# R Integration Package for IBM SPSS Statistics

*Note*: Before using this information and the product it supports, read the general information under Notices on p. 79.

This edition applies to IBM® SPSS® Statistics 20 and to all subsequent releases and modifications until otherwise indicated in new editions.

Adobe product screenshot(s) reprinted with permission from Adobe Systems Incorporated.

Microsoft product screenshot(s) reprinted with permission from Microsoft Corporation.

Licensed Materials - Property of IBM

**© Copyright IBM Corporation 1989, 2011.**

# Contents

## 1 Using the R Integration Package for IBM SPSS Statistics 1

Working with R Program Blocks . . . 1
- R Syntax Rules . . . 4

Retrieving Variable Dictionary Information . . . 6
Reading Case Data from IBM SPSS Statistics . . . 7
Writing Results to a New IBM SPSS Statistics Dataset . . . 11
Creating Pivot Table Output . . . 13
Displaying Graphical Output from R . . . 15
Retrieving Output from Syntax Commands . . . 16
Localizing Output from R . . . 18
- Modifying the R code . . . 19
- Extracting translatable text . . . 20
- Translating the pot file . . . 21
- Installing the mo files . . . 21

## 2 R Integration Package for IBM SPSS Statistics: Functions and Classes 23

BasePivotTable Class . . . 24
- General Approach to Creating Pivot Tables . . . 26
- BasePivotTable Methods . . . 30
- CellText Objects . . . 40
- Creating a Warnings Table . . . 44

GetSPSSPlugInVersion Function . . . 45
GetSPSSVersion Function . . . 45
spssdata Functions . . . 45
- spssdata.CloseDataConnection Function . . . 45
- spssdata.GetCaseCount Function . . . 45
- spssdata.GetDataFromSPSS Function . . . 45
- spssdata.GetDataSetList Function . . . 48
- spssdata.GetFileHandles Function . . . 48
- spssdata.GetOpenedDataSetList Function . . . 48
- spssdata.GetSplitDataFromSPSS Function . . . 49
- spssdata.GetSplitVariableNames Function . . . 50
- spssdata.IsLastSplit Function . . . 50
- spssdata.SetDataToSPSS Function . . . 51

spssdictionary Functions . . . 51
- spssdictionary.CloseDataset Function . . . 51

© Copyright IBM Corporation 1989, 2011.

spssdictionary.CreateSPSSDictionary Function . . . . . 52
spssdictionary.EditCategoricalDictionary Function . . . . . 53
spssdictionary.EndDataStep Function . . . . . 54
spssdictionary.GetCategoricalDictionaryFromSPSS Function . . . . . 54
spssdictionary.GetDataFileAttributeNames Function . . . . . 55
spssdictionary.GetDataFileAttributes Function . . . . . 55
spssdictionary.GetDictionaryFromSPSS Function . . . . . 55
spssdictionary.GetMultiResponseSetNames Function . . . . . 57
spssdictionary.GetMultiResponseSet Function . . . . . 57
spssdictionary.GetUserMissingValues Function . . . . . 57
spssdictionary.GetValueLabels Function . . . . . 58
spssdictionary.GetVariableAttributeNames Function . . . . . 59
spssdictionary.GetVariableAttributes Function . . . . . 59
spssdictionary.GetVariableCount Function . . . . . 60
spssdictionary.GetVariableFormat Function . . . . . 60
spssdictionary.GetVariableFormatType Function . . . . . 61
spssdictionary.GetVariableLabel Function . . . . . 61
spssdictionary.GetVariableMeasurementLevel Function . . . . . 61
spssdictionary.GetVariableName Function . . . . . 62
spssdictionary.GetVariableType Function . . . . . 62
spssdictionary.GetWeightVariable Function . . . . . 62
spssdictionary.IsWeighting Function . . . . . 62
spssdictionary.SetActive Function . . . . . 63
spssdictionary.SetDataFileAttributes Function . . . . . 63
spssdictionary.SetDictionaryToSPSS Function . . . . . 63
spssdictionary.SetMultiResponseSet Function . . . . . 64
spssdictionary.SetUserMissing Function . . . . . 65
spssdictionary.SetValueLabel Function . . . . . 66
spssdictionary.SetVariableAttributes Function . . . . . 66
spsspivottable.Display Function . . . . . 67
spsspkg.EndProcedure Function . . . . . 70
spsspkg.GetOutputLanguage Function . . . . . 70
spsspkg.GetSPSSLocale Function . . . . . 70
spsspkg.GetSPSSPlugInVersion Function . . . . . 70
spsspkg.GetSPSSVersion Function . . . . . 70
spsspkg.GetStatisticsPath Function . . . . . 71
spsspkg.processcmd Function . . . . . 71
spsspkg.SetOutput Function . . . . . 71
spsspkg.SetOutputLanguage Function . . . . . 72
spsspkg.StartProcedure Function . . . . . 72
spsspkg.Syntax Function . . . . . 73
spsspkg.Template Function . . . . . 74
spsspkg.Version Function . . . . . 75

spssRGraphics Functions . . . . . 75
spssRGraphics.Submit Function . . . . . 75
spssRGraphics.SetOutput Function . . . . . 76
spssxmlworkspace Functions . . . . . 76
spssxmlworkspace.CreateXPathDictionary Function . . . . . 76
spssxmlworkspace.DeleteXmlWorkspaceObject Function . . . . . 76
spssxmlworkspace.EvaluateXPath Function . . . . . 77
spssxmlworkspace.GetHandleList Function . . . . . 77
TextBlock Class . . . . . 77
append Method . . . . . 78

## Appendix

A Notices 79

Index 81

Chapter

# 1

# Using the R Integration Package for IBM SPSS Statistics

The R Integration Package for IBM® SPSS® Statistics provides the ability to use R programming features within SPSS Statistics. This feature requires the IBM® SPSS® Statistics - Integration Plug-In for R, installed with IBM® SPSS® Statistics - Essentials for R.

Within SPSS Statistics, R programming features are available inside `BEGIN PROGRAM R-END PROGRAM` program blocks in command syntax, as well as from implementation code for extension commands implemented in R. Within these structures, you have access to both the R programming language and the functions specific to SPSS Statistics, provided in the R Integration Package for SPSS Statistics. These functions allow you to:

- Read case data from the active dataset into R.
- Get information about data in the active dataset.
- Get output results from syntax commands.
- Write results from R back to SPSS Statistics.

With these tools you have everything you need to create custom procedures in R. Tutorials are available by choosing Help>Working with R.

## Working with R Program Blocks

The keyword R on the `BEGIN PROGRAM` command identifies a block of R programming statements, which are processed by R.

The basic specification is `BEGIN PROGRAM R` followed by one or more R statements, followed by `END PROGRAM`.

#### Example

```
*R_multiple_program_blocks.sps.
DATA LIST FREE /var1.
BEGIN DATA
1
END DATA.
DATASET NAME File1.
BEGIN PROGRAM R.
File1N <- spssdata.GetCaseCount()
END PROGRAM.
DATA LIST FREE /var1.
BEGIN DATA
1
2
END DATA.
DATASET NAME File2.
BEGIN PROGRAM R.
```

© Copyright IBM Corporation 1989, 2011. 

```
File2N <- spssdata.GetCaseCount()
{if (File2N > File1N)
  message <- "File2 has more cases than File1."
else if (File1N > File2N)
  message <- "File1 has more cases than File2."
else
  message <- "Both files have the same number of cases."
}
cat(message)
END PROGRAM.
```

- The first program block defines a programmatic variable, *File1N*, with a value set to the number of cases in the active dataset.
- The first program block is followed by command syntax that creates and names a new active dataset. Although you cannot execute IBM® SPSS® Statistics command syntax from within an R program block, you can have multiple R program blocks separated by command syntax that performs any necessary actions. Values of R variables assigned in a given program block are available in subsequent program blocks.
- The second program block defines a programmatic variable, *File2N*, with a value set to the number of cases in the SPSS Statistics dataset named *File2*. The value of *File1N* persists from the first program block, so the two case counts can be compared in the second program block.
- The R function `cat` is used to display the value of the R variable *message*. Output written to R's standard output—for instance, with the `cat` or `print` function—is directed to a log item in the SPSS Statistics Viewer.

*Note*: To minimize R memory usage, you may want to delete large objects such as SPSS Statistics datasets at the end of your R program block—for example, `rm(data)`.

#### Displaying Output from R

For SPSS Statistics version 18 and higher, and by default, console output and graphics from R are redirected to the SPSS Statistics Viewer. This includes implicit output from R functions that would be generated when running those functions from within an R console—for example, the model coefficients and various statistics displayed by the `glm` function, or the mean value displayed by the `mean` function. You can toggle the display of output from R with the `spsspkg.SetOutput` function.

#### Accessing R Help Within IBM SPSS Statistics

You can access help for R functions from within SPSS Statistics. Simply include a call to the R `help` function in a `BEGIN PROGRAM R-END PROGRAM` block and run the block. For example:

```
BEGIN PROGRAM R.
help(paste)
END PROGRAM.
```

to obtain help for the R `paste` function.

You can access R's main html help page with:

```
BEGIN PROGRAM R.
help.start()
END PROGRAM.
```

### *Debugging*

For SPSS Statistics version 18 and higher, you can use the R `browser`, `debug`, and `undebug` functions within `BEGIN PROGRAM R-END PROGRAM` blocks, as well as from within implementation code for extension commands implemented in R. This allows you to use some of the same debugging tools available in an R console. Briefly, the `browser` function interrupts execution and displays a console window that allows you to inspect objects in the associated environment, such as variable values and expressions. The `debug` function is used to flag a specific R function—for instance, an R function that implements an extension command—for debugging. When the function is called, a console window is displayed and you can step through the function one statement at a time, inspecting variable values and expressions.

- Results displayed in a console window associated with use of the `browser` or `debug` function are displayed in the SPSS Statistics Viewer after the completion of the program block or extension command containing the function call.

  *Note*: When a call to a function that generates explicit output—such as the R `print` function—precedes a call to `browser` or `debug`, the resulting output is displayed in the SPSS Statistics Viewer after the completion of the program block or extension command containing the function call. You can cause such output to be displayed in the R console window associated with `browser` or `debug` by ensuring that the call to `browser` or `debug` precedes the function that generates the output and then stepping through the call to the output function.
- Use of the `debug` and `browser` functions is not supported in distributed mode.

For more information on the use of the `debug` and `browser` functions, see the R help for those functions.

### *R Functions that Read from stdin*

Some R functions take input data from an external file or from the standard input connection `stdin`. For example, by default, the `scan` function reads from `stdin` but can also read from an external file specified by the *file* argument. When working with R functions within `BEGIN PROGRAM R-END PROGRAM` blocks, reading data from `stdin` is not supported, due to the fact that R is embedded within SPSS Statistics. For such functions, you will need to read data from an external file. For example:

```
BEGIN PROGRAM R.
data <- scan(file="/Rdata.txt")
END PROGRAM.
```

### *Versions*

Multiple versions of the IBM® SPSS® Statistics - Integration Plug-In for R can be used on the same machine, each associated with a major version of SPSS Statistics, such as 19.0 or 20. `BEGIN PROGRAM R-END PROGRAM` blocks automatically load the correct version of the R Integration Package for SPSS Statistics, so there is no need to use the R `library` command to load the package.

