an `SpssDimension` object for the specified row dimension.

Syntax

```
SpssDimension=SpssPivotMgr.GetRowDimension(index)
```

Parameters

index. Index of the row dimension, where the value 0 refers to the innermost row dimension.

MoveLayersToColumns Method

Moves all dimensions in layers to the outermost columns.

Syntax

```
SpssPivotMgr.MoveLayersToColumns()
```

MoveLayersToRows Method

Moves all dimensions in layers to the outermost rows.

Syntax

```
SpssPivotMgr.MoveLayersToRows()
```

TransposeRowsWithColumns Method

Moves all dimensions in the rows to the columns and moves all dimensions in the columns to the rows.

Syntax

```
SpssPivotMgr.TransposeRowsWithColumns()
```

Managing Remote Servers

SpssServerConf Class

The `SpssServerConf` class represents the configuration information for a server machine (may be an instance of IBM SPSS Statistics Server or the local computer). From the `SpssClient` object you can get an `SpssServerConf` object for the current server, the default server, the local computer, or you can get a list of `SpssServerConf` objects for all configured servers (includes the local computer).

Example: Connecting to a Server

```
import SpssClient
SpssClient.StartClient()
SpssServerConf = SpssClient.CreateNewServer("myservername",3016,"")
SpssServerConf.Connect("","myuserID","mypassword")
SpssClient.StopClient()
```

- The `CreateNewServer` method from the `SpssClient` class creates a new server configuration object for a server with a specified name (or IP address) on a specified port. It returns an `SpssServerConf` object.
- The `Connect` method of an `SpssServerConf` object establishes a connection to the server using the specified domain, user ID, and password.

Example: Configuring a New Server and Saving the Configuration

```
import SpssClient
SpssClient.StartClient()
ServerConfList = SpssClient.GetConfiguredServers()
SpssServerConf = SpssClient.CreateNewServer("myservername",3016,"")
ServerConfList.Add(SpssServerConf)
SpssServerConf = ServerConfList.GetItemAt(ServerConfList.Size()-1)
SpssServerConf.SetUserId("myuserID")
SpssServerConf.SetPassword("mypassword")
SpssServerConf.SetUserDomain("mydomain")
SpssServerConf.SetPasswordSaved(True)
SpssClient.SaveServers()
SpssClient.StopClient()
```

- `SpssClient.GetConfiguredServers()` gets an `SpssServerConfList` object that allows you to manage the list of configured servers.
- The `CreateNewServer` method from the `SpssClient` class creates a new server configuration object. The variable *SpssServerConf* is an `SpssServerConf` object.
- To add a new server configuration to the list, you use the `Add` method of the `SpssServerConfList` object.
- The user ID, password, and domain are set using the `SetUserId`, `SetPassword`, and `SetUserDomain` methods of the `SpssServerConf` object. The `SetPasswordSaved` method specifies that the password is to be saved for future use.
- The `SaveServers` method from the `SpssClient` class saves all server configurations so that they are available in future sessions.

Example: Connecting to a Server Using a Saved Configuration

```
import SpssClient
SpssClient.StartClient()
ServerConfList = SpssClient.GetConfiguredServers()
for i in range(ServerConfList.Size()):
    server = ServerConfList.GetItemAt(i)
    if server.GetServerName()=="myservername":
       server.ConnectWithSavedPassword()
SpssClient.StopClient()
```

- `SpssClient.GetConfiguredServers()` gets an `SpssServerConfList` object that provides access to the list of configured servers.
- The `GetItemAt` method of an `SpssServerConfList` object returns the `SpssServerConf` object at the specified index. Index values start from 0 and represent the order in which the servers were added to the list.
- The `ConnectWithSavedPassword` method uses the connection information (domain, user ID, and password) to connect to the server.

Connect Method

Attempts to connect to the associated instance of IBM SPSS Statistics Server using the provided domain, user ID, and password. Any existing connection to an instance of IBM SPSS Statistics Server is terminated.

Note: This method is not available when called from a Python program in distributed mode (Python programs make use of the interface exposed by the Python `spss` module).