#### *Syntax Rules*

- Within a program block, only statements recognized by the specified programming language are allowed.
- Within a program block, each line should not exceed 251 bytes.
- With the SPSS Statistics Batch Facility (available only with 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.

#### *Operations*

- Within a `BEGIN PROGRAM R` block, the R functions `quit()` and `q()` will terminate the SPSS Statistics session.

#### *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.
- R variables specified in a given program block persist to subsequent program blocks.
- You can nest a `BEGIN PROGRAM R` block in a `BEGIN PROGRAM PYTHON` block, but you cannot nest a Python program block in an R program block. For more information on nesting program blocks, see Introduction to Python Programs in the Help system.

## *R Syntax Rules*

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

**R is case-sensitive.**

This includes variable names, function names, and pretty much anything else you can think of. A variable name of *myRvariable* is not the same as *MyRVariable*, and the function `GetCaseCount()` cannot be written as `getcasecount()`.

**R uses a less than sign followed by a dash (<-) for assignment.**

For example:

```
var1 <- var2+1
```

**R commands are terminated with a semi-colon or new line; continuation lines do not require special characters or indentation.**

For example:

```
var1 <- var2+
```

```
3
```

is read as `var1<-var2+3`, since R continues to read input until a command is syntactically complete. However:

```
var1 <- var2
+3
```

will be read as two separate commands, and *var1* will be set to the value of *var2*.

**Groupings of statements are indicated by braces.** Groups of statements in structures such as loops, conditional expressions, and functions are indicated by enclosing the statements in braces, as in:

```
while (!spssdata.IsLastSplit()){
   data <- spssdata.GetSplitDataFromSPSS()
   cat("\nCases in Split: ",length(data[,1]))
}
```

#### *R Quoting Conventions*

- Strings in the R programming language can be enclosed in matching single quotes (`'`) or double quotes (`"`), as in SPSS Statistics.
- To specify an apostrophe (single quote) within a string, enclose the string in double quotes. For example,

  ```
  "Joe's Bar and Grille"
  ```

  is treated as

  ```
  Joe's Bar and Grille
  ```

- To specify quotation marks (double quote) within a string, use single quotes to enclose the string, as in

  ```
  'Categories Labeled "UNSTANDARD" in the Report'
  ```

- In the R programming language, doubled quotes of the same type as the outer quotes are not allowed. For example,

  ```
  'Joe''s Bar and Grille'
  ```

  results in an error.

**File Specifications.** Since escape sequences in the R programming language begin with a backslash (\)—such as `\n` for newline and `\t` for tab—it is recommended to use forward slashes (/) in file specifications on Windows. In this regard, SPSS Statistics always accepts a forward slash in file specifications.

```
spssRGraphics.Submit("/temp/R_graphic.jpg")
```

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

```
spssRGraphics.Submit("\\temp\\R_graphic.jpg")
```

## Retrieving Variable Dictionary Information

You can retrieve variable dictionary information from the active dataset using functions specific to each type of information (such as the variable label or the measurement level) or you can use the `spssdictionary.GetDictionaryFromSPSS` function to return results for a number of dictionary properties as an R data frame. For information on functions that retrieve specific dictionary properties, see the topic on spssdictionary functions.

#### Example

```
DATA LIST FREE /id (F4) gender (A1) training (F1).
VARIABLE LABELS id 'Employee ID'
   /training 'Training Level'.
VARIABLE LEVEL id (SCALE)
   /gender (NOMINAL)
   /training (ORDINAL).
VALUE LABELS training 1 'Beginning' 2 'Intermediate' 3 'Advanced'
   /gender 'f' 'Female' 'm' 'Male'.
BEGIN DATA
18 m 1
37 f 2
10 f 3
END DATA.
BEGIN PROGRAM R.
vardict <- spssdictionary.GetDictionaryFromSPSS()
print(vardict)
END PROGRAM.
```

#### Result

```
                             X1        X2             X3
varName                      id    gender       training
varLabel            Employee ID           Training Level
varType                       0         1              0
varFormat                    F4        A1             F1
varMeasurementLevel       scale   nominal        ordinal
```

Each column of the returned data frame contains the information for a single variable from the active dataset. The information for each variable consists of the variable name, the variable label, the variable type (0 for numeric variables, and an integer equal to the defined length for string variables), the display format, and the measurement level.

#### Working wth the Data Frame Representation of a Dictionary

The data frame returned by the `GetDictionaryFromSPSS` function contains the row labels *varName*, *varLabel*, *varType*, *varFormat*, and *varMeasurementLevel*. You can use these labels to specify the corresponding row. For example, the following code extracts the variable names:

```
varNames <- vardict["varName",]
```

It is often convenient to obtain separate lists of categorical and scale variables. The following code shows how to do this using the data frame representation of the IBM® SPSS® Statistics dictionary. The results are stored in the two R vectors *scaleVars* and *catVars*.

```
scaleVars<-vardict["varName",][vardict["varMeasurementLevel",]=="scale"]
catVars<-vardict["varName",][vardict["varMeasurementLevel",]=="nominal" |
                              vardict["varMeasurementLevel",]=="ordinal"]
```

## Reading Case Data from IBM SPSS Statistics

The `spssdata.GetDataFromSPSS` function reads case data from the IBM® SPSS® Statistics active dataset and, by default, stores it to an R data frame. You can choose to retrieve the cases for all variables or a selected subset of the variables in the active dataset. Variables are specified by name or by an index value representing position in the active dataset, starting with 0 for the first variable in file order.

#### Example: Retrieving Cases for All Variables

```
*R_get_all_cases.sps.
DATA LIST FREE /age (F4) income (F8.2) car (F8.2) employ (F4).
BEGIN DATA.
55  72  36.20  23
56  153 76.90  35
28  28  13.70  4
END DATA.
BEGIN PROGRAM R.
casedata <- spssdata.GetDataFromSPSS()
print(casedata)
END PROGRAM.
```

#### Result

```
  age income  car employ
1  55     72 36.2     23
2  56    153 76.9     35
3  28     28 13.7      4
```

Each column of the returned data frame contains the case data for a single variable from the active dataset. The column name is the variable name and can be used to extract the data for that variable, as in:

```
income <- casedata$income
```

Each row of the returned data frame contains the data for a single case. By default, the rows are labeled with consecutive integers. When calling `GetDataFromSPSS`, you can include the optional argument *row.label* to specify a variable from the active dataset whose case values will be the row labels of the resulting data frame.

#### Example: Retrieving Cases for Selected Variables

```
*R_get_specified_variables.sps.
DATA LIST FREE /age (F4) income (F8.2) car (F8.2) employ (F4).
BEGIN DATA.
55  72  36.20  23
56  153 76.90  35
28  28  13.70  4
END DATA.
BEGIN PROGRAM R.
casedata <- spssdata.GetDataFromSPSS(variables=c("age","income","employ"))
END PROGRAM.
```

The argument *variables* is an R vector specifying a subset of variables for which case data will be retrieved. In this example, the R function `c()` is used to create a character vector of variable names. The resulting R data frame (*casedata*) will contain the three columns labeled *age*, *income*, and *employ*.

You can use the `TO` keyword to specify a range of variables as you can in SPSS Statistics—for example, `variables=c("age TO car")`. If you prefer to work with variable index values (index values represent position in the dataset, starting with 0 for the first variable in file order), you can specify a range of variables with an expression such as `variables=c(0:2)`. The R code `c(0:2)` creates a vector consisting of the integers between `0` and `2` inclusive.

#### Example: Retrieving Categorical Variables

The analogue of a categorical variable in SPSS Statistics is a factor in R. You can specify that categorical variables are converted to factors, although by default they are not. To convert categorical variables to R factors, use the *factorMode* argument of the `GetDataFromSPSS` function.

```
*R_handle_catvars.sps.
DATA LIST FREE /id (F4) gender (A1) training (F1).
VARIABLE LABELS id 'Employee ID'
   /training 'Training Level'.
VARIABLE LEVEL id (SCALE)
   /gender (NOMINAL)
   /training (ORDINAL).
VALUE LABELS training 1 'Beginning' 2 'Intermediate' 3 'Advanced'
   /gender 'f' 'Female' 'm' 'Male'.
BEGIN DATA
18 m 1
37 f 2
10 f 3
22 m 2
END DATA.
BEGIN PROGRAM R.
casedata <- spssdata.GetDataFromSPSS(factorMode="labels")
casedata
END PROGRAM.
```

- The value `"labels"` for *factorMode*, used in this example, specifies that categorical variables are converted to factors whose levels are the value labels of the variables. The alternate value `"levels"` specifies that categorical variables are converted to factors whose levels are the values of the variables. For more information, see the topic spssdata.GetDataFromSPSS Function in Chapter 2 on p. 45.

#### Result

```
  id gender     training
1 18   Male    Beginning
2 37 Female Intermediate
3 10 Female     Advanced
4 22   Male Intermediate
```

*Note*: If you intend to write factors retrieved with `factorMode="labels"` to a new SPSS Statistics dataset, special handling is required. For more information, see the topic Writing Results to a New IBM SPSS Statistics Dataset on p. 11.

#### *Example: Handling IBM SPSS Statistics Datetime Values*

When retrieving values of SPSS Statistics variables with date or datetime formats, you'll most likely want to convert the values to R date/time (POSIXt) objects. By default, such variables are not converted and are simply returned in the internal representation used by SPSS Statistics (floating point numbers representing some number of seconds and fractional seconds from an initial date and time). To convert variables with date or datetime formats to R date/time objects, you use the *rDate* argument of the `GetDataFromSPSS` function.

```
*R_retrieve_datetime_values.sps.
DATA LIST FREE /bdate (ADATE10).
BEGIN DATA
05/02/2009
END DATA.
BEGIN PROGRAM R.
data<-spssdata.GetDataFromSPSS(rDate="POSIXct")
data
END PROGRAM.
```

#### *Result*

```
       bdate
1 2009-05-02
```

#### *Example: Missing Data*

By default, missing values for numeric variables (user-missing and system-missing) are converted to the R *NaN* value and user-missing values of string variables are converted to the R *NA* value.

```
*R_get_missing_data.sps.
DATA LIST LIST (',') /numVar (f) stringVar (a4).
BEGIN DATA
1,a
,b
3,,
9,d
END DATA.
MISSING VALUES numVar (9) stringVar (' ').
BEGIN PROGRAM R.
data <- spssdata.GetDataFromSPSS()
cat("Case data with missing values:\n")
print(data)
END PROGRAM.
```

#### *Result*

```
Case data with missing values:

  numVar stringVar
1      1         a
2    NaN         b
3      3      <NA>
4    NaN         d
```

*Note*: You can specify that missing values of numeric variables be converted to the R *NA* value, with the *missingValueToNA* argument, as in:

```
data<-spssdata.GetDataFromSPSS(missingValueToNA=TRUE)
```