Syntax

```
SpssServerConf.Connect(domain,userID,password)
```

Parameters

domain. A string specifying the domain of the user ID. Enter a blank string if the domain is not required.

userID. A string specifying the user ID.

password. A string specifying the password.

ConnectWithSavedPassword Method

Attempts to connect to the associated instance of IBM SPSS Statistics Server using the stored user domain, user ID, and password.

Note: This method is not available when called from a Python program in distributed mode (Python programs make use of the interface exposed by the Python `spss` module).

Syntax

```
SpssServerConf.ConnectWithSavedPassword()
```

Disconnect Method

Disconnects from the associated instance of IBM SPSS Statistics Server.

- The method has no effect when called on the local server.
- After calling the `Disconnect` method, you must connect to another server before calling other methods in the `SpssClient` module.
- It is not necessary to disconnect before connecting to a new server.

Note: This method is not available when called from a Python program in distributed mode (Python programs make use of the interface exposed by the Python `spss` module).

Syntax

```
SpssServerConf.Disconnect()
```

GetDescription Method

Returns the description text for the associated server.

Syntax

```
SpssServerConf.GetDescription()
```

GetServerName Method

Returns the machine name or IP address for the associated instance of IBM SPSS Statistics Server.

Syntax

```
SpssServerConf.GetServerName()
```

GetServerPort Method

Returns the port number for the associated instance of IBM SPSS Statistics Server.

Syntax

```
SpssServerConf.GetServerPort()
```

GetUserDomain Method

Returns the domain for the current user ID.

Syntax

```
SpssServerConf.GetUserDomain()
```

GetUserId Method

Returns the user ID if it is saved as part of the associated server configuration.

Syntax

```
SpssServerConf.GetUserId()
```

GetUseSSL Method

Indicates if SSL (**Secure Sockets Layer**) is in use for the associated instance of IBM SPSS Statistics Server. SSL is a commonly used protocol for managing the security of message transmission on the Internet. The result is Boolean--*True* if SSL is in use, *False* otherwise.

Syntax

```
SpssServerConf.GetUseSSL()
```

IsDefaultServer Method

Indicates whether the associated instance of IBM SPSS Statistics Server is set as the default server. The result is Boolean--*True* if this is the default server, *False* otherwise.

Syntax

```
SpssServerConf.IsDefaultServer()
```

IsEqualTo Method

Indicates if this server configuration object is the same object as a specified server configuration object. The result is Boolean--*True* if the two objects are identical, *False* otherwise.

Syntax

```
SpssServerConf.IsEqualTo(serverConf)
```

Parameters

serverConf. An `SpssServerConf` object

IsLocalServer Method

Indicates whether the associated instance of IBM SPSS Statistics Server represents the local server. The result is Boolean--*True* if this server is the local server, *False* otherwise.

Syntax

```
SpssServerConf.IsLocalServer()
```

IsPasswordSaved Method

Indicates whether the password is saved in the server configuration. The result is Boolean--*True* if the password is saved, *False* otherwise.

Syntax

```
SpssServerConf.IsPasswordSaved()
```

SetDefaultServer Method

Specifies whether the associated instance of IBM SPSS Statistics Server is set as the default server.

Syntax

```
SpssServerConf.SetDefaultServer(defaultServerFlag)
```

Parameters

defaultServerFlag. True to set as the default server, False otherwise.

SetDescription Method

Sets the description text for the associated server configuration.

Syntax

```
SpssServerConf.SetDescription(description)
```

SetPassword Method

Sets the password to be used by this server configuration.

Syntax

```
SpssServerConf.SetPassword(password)
```

SetPasswordSaved Method

Specifies whether the password is saved in the server configuration.

Syntax

```
SpssServerConf.SetPasswordSaved(savePassword)
```

Parameters

savePassword. True if the password is to be saved for future use, False otherwise.

SetServerName Method

Sets the machine name or IP address for the associated instance of IBM SPSS Statistics Server.

Syntax

```
SpssServerConf.SetServerName(serverName)
```

SetServerPort Method

Sets the port number for the associated instance of IBM SPSS Statistics Server.

Syntax

```
SpssServerConf.SetServerPort(port)
```

Parameters

port. An integer

SetUserDomain Method
Sets the domain for the current user ID.

Syntax

```
SpssServerConf.SetUserDomain(domain)
```

domain. A string

SetUserId Method
Sets the user ID for the associated server configuration.

Syntax

```
SpssServerConf.SetUserId(userId)
```

SetUseSSL Method
Specifies the setting for using SSL (**Secure Sockets Layer**) with the associated instance of IBM SPSS Statistics Server. SSL is a commonly used protocol for managing the security of message transmission on the Internet.

Syntax

```
SpssServerConf.SetUseSSL(useSSL)
```

Parameters

useSSL. True to use SSL, False otherwise.

SpssServerConfList Class
The `SpssServerConfList` class allows you to manage the list of configured servers, which includes the local computer. You obtain an `SpssServerConfList` object from the GetConfiguredServers method of the `SpssClient` class.

An `SpssServerConfList` object is not an iterable Python object. In order to iterate over the items in the list, use a `for` loop, as in:

```
for index in range(SpssServerConfList.Size()):
```

For an example that uses the `SpssServerConfList` class, see the example for the SpssServerConf class.

Add Method
Adds a server configuration to the list of available servers.

Syntax

```
SpssServerConfList.Add(serverConf)
```

Parameters

serverConf. An `SpssServerConf` object.

Server configuration objects are created with the `CreateNewServer` method in the `SpssClient` class.

Clear Method

Clears the list of server configurations, including the local computer.

Syntax

```
SpssServerConfList.Clear()
```

Contains Method

Indicates if the specified server configuration is a member of the list of available server configurations. The result is a Boolean--*True* if the specified server configuration object is equal to a member of the list of available server configuration objects, *False* otherwise.

Syntax

```
SpssServerConfList.Contains(serverConf)
```

Parameters

serverConf. An `SpssServerConf` object

GetItemAt Method

Returns an `SpssServerConf` object corresponding to the server configuration with the specified index. The index corresponds to the order in which the server configurations were created.

Syntax

```
SpssServerConf=SpssServerConfList.GetItemAt(index)
```

Remove Method

Removes the first occurrence of the specified server configuration from the list of available server configurations. There is no effect if the list does not contain the specified server configuration object.

Syntax

```
SpssServerConfList.Remove(serverConf)
```

Parameters

serverConf. An `SpssServerConf` object

RemoveItemAt Method

Removes the server configuration with the specified index from the list of available server configurations. The index corresponds to the order in which the server configurations were created.

Syntax

```
SpssServerConfList.RemoveItemAt(index)
```

Size Method

Returns the number of configured servers, including the local computer.

Syntax

```
SpssServerConfList.Size()
```

SpssScriptContext Class

The SpssScriptContext class provides access to the object that triggers an autoscript as well as the associated output document object. Autoscripts are scripts that run automatically when triggered by the creation of specific pieces of output from selected procedures. Scripts are specified as autoscripts and associated with output items (that trigger them) from the Scripts tab of the Options dialog.

SpssScriptContext objects are only for use when writing a script that will be used as an autoscript. They have a value of None if referenced by a script that is not being run as an autoscript.

You get an SpssScriptContext object from the GetScriptContext method of the SpssClient object.

Example: Get the Output Item that Triggered an Autoscript

```
import SpssClient
SpssClient.StartClient()
SpssScriptContext = SpssClient.GetScriptContext()
SpssOutputItem = SpssScriptContext.GetOutputItem()
```

The GetOutputItem method of the SpssScriptContext object returns the output item (SpssOutputItem object) that triggered the current autoscript.

GetOutputDoc Method

Returns an SpssOutputDoc object representing the output document associated with the current autoscript.

Syntax

```
SpssOutputDoc=SpssScriptContext.GetOutputDoc()
```

GetOutputItem Method

Returns an SpssOutputItem object representing the output item that triggered the current autoscript.

Syntax

```
SpssOutputItem=SpssScriptContext.GetOutputItem()
```

Note: To obtain an object of a specific output type, such as a pivot table or header item, from an SpssOutputItem object, call the GetSpecificType method of the SpssOutputItem object.