You can specify that user-missing values be treated as valid data by setting the optional argument *keepUserMissing* to *TRUE*, as shown in the following reworking of the previous example.

```
DATA LIST LIST (',') /numVar (f) stringVar (a4).
BEGIN DATA
1,a
,b
3,,
9,d
END DATA.
MISSING VALUES numVar (9) stringVar (' ').
BEGIN PROGRAM R.
data <- spssdata.GetDataFromSPSS(keepUserMissing=TRUE)
cat("Case data with user-missing values treated as valid:\n")
print(data)
END PROGRAM.
```

***Result***

```
Case data with user-missing values treated as valid:

  numVar stringVar
1      1         a
2    NaN         b
3      3
4      9         d
```

***Example: Handling Data with Splits***

When reading from SPSS Statistics datasets with split groups, use the `GetSplitDataFromSPSS` function to retrieve each split separately, as shown in this example.

```
*R_get_split_groups.sps.
DATA LIST FREE /salary (F6) jobcat (F2).
BEGIN DATA
21450 1
45000 1
30000 2
30750 2
103750 3
72500 3
57000 3
END DATA.

SORT CASES BY jobcat.
SPLIT FILE BY jobcat.
BEGIN PROGRAM R.
varnames <- spssdata.GetSplitVariableNames()
if(length(varnames) > 0)
{
   while (!spssdata.IsLastSplit()){
      data <- spssdata.GetSplitDataFromSPSS()
      cat("\n\nSplit variable values:")
      for (name in varnames) cat("\n",name,":",
                                 as.character(data[1,name]))
      cat("\nCases in Split: ",length(data[,1]))
   }
   spssdata.CloseDataConnection()
}
END PROGRAM.
```

#### Result

```
Split variable values:
 jobcat : 1
Cases in Split:  2

Split variable values:
 jobcat : 2
Cases in Split:  2

Split variable values:
 jobcat : 3
Cases in Split:  3
```

- The `GetSplitVariableNames` function returns the names of the split variables, if any, from the active dataset.
- The `GetSplitDataFromSPSS` function retrieves the case data for the next split group from the active dataset, and returns it as an R data frame.
- The `IsLastSplit` function returns *TRUE* if the current split group is the last one in the active dataset.
- The `CloseDataConnection` function should be called when the necessary split groups have been read. In particular, `GetSplitDataFromSPSS` implicitly starts a data connection for reading from split files and this data connection must be closed with `CloseDataConnection`.

## Writing Results to a New IBM SPSS Statistics Dataset

The IBM® SPSS® Statistics - Integration Plug-In for R provides the ability to write results from R to a new IBM® SPSS® Statistics dataset. The steps to create a new dataset are:

▶ Create the dataset's dictionary using the `SetDictionaryToSPSS` function. The function requires a data frame representation of the dictionary as created by the `GetDictionaryFromSPSS` function or the `CreateSPSSDictionary` function.

▶ Populate the case data using the `SetDataToSPSS` function.

*Note*: When setting values for a SPSS Statistics variable with a date or datetime format, specify the values as R POSIXt objects, which will then be correctly converted to the values appropriate for SPSS Statistics. Also note that SPSS Statistics variables with a time format are stored as the number of seconds from midnight.

#### Example

This example shows how to create a new dataset that is a copy of the active dataset with the addition of a single new variable.

```
dict <- spssdictionary.GetDictionaryFromSPSS()
casedata <- spssdata.GetDataFromSPSS()
varSpec <- c("meansal","Mean Salary",0,"F8","scale")
dict <- data.frame(dict,varSpec)
spssdictionary.SetDictionaryToSPSS("results",dict)
casedata <- data.frame(casedata,mean(casedata$salary))
spssdata.SetDataToSPSS("results",casedata)
spssdictionary.EndDataStep()
```

- The `GetDictionaryFromSPSS` function returns an R data frame representation of the active dataset's dictionary. The `GetDataFromSPSS` function returns an R data frame representation of the case data from the active dataset.
- New variables are specified as an R vector—in this example, *varSpec*—whose components are the properties of the variable in the following required order: variable name, variable label, variable type, variable format, measurement level. For more information, see the topic spssdictionary.CreateSPSSDictionary Function in Chapter 2 on p. 52.
- The code `data.frame(dict,varSpec)` creates a data frame representation of the new dictionary, consisting of the original dictionary and the new variable.

  You can also use `CreateSPSSDictionary` to create a dictionary from scratch without building onto one retrieved with `GetDictionaryFromSPSS`. In that case, you create R vectors specifying each of your variables, as done with *varSpec*, and include those vectors in the call to `CreateSPSSDictionary`. The order of the arguments to `CreateSPSSDictionary` is the order of the associated variables in the new dataset.
- The `SetDictionaryToSPSS` function creates a new dataset named *results* from the data frame representation of the dictionary.
- The code `data.frame(casedata,mean(casedata$salary))` creates a new data frame consisting of the data retrieved from the active dataset and the data for the new variable. In this example, the new variable is the mean of the variable *salary* from the active dataset. You can build data frames from existing data frames, as done here, or from vectors representing each of the columns. For example, `data.frame(var1,var2,var2)` creates a data frame whose columns are specified by the vectors *var1*, *var2*, and *var3*. The vectors must be of equal length and in the same order as the associated variables in the new dataset.
- The `SetDataToSPSS` function populates the case data of the new dataset. Its arguments are the name of the dataset to populate and a data frame representation of the case data.
- The `EndDataStep` function should be called after completing the steps for creating the new dataset.

*Note:* Missing values, value labels, custom variable attributes, datafile attributes, and multiple response sets are set with the spssdictionary.SetUserMissing, spssdictionary.SetValueLabel, spssdictionary.SetVariableAttributes, spssdictionary.SetDataFileAttributes, and spssdictionary.SetMultiResponseSet functions. When used, these functions must be called after the `SetDictionaryToSPSS` function and prior to the `EndDataStep` function.

### Writing Categorical Variables Back to IBM SPSS Statistics

When reading categorical variables from SPSS Statistics with `factorMode="labels"` and writing the associated R factors to a new SPSS Statistics dataset, special handling is required because labeled factors in R do not preserve the original values. In this example, we read data containing categorical variables from SPSS Statistics and create a new dataset containing the original data with the addition of a single new variable.

```
*R_read_write_catvars.sps.
DATA LIST FREE /id (F4) gender (A1) training (F1) salary (DOLLAR).
VARIABLE LABELS id 'Employee ID'
/training 'Training Level'.
VARIABLE LEVEL id (SCALE)
/gender (NOMINAL)
/training (ORDINAL)
/salary (SCALE).
VALUE LABELS training 1 'Beginning' 2 'Intermediate' 3 'Advanced'
/gender 'm' 'Male' 'f' 'Female'.
BEGIN DATA
18 m 3 57000
37 f 2 30750
10 f 1 22000
22 m 2 31950
END DATA.

BEGIN PROGRAM  R.
dict <- spssdictionary.GetDictionaryFromSPSS()
casedata <- spssdata.GetDataFromSPSS(factorMode="labels")
catdict <- spssdictionary.GetCategoricalDictionaryFromSPSS()
varSpec <- c("meansal","Mean Salary",0,"DOLLAR8","scale")
dict<-data.frame(dict,varSpec)
casedata<-data.frame(casedata,mean(casedata$salary))
spssdictionary.SetDictionaryToSPSS("results",dict,categoryDictionary=catdict)
spssdata.SetDataToSPSS("results",casedata,categoryDictionary=catdict)
spssdictionary.EndDataStep()
END PROGRAM.
```

- The `GetCategoricalDictionaryFromSPSS` function returns a structure (referred to as a **category dictionary**) containing the values and value labels of the categorical variables from the active dataset.
- The category dictionary stored in *catdict* is used when creating the new dataset with the `SetDictionaryToSPSS` function and when writing the data to the new dataset with the `SetDataToSPSS` function. The value labels of the categorical variables are automatically added to the new dataset and the case values of those variables (in the new dataset) are the values from the original dataset.

If you rename categorical variables when writing them back to SPSS Statistics, you must use the EditCategoricalDictionary function to change the name in the associated category dictionary.

### Saving New Datasets

To save a new dataset created from within an R program block, you include command syntax—such as `SAVE` or `SAVE TRANSLATE`—following the program block that created the dataset. Note, however, that a dataset created in an R program block is NOT set as the active dataset. To make a new dataset the active one, use the `spssdictionary.SetActive` function from within the program block or the `DATASET ACTIVATE` command outside of the program block. If you choose to use the `SetActive` function, it must be called prior to calling `spssdictionary.EndDataStep`.

## Creating Pivot Table Output

The IBM® SPSS® Statistics - Integration Plug-In for R provides the ability to render tabular output from R as a pivot table that can be displayed in the IBM® SPSS® Statistics Viewer or written to an external file using the SPSS Statistics Output Management System.

Typically, the output from an R analysis—such as a generalized linear model—is an object whose attributes contain the results of the analysis. You can extract the results of interest and render them as pivot tables in SPSS Statistics using the `spsspivottable.Display` function.

#### *Example*

In this example, we read the case data from the active dataset, create a generalized linear model, and write summary results of the model coefficients back to the SPSS Statistics Viewer as a pivot table.

```
casedata <- spssdata.GetDataFromSPSS(variables=c("car","income","ed","marital"))
model <- glm(car~income+ed+marital,data=casedata)
res <- summary(model)
spsspivottable.Display(res$coefficients,
                       title="Model Coefficients"),
                       format=formatSpec.GeneralStat)
```

#### *Result*

Figure 1-1
*Model Coefficients*

| | Estimate | Std. Error | t value | Pr(>\|t\|) |
|---|---|---|---|---|
| (Intercept) | 13.698 | .449 | 30.477 | .000 |
| income | .220 | .002 | 103.083 | .000 |
| ed | .475 | .140 | 3.390 | .001 |
| marital | -.162 | .335 | -.484 | .628 |

- The R variable *model* contains the results of the generalized linear model analysis.
- The R `summary` function takes the results of the glm analysis and produces an R object with a number of attributes that summarize the model. In particular, the *coefficients* attribute contains a table of the model coefficients and associated statistics.

  *Note:* You can obtain a list of the attributes available for an object using `attributes(object)`.
- The `spsspivottable.Display` function creates the pivot table. The first and only required argument is the data to be displayed as a pivot table. This can be a data frame, matrix, table, or any R object that can be converted to a data frame. In the present example, the *coefficients* attribute of the `summary` object is a matrix.
- The *format* argument specifies the format to be used for displaying numeric values, including cell values, row labels, and column labels. The argument is of the form `formatSpec.format`, as in `formatSpec.GeneralStat`. A list of available formats as well as a brief guide to choosing a format is provided in the topic on the `spsspivottable.Display` function on p. 67.

Optional arguments to the `spsspivottable.Display` function allow you to customize the pivot table.