GetOutputItemIndex Method

Returns the index, in the associated output document, of the output item that triggered the current autoscript. The index corresponds to the order of the items in the output document, starting with 0 for the root item.

Syntax

```
SpssScriptContext.GetOutputItemIndex()
```

Appendix A. Variable Format Types

Table 65. Variable format types supported by IBM SPSS Statistics

Type	Description
1	**A.** Standard characters.
2	**AHEX.** Hexadecimal characters.
3	**COMMA.** Numbers with commas as the grouping symbol and a period as the decimal indicator. For example: 1,234,567.89.
4	**DOLLAR.** Numbers with a leading dollar sign ($), commas as the grouping symbol, and a period as the decimal indicator. For example: $1,234,567.89.
5	**F.** Standard numeric.
6	**IB.** Integer binary.
7	**PIBHEX.** Hexadecimal of PIB (positive integer binary).
8	**P.** Packed decimal.
9	**PIB.** Positive integer binary.
10	**PK.** Unsigned packed decimal.
11	**RB.** Real binary.
12	**RBHEX.** Hexadecimal of RB (real binary).
15	**Z.** Zoned decimal.
16	**N.** Restricted numeric.
17	**E.** Scientific notation.
20	**DATE.** International date of the general form dd-mmm-yyyy.
21	**TIME.** Time of the general form hh:mm:ss.ss.
22	**DATETIME.** Date and time of the general form dd-mmm-yyyy hh:mm:ss.ss.
23	**ADATE.** American date of the general form mm/dd/yyyy.
24	**JDATE.** Julian date of the general form yyyyddd.
25	**DTIME.** Days and time of the general form dd hh:mm:ss.ss.
26	**WKDAY.** Day of the week.
27	**MONTH.** Month.
28	**MOYR.** Month and year.
29	**QYR.** Quarter and year of the general form qQyyyy.
30	**WKYR.** Week and year.
31	**PCT.** Percentage sign after numbers.
32	**DOT.** Numbers with periods as the grouping symbol and a comma as the decimal indicator. For example: 1.234.567,89.
33	**CCA.** Custom currency format 1.
34	**CCB.** Custom currency format 2.
35	**CCC.** Custom currency format 3.
36	**CCD.** Custom currency format 4.
37	**CCE.** Custom currency format 5.
38	**EDATE.** European date of the general form dd.mm.yyyy.

Table 65. Variable format types supported by IBM SPSS Statistics (continued)

Type	Description
39	**SDATE.** Sortable date of the general form yyyy/mm/dd.
85	**MTIME.** Time of the general form mm:ss.ss.
86	**YMDHMS.** Date and time of the general form yyyy-mm-dd hh:mm:ss.ss.

Appendix B. Setting Color Values

Color values are expressed as integers. If you're accustomed to specifying colors in RGB format, you can convert to the associated integer using the following:

```
integer color value = R + G*(256) + B*(256^2)
```

where R, G, and B are the RGB values. For reference, following are some of the most common colors and their integer values:

Table 66. Color codes

Color	**Integer Value**
Black	0
Blue	16711680
Cyan	16776960
Green	65280
Magenta	16711935
Red	255
White	16777215
Yellow	65535

Appendix C. Export Options

Export options are retrieved from the GetExportOption method of the `SpssClient` class and set from the SetExportOption method of that class. The option identifiers have the form `SpssClient.ExportOptions.<option>`, where the available option values are listed below--for example, `SpssClient.ExportOptions.GraphExportType`. All of the settings are strings.