By default, the name that appears in the outline pane of the Viewer associated with the pivot table is *R*. You can customize the name and nest multiple pivot tables under a common heading by wrapping the pivot table generation in a `StartProcedure-EndProcedure` block. For more information, see the topic spsspkg.StartProcedure Function in Chapter 2 on p. 72.

The `spsspivottable.Display` is limited to pivot tables with one row dimension and one column dimension. To create more complex pivot tables, use the BasePivotTable class.

# Displaying Graphical Output from R

By default, graphical output from R—for instance, from the R `plot` function—is displayed in the IBM® SPSS® Statistics Viewer. You can display a specified R graphics file, from disk, in the SPSS Statistics Viewer using the spssRGraphics.Submit function, and you can turn display of R graphics on or off using the spssRGraphics.SetOutput function. R graphics displayed in the SPSS Statistics Viewer cannot be edited and do not use the graphics preference settings in SPSS Statistics.

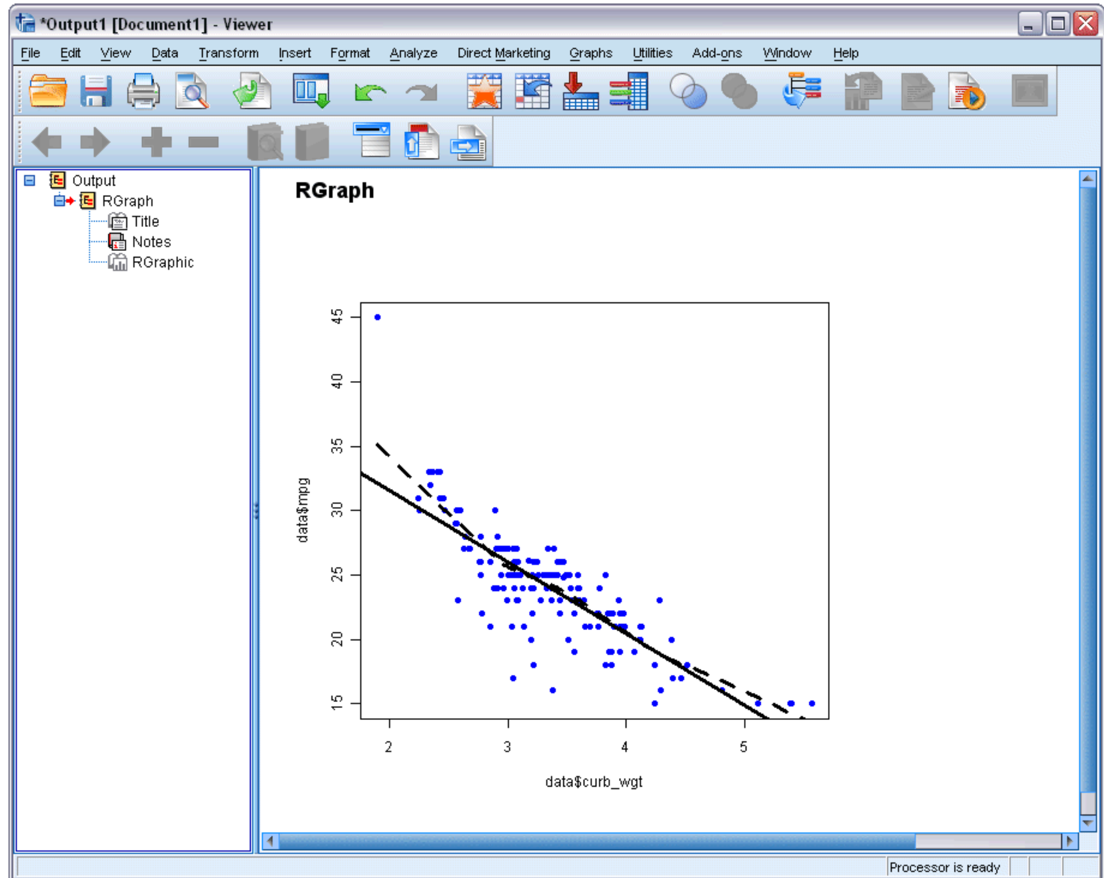
### Example

This example makes use of the default behavior for rendering graphical output from R in the Viewer. It produces a scatterplot, along with fit lines computed from a linear model and separately from a smoothing algorithm.

```
COMPUTE filter_$=(nvalid(mpg, curb_wgt) = 2).
FILTER BY filter_$.
BEGIN PROGRAM R.
data <- spssdata.GetDataFromSPSS()
plot(data$curb_wgt, data$mpg, col="blue", pch=19)
abline(lm(data$mpg ~ data$curb_wgt), lwd=3 )
lines(lowess(data$mpg ~ data$curb_wgt), lwd=3, lty=2)
END PROGRAM.
```

#### Result

Figure 1-2
*R graphic displayed in the Viewer*




#### Note on Generating Multiple Graphics

When invoking a graphics command that generates multiple graphics, you will need to add the parameter `ask=FALSE`, as in: `plot(result,ask=FALSE)`.

## Retrieving Output from Syntax Commands

Functionality provided with the IBM® SPSS® Statistics - Integration Plug-In for R allows you to access output from IBM® SPSS® Statistics syntax commands in a programmatic fashion. To retrieve command output, you first route it via the Output Management System (OMS) to an area in memory referred to as the **XML workspace** where it is stored as an XPath DOM that conforms to the Output XML Schema (*xml.spss.com/spss/oms*). Output is retrieved from this workspace with functions that employ XPath expressions.

Constructing the correct XPath expression (SPSS Statistics currently supports XPath 1.0) requires an understanding of the Output XML schema. Documentation for the output schema is available from the Help system.

***Example***

In this example, we'll use output from the `DESCRIPTIVES` command to determine the percentage of valid cases for a specified variable.

```
*Route output to the XML workspace.
OMS SELECT TABLES
 /IF COMMANDS=['Descriptives'] SUBTYPES=['Descriptive Statistics']
 /DESTINATION FORMAT=OXML XMLWORKSPACE='desc_table'
 /TAG='desc_out'.
DESCRIPTIVES VARIABLES=mpg.
OMSEND TAG='desc_out'.
*Get output from the XML workspace using XPath.
BEGIN PROGRAM R.
handle <- "desc_table"
context <- "/outputTree"
xpath <- paste("//pivotTable[@subType='Descriptive Statistics']",
               "/dimension[@axis='row']",
               "/category[@varName='mpg']",
               "/dimension[@axis='column']",
               "/category[@text='N']",
               "/cell/@number")
res <- spssxmlworkspace.EvaluateXPath(handle,context,xpath)
ncases <- spssdata.GetCaseCount()
cat("Percentage of valid cases for variable mpg: ",
     round(100*as.integer(res)/ncases),"%")
spssxmlworkspace.DeleteXmlWorkspaceObject(handle)
END PROGRAM.
```

- The `OMS` command is used to direct output from a syntax command to the XML workspace. The `XMLWORKSPACE` keyword on the `DESTINATION` subcommand, along with `FORMAT=OXML`, specifies the XML workspace as the output destination. It is a good practice to use the `TAG` subcommand, as done here, so as not to interfere with any other OMS requests that may be operating. The identifiers used for the `COMMANDS` and `SUBTYPES` keywords on the `IF` subcommand can be found in the OMS Identifiers dialog box, available from the Utilities menu in SPSS Statistics.
- The `XMLWORKSPACE` keyword is used to associate a name with this XPath DOM in the workspace. In the current example, output from the `DESCRIPTIVES` command will be identified with the name *desc_table*. You can have many XPath DOM's in the XML workspace, each with its own unique name.
- The `OMSEND` command terminates active `OMS` commands, causing the output to be written to the specified destination—in this case, the XML workspace.
- You retrieve values from the XML workspace with the `spssxmlworkspace.EvaluateXPath` function. The function takes an explicit XPath expression, evaluates it against a specified XPath DOM in the XML workspace, and returns the result as a vector of character strings.
- The first argument to the `EvaluateXPath` function specifies the XPath DOM to which an XPath expression will be applied. This argument is referred to as the handle name for the XPath DOM and is simply the name given on the `XMLWORKSPACE` keyword on the associated `OMS` command. In this case the handle name is *desc_table*.
- The second argument to `EvaluateXPath` defines the XPath context for the expression and should be set to `"/outputTree"` for items routed to the XML workspace by the `OMS` command.

- The third argument to `EvaluateXPath` specifies the remainder of the XPath expression (the context is the first part) and must be quoted. Since XPath expressions almost always contain quoted strings, you'll need to use a different quote type from that used to enclose the expression. For users familiar with XSLT for OXML and accustomed to including a namespace prefix, note that XPath expressions for the `EvaluateXPath` function should not contain the `oms:` namespace prefix.
- The XPath expression in this example is specified by the variable *xpath*. It is not the minimal expression needed to select the value of interest but is used for illustration purposes and serves to highlight the structure of the XML output.

  `//pivotTable[@subType='Descriptive Statistics']` selects the Descriptives Statistics table.

  `/dimension[@axis='row']/category[@varName='mpg']` selects the row for the variable *mpg*.

  `/dimension[@axis='column']/category[@text='N']` selects the column labeled *N* (the number of valid cases), thus specifying a single cell in the pivot table.

  `/cell/@text` selects the textual representation of the cell contents.
- When you have finished with a particular output item, it is a good idea to delete it from the XML workspace. This is done with the `DeleteXmlWorkspaceObject` function, whose single argument is the name of the handle associated with the item.

If you're familiar with XPath, you might want to convince yourself that the number of valid cases for *mpg* can also be selected with the following simpler XPath expression:

```
//category[@varName='mpg']//category[@text='N']/cell/@text
```

*Note*: To the extent possible, construct your XPath expressions using language-independent attributes, such as the variable name rather than the variable label. That will help reduce the translation effort if you need to deploy your code in multiple languages. Also consider factoring out language-dependent identifiers, such as the name of a statistic, into constants. You can obtain the current language used for pivot table output with the `spsspkg.GetOutputLanguage` function.

You may also consider using text_eng attributes in place of text attributes in XPath expressions. text_eng attributes are English versions of text attributes and have the same value regardless of the output language. The `OATTRS` subcommand of the `SET` command specifies whether text_eng attributes are included in OXML output.

## Localizing Output from R

You can localize output, such as messages and pivot table strings, from extension commands implemented in R as well as from explicit `BEGIN PROGRAM R` blocks. The localization process consists of the following steps:

- ▶ Modifying the R implementation code to mark translatable strings and specify the location of translation files
- ▶ Extracting translatable text from the implementation code

- Preparing a translated file of strings for each target language
- Installing the translation files

#### *Notes*

- The language for extension command and program block output will be automatically synchronized with the IBM® SPSS® Statistics output language (`OLANG`). However, users of your extension command may need to set their SPSS Statistics locale to match the SPSS Statistics output language in order to properly display extended characters, even when working in Unicode mode. For example, if the output language is Japanese then they may need to set their SPSS Statistics locale to Japanese, as in `SET LOCALE='japanese'`.
- Translation of dialog boxes built with the Custom Dialog Builder is a separate process, but translators should ensure that the dialog and any associated extension command translations are consistent.

#### *Additional Resources*

- String translation in R utilizes implementations of functionality in the GNU gettext facility. Complete documentation on the GNU gettext facility is available from *http://www.gnu.org/software/gettext/*.
- Examples of extension commands implemented in R with localized output are included with IBM® SPSS® Statistics - Essentials for R. The R source code files for these examples are located in the *extensions* directory under the SPSS Statistics installation directory and have a file extension of *.R*. If you have specified alternate locations for extension commands with the *SPSS_EXTENSIONS_PATH* environment variable then the R source code files will be located in the first writable location in that variable instead of in the *extensions* directory.

## Modifying the R code

To enable the translation mechanism, you must modify the R code that generates your output—for example, the R source code that implements an extension command. First, however, 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.

To enable the translation mechanism, you must include a call to the R `bindtextdomain` function to associate a name—called the domain name—with a set of translation files. The function takes two arguments: the domain name, and the location where the associated translation files reside. If you are creating translations for an extension command implemented in R, then it is recommended

to use the name of the extension command as the domain name. For multi-word extension command names, replace the spaces with underscores. For example:

```
bindtextdomain(domain="MYORG_MYSTAT",dirname=paste(spsspkg.GetStatisticsPath(),
               "extensions/MYORG_MYSTAT/lang",sep=""))
```

- The domain name in this example is *"MYORG_MYSTAT"*, and it will represent translations for an extension command named `MYORG MYSTAT`.
- The *dirname* argument specifies the path to the directory containing the translation files. In this example, translation files are located in the *extensions/MYORG_MYSTAT/lang* directory under the location where IBM® SPSS® Statistics is installed. For more information, see the topic Installing the mo files on p. 21.

In addition to the `bindtextdomain` function, you must enclose each translatable string in a call to the R `gettext`, `ngettext`, or `gettextf` function. For example:

```
gettext("ERROR:",domain="MYORG_MYSTAT")
```

- The arguments to `gettext` are the untranslated string—in this case, `"ERROR:"`—and the domain name specified in the `bindtextdomain` function. The function will fetch the translation, if available, when the statement containing the string is executed.

Calls to the `spsspkg.StartProcedure` function should use the form `spsspkg.StartProcedure(pName,omsId)` where *pName* is the translatable name associated with output from the procedure and *omsId* is the language invariant `OMS` command identifier associated with the procedure. For example:

```
spsspkg.StartProcedure(gettext("Demo",domain="MYORG_MYSTAT"),"demoId")
```

## Extracting translatable text

The R 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 R `gettext`, `ngettext`, and `gettextf` functions use compiled versions of these files.

You can use the R `xgettext2pot` function to extract strings marked as translatable (i.e., strings wrapped in the `gettext`, `ngettext`, or `gettextf` functions) and save them to a *.pot* (po template) file. The *.pot* file should have the same name as the *domain* specified in the `bindtextdomain` function. If you are localizing output from an extension command implemented in R, then the *.pot* file should have the same name as the extension command, in upper case, and with any spaces replaced with underscores—for example, *MYORG_MYSTAT.pot* for an extension command named `MYORG MYSTAT`.

In the *.pot* file:

- 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.

## 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.
- `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 the `msgfmt` program (included with the freely available GNU gettext utilities package), 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/<domain name>.mo*

- *<domain name>* is the name of the domain specified in the call to the bindtextdomain function. 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 Language Identifiers table.

For example, if the translations are for an extension command 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 *<domain name>* directory under the directory where the R source code file is installed.

For example, if an extension command is named *MYORG MYSTAT* and the associated R source code file (*MYORG_MYSTAT.R*) is located in the *extensions* directory (under the location where 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 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 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 |

Chapter

2

# R Integration Package for IBM SPSS Statistics: Functions and Classes

The R Integration Package for IBM® SPSS® Statistics contains functions that facilitate the process of using R programming features with command syntax, including functions that:

***Get information about data files in the current IBM SPSS Statistics session***

- spssdata.GetCaseCount
- spssdata.GetDataSetList
- spssdata.GetFileHandles
- spssdata.GetSplitVariableNames
- spssdictionary.GetDataFileAttributes
- spssdictionary.GetMultiResponseSet
- spssdictionary.GetUserMissingValues
- spssdictionary.GetValueLabels
- spssdictionary.GetVariableAttributes
- spssdictionary.GetVariableCount
- spssdictionary.GetVariableFormat
- spssdictionary.GetVariableLabel
- spssdictionary.GetVariableMeasurementLevel
- spssdictionary.GetVariableName
- spssdictionary.GetVariableType
- spssdictionary.GetWeightVariable

***Get data from the active dataset and create new datasets***

- spssdata.GetDataFromSPSS
- spssdata.GetSplitDataFromSPSS
- spssdata.SetDataToSPSS
- spssdictionary.CreateSPSSDictionary
- spssdictionary.SetDictionaryToSPSS
- spssdictionary.SetUserMissing
- spssdictionary.SetValueLabel
- spssdictionary.SetVariableAttributes
- spssdictionary.SetMultiResponseSet
- spssdictionary.SetDataFileAttributes

© Copyright IBM Corporation 1989, 2011.

#### Create custom pivot tables and text blocks

- spss.BasePivotTable
- spss.TextBlock
- spsspivottable.Display

#### Get output results

- spssxmlworkspace.EvaluateXPath

#### Control display of R graphics and output

- spssRGraphics.Submit
- spssRGraphics.SetOutput
- spsspkg.SetOutput

#### Get version information

- spsspkg.GetSPSSPlugInVersion
- spsspkg.GetSPSSVersion
- spsspkg.Version

#### Locale and output language settings

- spsspkg.GetOutputLanguage
- spsspkg.GetSPSSLocale
- spsspkg.SetOutputLanguage

#### Utility functions for extension commands

- spsspkg.processcmd
- spsspkg.Syntax
- spsspkg.Template

## 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 SPSS Statistics Output Management System.*

*Note*: If you only need a pivot table with a single column dimension and a single row dimension, you may want to use the much simpler spsspivottable.Display function.

- 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.
- 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. Split file processing refers to whether results from different split groups are displayed in separate tables or in the same table but grouped by split, and is controlled by the `SPLIT FILE` command. By default, split processing is enabled. To disable split processing for pivot tables, specify `isSplit=FALSE`.

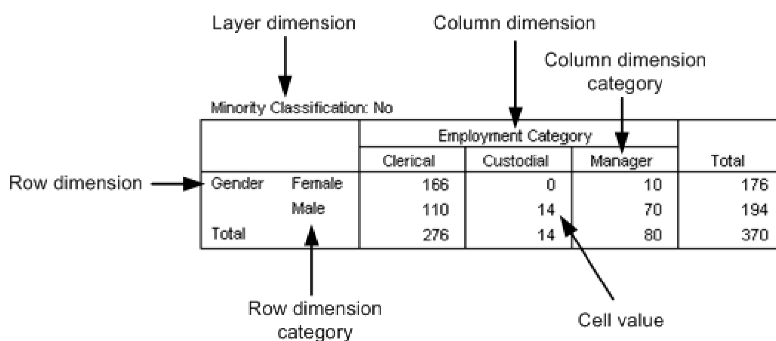
  When retrieving data with `spssdata.GetSplitDataFromSPSS`, simply repopulate the pivot table cells with the results for each new split group. The results from each split group are accumulated and the subsequent table(s) are displayed when `spssdata.CloseDataConnection` is called.
- The optional argument *caption* is a string that specifies a table caption.

*Note*: An instance of the `BasePivotTable` class can only be used within an `spsspkg.StartProcedure-spsspkg.EndProcedure` block. See Setting Cell Values for an example.

Figure 2-1 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 2-1
*Pivot table structure*




| | | Employment Category | | | Total |
|---|---|---|---|---|---|
| | | Clerical | Custodial | Manager | |
| Gender | Female | 166 | 0 | 10 | 176 |
| | Male | 110 | 14 | 70 | 194 |
| Total | | 276 | 14 | 80 | 370 |

## General Approach to Creating Pivot Tables

The `BasePivotTable` class provides the means for creating pivot tables that cannot be created with the spsspivottable.Display function. The basic steps for creating a pivot table are:

- Create an instance of the `BasePivotTable` class.
- Add dimensions.
- Define categories.
- 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.

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

- The first argument to `Append` is a reference to the `BasePivotTable` object—in this example, the R variable *table*.
- The second argument to the `Append` method specifies the type of dimension, using one member from a set of built-in object properties: `Dimension.Place.row` for a row dimension, `Dimension.Place.column` for a column dimension, and `Dimension.Place.layer` for a layer dimension.
- The third argument to `Append` is a string that specifies the name used to label this dimension in the displayed table.
- A reference to each newly created dimension object is stored in a variable. For instance, the variable *rowdim1* holds a reference to the object for the row dimension named *rowdim-1*.

Figure 2-2
*Resulting table structure*

| | | coldim |
|---|---|---|
| rowdim-1 | rowdim-2 | |
| | | |

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=BasePivotTable.Append(table,Dimension.Place.row,"rowdim-1")
rowdim2=BasePivotTable.Append(table,Dimension.Place.row,"rowdim-2")
rowdim3=BasePivotTable.Insert(table,2,Dimension.Place.row,"rowdim-3")
coldim=BasePivotTable.Append(table,Dimension.Place.column,"coldim")
```

- The first argument to `Insert` is a reference to the `BasePivotTable` object—in this example, the R variable *table*.
- The second argument to the `Insert` method 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. In the current example, *rowdim-3* is inserted at position 2 and *rowdim-1* is moved from position 2 to position 3.
- The third argument to `Insert` specifies the type of dimension, using one member from a set of built-in object properties: `Dimension.Place.row` for a row dimension, `Dimension.Place.column` for a column dimension, and `Dimension.Place.layer` for a layer dimension.
- The fourth argument to `Insert` is a string that specifies the name used to label this dimension in the displayed table.
- A reference to each newly created dimension object is stored in a variable. For instance, the variable *rowdim3* holds a reference to the object for the row dimension named *rowdim-3*.

Figure 2-3
*Resulting table structure*

| | | | coldim |
|---|---|---|---|
| rowdim-1 | rowdim-3 | rowdim-2 | |
| | | | |

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

### *Step 2: Defining Categories*

You define categories for each dimension using the SetCategories method.

#### *Example*

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

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

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

BasePivotTable.SetCategories(table,rowdim1,list(cat1,cat2))
```

```
BasePivotTable.SetCategories(table,rowdim2,list(cat3,cat4))
BasePivotTable.SetCategories(table,coldim,list(cat5,cat6,cat7))
```

- 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 a reference to the `BasePivotTable` object—in this example, the R variable *table*.
- The second argument to `SetCategories` is an object reference to the dimension for which the categories are being defined.
- The third argument to `SetCategories` is a single category or a list 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.

Figure 2-4
*Resulting table structure*

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

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

## Step 3: Setting Cell Values

You can set cell values by row or by column using the SetCellsByRow or SetCellsByColumn method respectively. The `SetCellsByRow` method is limited to pivot tables with one column dimension and the `SetCellsByColumn` method is limited to pivot tables with one row dimension. To set cells for pivot tables with multiple row and column dimensions, use the SetCellValue method.