Table 67. Specifications for export options

Option	Valid Settings
ObjectsToExport	"all", "visible", "selected"
DocExportType	"excel", "html", "pdf", "plain", "utf8", "utf16", "word_rtf", "none" (graphics only)
DocFilePath	Export document file path
GraphExportType	"bmp", "emf", "eps", "jpg", "png", "tiff"
GraphFilePath	Export graph file path
XLSLayers	"all", "honor" (honors print layer setting), "visible"
XLSFootnotes	"No", "Yes"
HTMLayers	"all", "honor" (honors print layer setting), "visible"
HTMFootnotes	"No", "Yes"
WordRTFLayers	"all", "honor" (honors print layer setting), "visible"
WordRTFFootnotes	"No", "Yes"
TXTPlainTabsOrSpaces	"tabs", "spaces"
TXTPlainColumnWidthType	"autofit", "custom"
TXTPlainNoOfChars	Character representation of integer
TXTPlainRowBorderChar	Row border character
TXTPlainColBorderChar	Column border character
TxtPlainLayersInPivotTable	"all", "honor" (honors print layer setting), "visible"
TXTPlainFootnoteCaption	"No", "Yes"
TXTPlainInsertPageBreak	"No", "Yes"
TXTUTF8TabsOrSpaces	"tabs", "spaces"
TXTUTF8ColumnWidthType	"autofit", "custom"
TXTUTF8NoOfChars	Character representation of integer
TXTUTF8RowBorderChar	Row border character
TXTUTF8ColBorderChar	Column border character
TxtUTF8LayersInPivotTable	"all", "honor" (honors print layer setting), "visible"
TXTUTF8FootnoteCaption	"No", "Yes"
TXTUTF8InsertPageBreak	"No", "Yes"
TXTUTF16TabsOrSpaces	"tabs", "spaces"
TXTUTF16ColumnWidthType	"autofit", "custom"
TXTUTF16NoOfChars	Character representation of integer
TXTUTF16RowBorderChar	Row border character
TXTUTF16ColBorderChar	Column border character

Table 67. Specifications for export options (continued)

Option	Valid Settings
TxtUTF16LayersInPivotTable	"all", "honor" (honors print layer setting), "visible"
TXTUTF16FootnoteCaption	"No", "Yes"
TXTUTF16InsertPageBreak	"No", "Yes"
PDFOptimize	"No", "Yes"
PDFEmbedBookmarks	"No", "Yes"
PDFEmbedFonts	"No", "Yes"
PDFLayers	"all", "honor" (honors print layer setting), "visible"
JPEGSize	Character representation of image size in percent
JPEGGreyScale	"No", "Yes"
BMPSize	Character representation of image size in percent
BMPCompressImage	"no", "yes"
PNGSize	Character representation of image size in percent
PNGColorDepth	"current" (current screen depth), "bw" (black and white), "256gray", "16color", "256color", "24bit" (true color), "32bit" (true color)
TIFSize	Character representation of image size in percent
EPSSize	"physical_size" (same aspect ratio), "current_size"
EPSPercent	Character representation of image size in percent. Applies to "current_size".
EPSWidthPoints	Character representation of image width in points. Applies to "physical_size".
EPSPreviewImage	"no", "yes"
EPSFont	"replace_font", "use_font_ref"

Appendix D. String Description of Numeric Formats

If you are using a localized version of IBM SPSS Statistics, use the strings displayed in the Format list box on the Format Value tab (accessed from Format>Cell Properties in the pivot table editor).

Table 68. Specifications for numeric formats

String	Example / Description
#.#	1234.567
#.#;#.##E-#	1234.567(scientific notation if the cell is not wide enough)
#,###.##	1,234.567
#.###,##	1.234,567
#.##E+##	1.23E+03
##.#%	56.7%
dd.mmm.yy	28-OCT-94
dd-mmm-yyyy	28-OCT-1994
mm/dd/yy	10/28/94
mm/dd/yyyy	10/28/1994
dd.mm.yy	28.10.94
dd.mm.yyyy	28.10.1994
yy/mm/dd	94/10/28
yyyy/mm/dd	1994/10/28
yyddd	94301 (Julian date)
yyyyddd	1994301 (Julian date)
q Q yy	4 Q 94
q Q yyyy	4 Q 1994
mmm yy	OCT 94
mmm yyyy	OCT 1994
ww WK yy	43 WK 94
ww WK yyyy	43 WK 1994
dd-mmm-yyyy hh:mm	28-OCT-1994 08:03
dd-mmm-yyyy hh:mm:ss	28-OCT-1994 08:03:00
yyyy-mm-dd hh:mm	1994-10-28 08:03
yyyy-mm-dd hh:mm:ss	1994-10-28 08:03:00
ddd.hh.mm	301 20:03
ddd.hh/mm.ss.##	301 20:03:00.04
Monday, Tuesday...	Friday
January, February...	October
mm:ss	03:00
hh:mm	08:03
hh:mm:ss.##	08:03:00.04
$#,###.##	$1,234.56