### Example: Setting Cell Values by Row

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

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

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

BasePivotTable.SetCategories(table,rowdim,list(row_cat1,row_cat2))
```

```
BasePivotTable.SetCategories(table,coldim,list(col_cat1,col_cat2))

BasePivotTable.SetCellsByRow(table,row_cat1,list(spss.CellText.Number(11),
                             spss.CellText.Number(12)))
BasePivotTable.SetCellsByRow(table,row_cat2,list(spss.CellText.Number(21),
                             spss.CellText.Number(22)))
spsspkg.EndProcedure()
```

- This example also shows how to wrap the code for creating a pivot table in an `spsspkg.StartProcedure-spsspkg.EndProcedure` block. When creating a pivot table with the `BasePivotTable` class, you must always wrap the associated code in an `spsspkg.StartProcedure-spsspkg.EndProcedure` block. You can copy the code for this example to a syntax editor window, enclose it in a `BEGIN PROGRAM R-END PROGRAM` block and run it to produce a pivot table.
- The `SetCellsByRow` method is called for each of the two categories in the row dimension.
- The first argument to `SetCellsByRow` is a reference to the `BasePivotTable` object—in this example, the R variable *table*.
- The second 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 third argument to the `SetCellsByRow` method is a list 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 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: `spss.CellText.Number(22,formatSpec.Correlation)`. For more information, see the topic CellText.Number Class on p. 40.

### *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

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

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

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

BasePivotTable.SetCategories(table,rowdim,list(row_cat1,row_cat2))
BasePivotTable.SetCategories(table,coldim,list(col_cat1,col_cat2))

BasePivotTable.SetCellValue(table,list(row_cat1,col_cat1),spss.CellText.Number(11))
cellValue = CellText.toNumber(BasePivotTable.GetCellValue(table,list(row_cat1,col_cat1)))
BasePivotTable.SetCellValue(table,list(row_cat2,col_cat2),spss.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

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

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

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

BasePivotTable.SetCategories(table,rowdim,list(row_cat1,row_cat2))
BasePivotTable.SetCategories(table,coldim,list(col_cat1,col_cat2))

BasePivotTable.SetCellValue(table,list(row_cat1,col_cat1),spss.CellText.String("abc"))
cellValue = CellText.toString(BasePivotTable.GetCellValue(table,list(row_cat1,col_cat1)))
BasePivotTable.SetCellValue(table,list(row_cat2,col_cat2),
                            spss.CellText.String(paste(cellValue,"d",sep="")))
```

- 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.

## 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 much simpler spsspivottable.Display function.

### Append Method

**.Append(object,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 *object* is a reference to the associated `BasePivotTable` object. The argument *place* specifies the type of dimension: `Dimension.Place.row` for a row dimension, `Dimension.Place.column` for a column dimension, and `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=BasePivotTable.Append(table,Dimension.Place.column,"coldim")
rowdim1=BasePivotTable.Append(table,Dimension.Place.row,"rowdim-1")
rowdim2=BasePivotTable.Append(table,Dimension.Place.row,"rowdim-2")
```

Figure 2-5
*Resulting table structure*

| | | coldim |
|---|---|---|
| rowdim-1 | rowdim-2 | |
| | | |

Examples of using the `Append` method are most easily understood in the context of going through the steps to create a pivot table. For more information, see the topic General Approach to Creating Pivot Tables on p. 26.

### Caption Method

**.Caption(object,caption).** *Adds a caption to the pivot table.* The argument *object* is a reference to the associated `BasePivotTable` object. The argument *caption* is a string specifying the caption.

#### Example

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

### CategoryFootnotes Method

**.CategoryFootnotes(object,dimPlace,dimName,category,footnotes).** *Used to add a footnote to a specified category.*

- The argument *object* is a reference to the associated `BasePivotTable` object.
- The argument *dimPlace* specifies the dimension type associated with the category, using one member from a set of built-in object properties: `Dimension.Place.row` for a row dimension, `Dimension.Place.column` for a column dimension, and `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 *footnotes* is a string specifying the footnote.

#### Example

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

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

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

BasePivotTable.SetCategories(table,rowdim,list(row_cat1,row_cat2))
BasePivotTable.SetCategories(table,coldim,list(col_cat1,col_cat2))

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

### DimensionFootnotes Method

**.DimensionFootnotes(object,dimPlace,dimName,footnotes).** *Used to add a footnote to a dimension.*

- The argument *object* is a reference to the associated `BasePivotTable` object.
- The argument *dimPlace* specifies the type of dimension, using one member from a set of built-in object properties: `Dimension.Place.row` for a row dimension, `Dimension.Place.column` for a column dimension, and `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 *footnotes* is a string specifying the footnote.

#### *Example*

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

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

### *Footnotes Method*

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

#### *Example*

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

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

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

BasePivotTable.SetCellValue(table,list(spss.CellText.String("row1"),
                                       spss.CellText.String("column1")),
                                       spss.CellText.String("cell value"))
BasePivotTable.Footnotes(table,list(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 `list(spss.CellText.String("row1"),spss.CellText.String("column1"))` specifies a category of *rowdim* and the second element specifies a category of *coldim*.

### *GetCellValue Method*

**.GetCellValue(object,categories).** *Gets the value of the specified cell.* The argument *object* is a reference to the associated `BasePivotTable` object. The argument *categories* is the list of the category values that specifies the cell—one value for each of the dimensions in the pivot table. Category values must be specified as CellText objects (one of `CellText.Number`, `CellText.String`, `CellText.VarName`, or `CellText.VarValue`).

- In the list that specifies the category values, the first element corresponds to the first appended dimension, the second element to the second appended dimension, and so on. For example, if the pivot table has two row dimensions and one column dimension and the row dimensions are appended before the column dimension, then:

  ```
  list(rowdim1_cat1,rowdim2_cat3,coldim_cat1)
  ```

specifies the cell whose category values are `rowdim1_cat1` in the first appended row dimension, `rowdim2_cat3` in the second appended row dimension, and `coldim_cat1` in the column dimension.

For an example of using the `GetCellValue` method, see the CellText.toNumber method.

### GetDefaultFormatSpec Method

**.GetDefaultFormatSpec(object).** *Returns the default format for CellText.Number objects.* The argument *object* is a reference to the associated `BasePivotTable` object. The function returns a single value or a vector with two elements depending on the type of format. When a single value is returned, it is the integer code associated with the format. Codes and associated formats are listed in Table 2-2 on p. 69. For formats with codes `5` (`Mean`), `12` (`Variable`), `13` (`StdDev`), `14` (`Difference`), and `15` (`Sum`), the result is a 2-element vector whose first element is the integer code and whose second element is the index of the variable in the active dataset used to determine details of the resulting format. 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")
cat("Default format: ", BasePivotTable.GetDefaultFormatSpec(table))
```

### HideTitle Method

**.HideTitle(object).** *Used to hide the title of a pivot table.* By default, the title is shown. The argument *object* is a reference to the associated `BasePivotTable` object.

#### Example

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

### Insert Method

**.Insert(object,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 *object* is a reference to the associated `BasePivotTable` object. 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: `Dimension.Place.row` for a row dimension, `Dimension.Place.column` for a column dimension, and `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=BasePivotTable.Append(table,Dimension.Place.row,"rowdim-1")
rowdim2=BasePivotTable.Append(table,Dimension.Place.row,"rowdim-2")
rowdim3=BasePivotTable.Insert(table,2,Dimension.Place.row,"rowdim-3")
coldim=BasePivotTable.Append(table,Dimension.Place.column,"coldim")
```

Figure 2-6
*Resulting table structure*

| | | | coldim |
|---|---|---|---|
| rowdim-1 | rowdim-3 | rowdim-2 | |
| | | | |

Examples of using the Insert method are most easily understood in the context of going through the steps to create a pivot table. For more information, see the topic General Approach to Creating Pivot Tables on p. 26.

### SetCategories Method

**.SetCategories(object,dim,categories).** *Sets categories for the specified dimension.* The argument *object* is a reference to the associated BasePivotTable object. 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 list of unique values, each of which is a CellText object (one of CellText.Number, CellText.String, CellText.VarName, or CellText.VarValue).

#### Example

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

rowdim=BasePivotTable.Append(table,Dimension.Place.row,"rowdim")
```

```
coldim=BasePivotTable.Append(table,Dimension.Place.column,"coldim")

BasePivotTable.SetCategories(table,rowdim,list(spss.CellText.String("row1"),
                             spss.CellText.String("row2")))
BasePivotTable.SetCategories(table,coldim,list(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. For more information, see the topic General Approach to Creating Pivot Tables on p. 26.

### SetCellsByColumn Method

**.SetCellsByColumn(object,collabels,cells).** *Sets cell values for the column specified by a set of categories, one for each column dimension.* The argument *object* is a reference to the associated `BasePivotTable` object. The argument *collabels* specifies the set of categories that defines the column—a single value or a list. The argument *cells* is a 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

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

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

BasePivotTable.SetCategories(table,coldim1,list(cat1,cat2))
BasePivotTable.SetCategories(table,coldim2,list(cat3,cat4))
BasePivotTable.SetCategories(table,rowdim,list(cat5,cat6))

BasePivotTable.SetCellsByColumn(table,list(cat1,cat3),
                    list(spss.CellText.Number(11),
                     spss.CellText.Number(21)))
BasePivotTable.SetCellsByColumn(table,list(cat1,cat4),
                    list(spss.CellText.Number(12),
                     spss.CellText.Number(22)))
BasePivotTable.SetCellsByColumn(table,list(cat2,cat3),
                    list(spss.CellText.Number(13),
                     spss.CellText.Number(23)))
BasePivotTable.SetCellsByColumn(table,list(cat2,cat4),
```

```
                    list(spss.CellText.Number(14),
                      spss.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: `spss.CellText.Number(22,formatSpec.Correlation)`. For more information, see the topic CellText.Number Class on p. 40.

Figure 2-7
*Resulting table structure*

| | 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 |

Examples of using the `SetCellsByColumn` method are most easily understood in the context of going through the steps to create a pivot table. For more information, see the topic General Approach to Creating Pivot Tables on p. 26.

### *SetCellsByRow Method*

**.SetCellsByRow(object,rowlabels,cells).** *Sets cell values for the row specified by a set of categories, one for each row dimension.* The argument *object* is a reference to the associated `BasePivotTable` object. The argument *rowlabels* specifies the set of categories that defines the row—a single value or a list. The argument *cells* is a 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*

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

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

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

```
cat6=spss.CellText.String("D")

BasePivotTable.SetCategories(table,rowdim1,list(cat1,cat2))
BasePivotTable.SetCategories(table,rowdim2,list(cat3,cat4))
BasePivotTable.SetCategories(table,coldim,list(cat5,cat6))

BasePivotTable.SetCellsByRow(table,list(cat1,cat3),
                        list(spss.CellText.Number(11),
                             spss.CellText.Number(12)))
BasePivotTable.SetCellsByRow(table,list(cat1,cat4),
                        list(spss.CellText.Number(21),
                             spss.CellText.Number(22)))
BasePivotTable.SetCellsByRow(table,list(cat2,cat3),
                        list(spss.CellText.Number(31),
                             spss.CellText.Number(32)))
BasePivotTable.SetCellsByRow(table,list(cat2,cat4),
                        list(spss.CellText.Number(41),
                             spss.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: `spss.CellText.Number(22,formatSpec.Correlation)`. For more information, see the topic CellText.Number Class on p. 40.

Figure 2-8
*Resulting table*

| | | 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 |

Examples of using the `SetCellsByRow` method are most easily understood in the context of going through the steps to create a pivot table. For more information, see the topic General Approach to Creating Pivot Tables on p. 26.

### SetCellValue Method

**.SetCellValue(object,categories,cell).** *Sets the value of the specified cell.* The argument *object* is a reference to the associated `BasePivotTable` object. The argument *categories* is the list of the category values that specifies the cell—one value for each of the dimensions in the pivot table. The argument *cell* is the cell value. Category values and the cell value must be specified as CellText objects (one of `CellText.Number`, `CellText.String`, `CellText.VarName`, or `CellText.VarValue`).

- In the list that specifies the category values, the first element corresponds to the first appended dimension, the second element to the second appended dimension, and so on. For example, if the pivot table has two row dimensions and one column dimension and the row dimensions are appended before the column dimension, then:

  ```
  list(rowdim1_cat1,rowdim2_cat3,coldim_cat1)
  ```

specifies the cell whose category values are `rowdim1_cat1` in the first appended row dimension, `rowdim2_cat3` in the second appended row dimension, and `coldim_cat1` in the column dimension.

For an example of using the `SetCellValue` method, see the CellText.toNumber method.

### *SetDefaultFormatSpec Method*

**.SetDefaultFormatSpec(object,formatSpec,varIndex).** *Sets the default format for CellText.Number objects.* The argument *object* is a reference to the associated `BasePivotTable` object. The argument *formatSpec* is of the form `formatSpec.format` where `format` is one of those listed in Table 2-2 on p. 69—for example, `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*

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

BasePivotTable.SetCategories(table,rowdim,list(spss.CellText.String("row1"),
                                               spss.CellText.String("row2")))
BasePivotTable.SetCategories(table,coldim,spss.CellText.String("col1"))

BasePivotTable.SetCellValue(table,list(spss.CellText.String("row1"),
                                       spss.CellText.String("col1")),
                                       spss.CellText.Number(2.37))
BasePivotTable.SetCellValue(table,list(spss.CellText.String("row2"),
                                       spss.CellText.String("col1")),
                                       spss.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.

### *TitleFootnotes Method*

**.TitleFootnotes(object,footnotes).** *Used to add a footnote to the table title.* The argument *object* is a reference to the associated `BasePivotTable` object. The argument *footnotes* is a string specifying the footnote.

#### *Example*

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

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

## *CellText Objects*

`CellText` objects are used to create a dimension category or a cell in a pivot table and are only for use with the `BasePivotTable` class. The following object types are available:

- 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.

### *CellText.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 pass an R POSIXt date/time object to this argument. The optional argument *formatspec* is of the form `formatSpec.format` where `format` is one of those listed in the table below—for example, `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.

#### *Example*

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

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

BasePivotTable.SetCategories(table,rowdim,list(spss.CellText.String("row1"),
                                               spss.CellText.String("row2")))
BasePivotTable.SetCategories(table,coldim,spss.CellText.String("col1"))

BasePivotTable.SetCellValue(table,list(spss.CellText.String("row1"),
                                       spss.CellText.String("col1")),
                                       spss.CellText.Number(25.632,formatSpec.Mean,2))
BasePivotTable.SetCellValue(table,list(spss.CellText.String("row2"),
                                       spss.CellText.String("col1")),
                                       spss.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 2-1
*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.

### CellText.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 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

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

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

BasePivotTable.SetCategories(table,rowdim,list(spss.CellText.String("row1"),
                                               spss.CellText.String("row2")))
BasePivotTable.SetCategories(table,coldim,spss.CellText.String("col1"))

BasePivotTable.SetCellValue(table,list(spss.CellText.String("row1"),
                                       spss.CellText.String("col1")),
                                       spss.CellText.String("1"))
BasePivotTable.SetCellValue(table,list(spss.CellText.String("row2"),
                                       spss.CellText.String("col1")),
                                       spss.CellText.String("2"))
```

### CellText.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

```
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")
coldim=BasePivotTable.Append(table,Dimension.Place.column,"coldim")
rowdim=BasePivotTable.Append(table,Dimension.Place.row,"rowdim")
BasePivotTable.SetCategories(table,rowdim,list(spss.CellText.VarName(0),
                                               spss.CellText.VarName(1)))
BasePivotTable.SetCategories(table,coldim,spss.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.

### CellText.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

```
table = spss.BasePivotTable("Table Title",
                            "OMS table subtype")
coldim=BasePivotTable.Append(table,Dimension.Place.column,"coldim")
rowdim=BasePivotTable.Append(table,Dimension.Place.row,"rowdim")
BasePivotTable.SetCategories(table,rowdim,list(spss.CellText.VarValue(0,1),
                                               spss.CellText.VarValue(0,2)))
BasePivotTable.SetCategories(table,coldim,spss.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.

### CellText.toNumber Method

**CellText.toNumber(object).** 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

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

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

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

BasePivotTable.SetCategories(table,rowdim,list(row_cat1,row_cat2))
BasePivotTable.SetCategories(table,coldim,list(col_cat1,col_cat2))

BasePivotTable.SetCellValue(table,list(row_cat1,col_cat1),spss.CellText.Number(11))
cellValue = CellText.toNumber(BasePivotTable.GetCellValue(table,list(row_cat1,col_cat1)))
BasePivotTable.SetCellValue(table,list(row_cat2,col_cat2),spss.CellText.Number(2*cellValue))
```

### CellText.toString Method

**CellText.toString(object).** 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*

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

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

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

BasePivotTable.SetCategories(table,rowdim,list(row_cat1,row_cat2))
BasePivotTable.SetCategories(table,coldim,list(col_cat1,col_cat2))

BasePivotTable.SetCellValue(table,list(row_cat1,col_cat1),spss.CellText.String("abc"))
cellValue = CellText.toString(BasePivotTable.GetCellValue(table,list(row_cat1,col_cat1)))
BasePivotTable.SetCellValue(table,list(row_cat2,col_cat2),
                            spss.CellText.String(paste(cellValue,"d",sep="")))
```

## *Creating a Warnings Table*

You can create an IBM® SPSS® Statistics Warnings table using the `spss.BasePivotTable` function by specifying `"Warnings"` for the *templateName* argument. Note that an 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*

```
BEGIN PROGRAM R.
spsspkg.StartProcedure("demo")
msg=spss.CellText.String("First line of Warnings table content
Second line of Warnings table content")
table=spss.BasePivotTable("Warnings ","Warnings")
rowdim=BasePivotTable.Append(table,Dimension.Place.row,"rowdim",
                             hideName=TRUE,hideLabels=TRUE)
cat=spss.CellText.String("1")
BasePivotTable.SetCategories(table,rowdim,cat)
BasePivotTable.SetCellValue(table,cat,msg)
spsspkg.EndProcedure()
END PROGRAM.
```

- The *title* argument to the `spss.BasePivotTable` function 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 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.

#### *Result*

Figure 2-9
*Warnings table*

**Warnings**

| First line of Warnings table content<br>Second line of Warnings table content |
|---|

# GetSPSSPlugInVersion Function

This function is deprecated for release 18 and higher. Please use the spsspkg.GetSPSSPlugInVersion function instead.

# GetSPSSVersion Function

This function is deprecated for release 18 and higher. Please use the spsspkg.GetSPSSVersion function instead.

# spssdata Functions

The `spssdata` group of functions enables R programs to exchange data with IBM® SPSS® Statistics. It includes functions to read cases from the SPSS Statistics active dataset into an R data frame and to create a new SPSS Statistics dataset from an R data frame.

## spssdata.CloseDataConnection Function

**spssdata.CloseDataConnection().** *Closes a data connection created by the GetSplitDataFromSPSS function.* This function only applies to data connections created to handle IBM® SPSS® Statistics datasets containing split groups. `CloseDataConnection` should be called when done reading from such datasets.

- Data connections are implicitly closed at the end of a `BEGIN PROGRAM R-END PROGRAM` block.

For an example of using `CloseDataConnection` see the topic on the GetSplitDataFromSPSS function.

## spssdata.GetCaseCount Function

**spssdata.GetCaseCount().** *Returns the number of cases (rows) in the active IBM® SPSS® Statistics dataset.*

### Example