You can create up to five custom currency display formats that can include special prefix and suffix characters and special treatment for negative values. The five custom currency format names are CCA, CCB, CCC, CCD, and CCE. The string can be used to specify the currency formats.

Appendix E. Preference Options

Preference options are retrieved from the GetPreference method of the `SpssClient` class and set from the SetPreference method of that class. The option identifiers have the form `SpssClient.PreferenceOptions.<option>`, where the available option values are listed below--for example, `SpssClient.PreferenceOptions.VariableListDisplay`. All of the settings are strings.

Table 69. General Options.

Option	Valid Settings
VariableListDisplay	"labels", "names"
VariableListSort	"alphabetical", "file", "measurement"
MeasurementSystem	"points", "inches", "centimeters"
Language	"Russian", "French", "German", "English", "Italian", "Japanese", "Korean", "Polish", "SChinese", "Spanish", "TChinese",
AutoRaise	"true", "false"
OutputScroll	"true", "false"
OutputSound	"system_beep", "none", "sound"
OutputSoundFile	Path to a custom sound file
ScientificNotation	"true", "false"
DigitGrouping	Option associated with SET/SHOW DIGITGROUPING. Settings are "true" or "false".
OpenSyntaxAtStartup	"true", "false"
OnlyOneDataset	"true", "false"
OXMLVersion	Option associated with SET/SHOW XVERSION. Settings are "default" or the Output XML schema version.
OutputAttributes	Option associated with SET/SHOW OATTRS. Settings are "olang", "eng", or "both".

Table 70. Viewer Options.

Option	Valid Settings
TitleFont	Font name, e.g. "Serif"
TitleFontSize	Character representation of integer
TitleFontBold	"true", "false"
TitleFontItalic	"true", "false"
TitleFontUnderline	"true", "false"
TitleFontColor	Character representation of integer color
PageTitleFont	Font name, e.g. "Serif"
PageTitleFontSize	Character representation of integer
PageTitleFontBold	"true", "false"
PageTitleFontItalic	"true", "false"
PageTitleFontUnderline	"true", "false"
PageTitleFontColor	Character representation of integer color

Table 70. Viewer Options (continued).

Option	Valid Settings
TextOutputFont	Font name, e.g. "Serif"
TextOutputFontSize	Character representation of integer
TextOutputFontBold	"true", "false"
TextOutputFontItalic	"true", "false"
TextOutputFontUnderline	"true", "false"
TextOutputFontColor	Character representation of integer color
DisplayCommandsLog	"Off", "On"
LogContents	"hidden", "shown"
WarningsContents	"hidden", "shown"
WarningsJustification	"align_left", "align_center", "align_right"
NotesContents	"hidden", "shown"
NotesJustification	"align_left", "align_center", "align_right"
TitleContents	"hidden", "shown"
TitleJustification	"align_left", "align_center", "align_right"
PageTitleContents	"hidden", "shown"
PageTitleJustification	"align_left", "align_center", "align_right"
PivotTableContents	"hidden", "shown"
PivotTableJustification	"align_left", "align_center", "align_right"
ChartContents	"hidden", "shown"
ChartJustification	"align_left", "align_center", "align_right"
TextOutputContents	"hidden", "shown"
TreeModelContents	"hidden", "shown"
GenericJustification	"align_left", "align_center", "align_right"
Orientation	1 (portrait), 2 (landscape)
LeftMargin	Character representation of positive number in units of MeasurementSystem
RightMargin	Character representation of positive number in units of MeasurementSystem
TopMargin	Character representation of positive number in units of MeasurementSystem
BottomMargin	Character representation of positive number in units of MeasurementSystem

Table 71. Data Options.

Option	Valid Settings
TransformationMergeOptions	"calculate_before_used", "calculate_immediately"
RandomNumberGenerator	"MC" (compatible with SPSS 12 and earlier), "MT" (Mersenne Twister)
DisplayFormatWidth	Character representation of integer between 1 and 40.
DisplayFormatDecimal	Character representation of integer between 0and 15.
ReadingExternalData	Character representation of integer between 1 and 40.

Table 71. Data Options (continued).

Option	Valid Settings
CenturyRangeValue	"Automatic", "custom"
CenturyRangeBeginYear	Character representation of integer between 1582 and 9900.

Table 72. File Locations.

Option	Valid Settings
RecordSyntax	"false", "true"
RecordMode	"append", "overwrite"
SessionJournalFile	Path to journal file
TempDir	Path to temp directory
RecentFiles	Recently used file list. Character representation of integer between 0 and 9.
DataFiles	Path to startup folder for data files on Open and Save dialogs
OtherFiles	Path to startup folder for other files on Open and Save dialogs
SpecifiedAndLastFolder	"true" (last folder used), "false" (specified folder)

Table 73. Currency Options.

Option	Valid Settings
CustomOutputFormat	"CCA", "CCB", "CCC", "CCD", "CCE"
AllValuesPrefix	All values prefix
AllValuesSuffix	All values suffix
NegativeValuesPrefix	Negative values prefix
NegativeValuesSuffix	Negative values suffix
DecimalSeparator	"comma", "period"

Table 74. Output Options.

Option	Valid Settings
OutlineVariables	"Names", "Labels", "Both"
OutlineVariableValues	"Values", "Labels", "Both"
PivotTableVariables	"Names", "Labels", "Both"
PivotTableVariableValues	"Values", "Labels", "Both"
OutputDisplay	"ModelViewer", "Tables"

Table 75. Chart Options.

Option	Valid Settings
ChartTemplate	"On", "Off"
ChartTemplateFile	Path to chart template file
ChartAspectRatio	Chart aspect ratio
ChartFont	Font name, e.g. "Arial"
ChartFrameInner	"true", "false"

Table 75. Chart Options (continued).

Option	Valid Settings
ChartFrameOuter	"true", "false"
GridLineScale	"true", "false"
GridLineCategory	"true", "false"
StyleCyclePref	"ColorsOnly", "PatternsOnly"

Table 76. Pivot Table Options.

Option	Valid Settings
ColumnWidth	"Labels", "Both" (labels and data)
EditingMode	"all_tables", "large_tables", "open_tables_window"
TableRender	"full", "fast", "light" (alias for "fast")

Note: For the `TableRender` option, `"light"` is deprecated for release 20 and higher, and has the same effect as `"fast"`.

Appendix F. Python Extension Commands for SPSS Statistics

IBM SPSS Statistics - Essentials for Python, which is installed by default with your IBM SPSS Statistics product, includes a set of extension commands that are implemented in Python and that provide capabilities beyond what is available with built-in SPSS Statistics procedures. Each extension command has an associated dialog box that generates command syntax for the command and is available from the SPSS Statistics menus. The extension commands can also be run from SPSS Statistics command syntax in the same manner as any built-in command such as `FREQUENCIES`.

Table 77. Listing of Python extension commands.

Menu location	Command name	Description
Data>Case Control Matching	FUZZY	Perform exact or fuzzy case-control matching.
File>Collect Variable Information	GATHERMD	Build a dataset of variable information from multiple datasets.
Analyze>Regression>Partial Least Squares	PLS	Partial least squares regression.
Data>Propensity Score Matching	PSM	Propensity score matching.
Utilities>Censor Table	SPSSINC CENSOR TABLES	Censor cells of a pivot table that is based on the values of a test statistic.
Transform>Create Dummy Variables	SPSSINC CREATE DUMMIES	Create a set of dummy variables that represent the values of a variable.
File>Open>Internet Data	SPSSINC GETURI DATA	Open an SPSS, Excel, SAS, or Stata dataset from a web url.
Utilities>Process Data Files	SPSSINC PROCESS FILES	Apply a file of syntax to a set of data files.
Edit>Search Data Files	SPSSINC PROCESS FILES SEARCH	Search the cases in a set of SPSS Statistics data files.
Data>Rake Weights	SPSSINC RAKE	Calculate weights to control totals in up to 10 dimensions by rim weighting, that is, raking.
Utilities>Define Variable Macro	SPSSINC SELECT VARIABLES	Define a macro listing variables selected according to variable dictionary properties.
Data>Split into Files	SPSSINC SPLIT DATASET	Split a dataset into separate files according to splitting variables.
Analyze>Compare Means>Summary Independent-Samples T Test	SPSSINC SUMMARY TTEST	Calculate a t test from sample summary information.
Transform>Programmability Transformation	SPSSINC TRANS	Apply a Python function to case data.
Analyze>Descriptive Statistics>TURF Analysis	SPSSINC TURF	Perform a TURF (Total Unduplicated Reach and Frequency) analysis.
Data>Adjust String Widths Across Files	STATS ADJUST WIDTHS	Adjust widths of string variables across files.
Analyze>Correlate>Canonical Correlation	STATS CANCORR	Calculate canonical correlations.
Analyze>Custom Tables>Define Category Order	STATS CATEGORY ORDER	Create a macro or multiple dichotomy set with a specified variable order.

Table 77. Listing of Python extension commands (continued).

Menu location	Command name	Description
Analyze>Classify>Cluster Silhouettes	STATS CLUS SIL	Compute silhouette measure for cluster analysis.
Graphs>Regression Variable Plots	STATS REGRESS PLOT	Plots useful in assessing regression relationships.
Graphs>Compare Subgroups	STATS SUBGROUP PLOTS	Graphically compare the distributions of a set of variables across a partition of the data.
Utilities>Calculate with Pivot Table	STATS TABLE CALC	Calculate with pivot table cells.
Graphs>Weibull Plot	STATS WEIBULL PLOT	Create Weibull probability plot for failure data.
Utilities>Create Text Output	TEXT	Create a text block in the Viewer, optionally with formatted text.

Notes

- Help for each of the Python extension commands is available by clicking **Help** on the associated dialog box. The help is not, however, integrated with the SPSS Statistics Help system.
- Complete syntax help for each of the extension commands is available by positioning the cursor within the command (in a syntax window) and pressing the F1 key. It is also available by running the command and including the `/HELP` subcommand. For example:

  ```
  STATS TABLE CALC /HELP.
  ```

  The command syntax help is not, however, integrated with the SPSS Statistics Help system and is not included in the *Command Syntax Reference*.

  Note: The F1 mechanism for displaying help is not supported in distributed mode.
- If the menu location that is specified for an extension command is not present in your IBM SPSS Statistics product, then look on the **Extensions** menu for the associated dialog.
- The dialogs were created with the Custom Dialog Builder in IBM SPSS Statistics. You can view the design for any of the dialogs and you can customize them using the Custom Dialog Builder. It is available from **Extensions>Utilities>Custom Dialog Builder (Compatibility mode)...**. To view the design for a dialog, choose **File>Open Installed** from within the Custom Dialog Builder.
- The implementation code (Python modules) and XML specification files for each of the Python extension commands can be found in the location where extension commands are installed on your computer. To view the location, run the `SHOW EXTPATHS` syntax command. The output displays a list of locations under the heading "Locations for extension commands". The files are installed to the first writable location in the list.
- Other extension commands that are not included in IBM SPSS Statistics - Essentials for Python are available for download from the Extension Hub, accessible from **Extensions>Extension Hub**. The Extension Hub also displays any updates that are available for the extension commands included with IBM SPSS Statistics - Essentials for Python in addition to updates for any other extensions that you installed.
- If you are installing extensions on SPSS Statistics Server, you can use a script to install multiple extensions at once. For information, see **Core System > Extensions> Installing local extension bundles > Batch installation of extension bundles** in the Help system.

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBMproducts. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_. All rights reserved.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Index

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Printed in USA