```
ncases <- spssdata.GetCaseCount()
casedata <- spssdata.GetDataFromSPSS()
sampledata <- casedata[sample(1:ncases,0.1*ncases),]
```

## spssdata.GetDataFromSPSS Function

**spssdata.GetDataFromSPSS(variables,cases,row.label,keepUserMissing,missingValueToNA,factorMode, rDate,dateVar,asList,orderedContrast)** *Retrieves case data from the active dataset and returns it as an R data frame or optionally as a list.* All of the arguments are optional. If no arguments are provided and the active dataset does not have split groups, the function retrieves all cases for all variables in the active dataset. If the active dataset has split groups and no arguments are

provided, the function retrieves all cases in the first split group. For the default of returning an R data frame, each retrieved case is stored as a row in the data frame. The option of returning the data as a list is most useful when retrieving large datasets since the list structure requires less memory than the data frame structure.

- The argument *variables* specifies the set of variables whose case values will be retrieved. The argument can be a character vector or list specifying the variable names, a character string consisting of variable names separated by blanks, or a numeric vector or list of integers specifying the index values of the variables (index values represent position in the dataset, starting with 0 for the first variable in file order). Variable names must match case with the names as they exist in the active dataset's dictionary. If the argument is omitted, all variables will be retrieved.

  When specifying variable names, you can use `TO` to indicate a range of variables. For example, `variables=c("age TO income")` specifies *age*, *income*, and all variables between them in the active dataset's dictionary. You can also specify a range of values using index values, as in `variables=c(2:4)`, which specifies the variables with index values 2 through 4.

- The argument *cases* is an integer specifying the number of cases to retrieve, beginning with the first case in the active dataset. If the argument is omitted and the active dataset has no split groups, all cases will be retrieved. If the active dataset has split groups and the argument is either omitted or greater than or equal to the number of cases in the first split group, then all cases in the first split group are retrieved.

- The argument *row.label* specifies a variable from the active dataset whose case values will be the row labels of the resulting data frame. The argument is either the variable name or the index value of the variable (index values represent position in the dataset, starting with 0 for the first variable in file order). If the argument is omitted, the row labels will be the default used by R. The argument has no effect if `asList=TRUE`.

- The argument *keepUserMissing* specifies whether user-missing values should be treated as valid data. The argument is boolean and the default is *FALSE*, meaning that user-missing values are treated as missing. With *keepUserMissing* set to *FALSE* (or omitted), user-missing values of string variables are converted to the R *NA* value. The handling of missing values of numeric variables depends on the argument *missingValueToNA* described below.

- The argument *missingValueToNA* specifies whether missing values of numeric variables are converted to the R *NA* value. The argument is boolean and the default is *FALSE*, which specifies that missing values (system and user) of numeric variables are converted to the R *NaN*.

- The argument *factorMode* specifies whether categorical variables from IBM® SPSS® Statistics(variables with a measurement level of nominal or ordinal) are converted to R factors. The value `"none"` is the default and specifies that categorical variables are not converted to factors. The value `"levels"` specifies that categorical variables are converted to factors whose levels are the values of the variables. The value `"labels"` specifies that categorical variables are converted to factors whose levels are the value labels of the variables. Values for which value labels do not exist have a level equal to the value itself. Ordinal variables are converted to ordered factors and nominal variables are converted to unordered factors.

- The argument *rDate* specifies how variables in *dateVar*, with date or datetime formats, are converted to R date/time objects. The value `"none"` is the default and specifies that no conversion will be done. The value `"POSIXct"` specifies to convert to R `POSIXct` objects and `"POSIXlt"` specifies to convert to R `POSIXlt` objects.
- The argument *dateVar* specifies a set of SPSS Statistics variables with date or datetime formats to convert to R date/time objects. The argument supports the same options for specifying variables as described for the *variables* argument. If the argument is omitted and *rDate* specifies a POSIXt object, then all variables with date or datetime formats are converted.
- The argument *asList* specifies whether the result from `GetDataFromSPSS` is a list. The argument is boolean with a default of *FALSE*, which specifies that the result is returned as a data frame. If *asList* is *TRUE* the result is a list with an element for each retrieved variable. Setting *asList* to *TRUE* is most useful when retrieving large datasets since the list structure requires less memory than the default data frame structure.
- The argument *orderedContrast* specifies a contrast function to associate with the ordered factors created from any ordinal variables retrieved from SPSS Statistics. It only applies (to ordinal variables) in the case that *factorMode* is set to `"levels"` or `"labels"`. You can specify any valid contrast function, as a quoted string, such as `"contr.helmert"`. The default is `"contr.treatment"`.
- If the active dataset has split groups, `GetDataFromSPSS` will only return data from the first split group. To get data from SPSS Statistics datasets with split groups, use the `GetSplitDataFromSPSS` function.
- 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 retrieving data with `GetDataFromSPSS`.
- String values are right-padded to the defined width of the string variable.
- The default value of *FALSE* for *asList* results in strings being returned as R factors. Set *asList* to *TRUE* if you don't want strings to be returned as factors. Note, however, that with *asList* set to *TRUE*, the result from `GetDataFromSPSS` is a list, not a data frame.
- Values retrieved from SPSS Statistics variables with time formats are returned as integers representing the number of seconds from midnight.

#### Example

```
DATA LIST FREE /var1 (F2) var2 (A2) var3 (F2) var4 (F2).
BEGIN DATA
11 ab 13 14
21 cd 23 24
31 ef 33 34
END DATA.
BEGIN PROGRAM R.
casedata <- spssdata.GetDataFromSPSS()
print(casedata)
END PROGRAM.
```

#### Result

```
  var1 var2 var3 var4
1   11   ab   13   14
2   21   cd   23   24
```

```
3    31    ef    33    34
```

- Since the argument *row.label* was not specified, the row labels are the default provided by R.
- The column labels of the resulting data frame are the names of the variables retrieved from the active dataset.

## spssdata.GetDataSetList Function

**spssdata.GetDataSetList().** *Returns the names of all defined IBM® SPSS® Statistics datasets in the current session.* The function returns '*' for the active dataset if it is unnamed.

- Datasets that have been declared (with `DATASET DECLARE`) but not yet opened are included in the list. To obtain only the list of open datasets, use the GetOpenedDataSetList function.

### Example

```
datasets <- spssdata.GetDataSetList()
```

## spssdata.GetFileHandles Function

**spssdata.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. The string value `"None"` is returned for the encoding if no encoding is specified. File handles are created with the `FILE HANDLE` command.

### Example

```
handles <- spssdata.GetFileHandles()
```

## spssdata.GetOpenedDataSetList Function

**spssdata.GetOpenedDataSetList().** *Returns the names of the open IBM® SPSS® Statistics datasets in the current session.*The function returns '*' for the active dataset if it is unnamed.

- Datasets that have been declared (with `DATASET DECLARE`) but not yet opened are NOT included in the list. To obtain the list of all defined datasets, use the GetDataSetList function.

### Example

```
datasets <- spssdata.GetOpenedDataSetList()
```

## *spssdata.GetSplitDataFromSPSS Function*

**spssdata.GetSplitDataFromSPSS(variables,row.label,keepUserMissing,missingValueToNA,factorMode, rDate,dateVar,orderedContrast).** *Retrieves the case data for a split group from the active dataset, and returns it as an R data frame.* Each retrieved case is stored as a row in the resulting R data frame.

- The argument *variables* specifies the set of variables whose case values will be retrieved. The argument can be a character vector or list specifying the variable names, a character string consisting of variable names separated by blanks, or a numeric vector or list of integers specifying the index values of the variables (index values represent position in the dataset, starting with 0 for the first variable in file order). Variable names must match case with the names as they exist in the active dataset's dictionary. If the argument is omitted, all variables will be retrieved.

  When specifying variable names, you can use `TO` to indicate a range of variables. For example, `variables=c("age TO income")` specifies *age*, *income*, and all variables between them in the active dataset's dictionary. You can also specify a range of values using index values, as in `variables=c(2:4)`, which specifies the variables with index values 2 through 4.
- The argument *row.label* specifies a variable from the active dataset whose case values will be the row labels of the resulting data frame. The argument is either the variable name or the index value of the variable (index values represent position in the dataset, starting with 0 for the first variable in file order). If the argument is omitted, the row labels will be the default used by R.
- The argument *keepUserMissing* specifies whether user-missing values should be treated as valid data. The argument is boolean and the default is *FALSE*, meaning that user-missing values are treated as missing. With *keepUserMissing* set to *FALSE* (or omitted), user-missing values of string variables are converted to the R *NA* value. The handling of missing values of numeric variables depends on the argument *missingValueToNA* described below.
- The argument *missingValueToNA* specifies whether missing values of numeric variables are converted to the R *NA* value. The argument is boolean and the default is *FALSE*, which specifies that missing values (system and user) of numeric variables are converted to the R *NaN*.
- The argument *factorMode* specifies whether categorical variables from IBM® SPSS® Statistics(variables with a measurement level of nominal or ordinal) are converted to R factors. The value `"none"` is the default and specifies that categorical variables are not converted to factors. The value `"levels"` specifies that categorical variables are converted to factors whose levels are the values of the variables. The value `"labels"` specifies that categorical variables are converted to factors whose levels are the value labels of the variables. Values for which value labels do not exist have a level equal to the value itself. Ordinal variables are converted to ordered factors and nominal variables are converted to unordered factors.
- The argument *rDate* specifies how variables in *dateVar*, with date or datetime formats, are converted to R date/time objects. The value `"none"` is the default and specifies that no conversion will be done. The value `"POSIXct"` specifies to convert to R `POSIXct` objects and `"POSIXlt"` specifies to convert to R `POSIXlt` objects.

- The argument *dateVar* specifies a set of SPSS Statistics variables with date or datetime formats to convert to R date/time objects. The argument supports the same options for specifying variables as described for the *variables* argument. If the argument is omitted and *rDate* specifies a POSIXt object, then all variables with date or datetime formats are converted.
- The argument *orderedContrast* specifies a contrast function to associate with the ordered factors created from any ordinal variables retrieved from SPSS Statistics. It only applies (to ordinal variables) in the case that *factorMode* is set to `"levels"` or `"labels"`. You can specify any valid contrast function, as a quoted string, such as `"contr.helmert"`. The default is `"contr.treatment"`.
- Each call to `GetSplitDataFromSPSS` returns the case data for the next split group from the active dataset. For example, the first call to `GetSplitDataFromSPSS` returns the data for the first split group, the second call returns the data for the second split group, and so on. In the case that the active dataset has no split groups, `GetSplitDataFromSPSS` returns all cases on its first call.
- `GetSplitDataFromSPSS` returns *NULL* if there are no more split groups in the active dataset.
- 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 retrieving data with `GetSplitDataFromSPSS`.

#### Example

```
varnames <- spssdata.GetSplitVariableNames()
if(length(varnames) > 0)
{
   while (!spssdata.IsLastSplit()){
      data <- spssdata.GetSplitDataFromSPSS()
      cat("\n\nSplit variable values:")
      for (name in varnames) cat("\n",name,":",data[1,name])
      cat("\nCases in Split: ",length(data[,1]))
   }
   spssdata.CloseDataConnection()
}
```

## spssdata.GetSplitVariableNames Function

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

For an example of using `GetSplitVariableNames`, see the topic on the GetSplitDataFromSPSS function.

## spssdata.IsLastSplit Function

**spssdata.IsLastSplit().** *Indicates if the current split group is the last one in the active dataset, for datasets with splits.* The result is a logical value—*TRUE* if the current split group is the last one, otherwise *FALSE*.

For an example of using `IsLastSplit` see the topic on the GetSplitDataFromSPSS function.

### spssdata.SetDataToSPSS Function

**spssdata.SetDataToSPSS(datasetName,x,categoryDictionary).** *Populates the case data for a new IBM® SPSS® Statistics dataset.*

- The argument *datasetName* is the name of the SPSS Statistics dataset as specified on the call to the `SetDictionaryToSPSS` function used to create the dataset.
- The argument *x* is an R data frame whose rows represent cases and whose columns represent the variables of the resulting SPSS Statistics dataset. Values in the first column of the data frame populate the first variable in the dataset, values in the second column of the data frame populate the second variable in the dataset, and so on.
- The optional argument *categoryDictionary* specifies the name of a category dictionary created with the `GetCategoricalDictionaryFromSPSS` function. Category dictionaries are for use when retrieving categorical variables (variables with a measurement level of `"nominal"` or `"ordinal"`) into labelled R factors, with the intention of writing those factors to a new dataset.
- The `SetDictionaryToSPSS` function must be called prior to calling `SetDataToSPSS` in order to specify the dictionary for the new dataset.
- Logical, integer, and double values from R are mapped to numeric values in SPSS Statistics.
- For numeric variables, the R *NaN* and *NA* values are converted to the system-missing value in SPSS Statistics.
- When setting values for a SPSS Statistics variable with a date or datetime format, specify the values as R POSIXt objects, which will then be correctly converted to the values appropriate for SPSS Statistics. Also note that SPSS Statistics variables with a time format are stored as the number of seconds from midnight.

Examples of using the `SetDataToSPSS` function are best understood in the context of creating a new dataset. For more information, see the topic Writing Results to a New IBM SPSS Statistics Dataset in Chapter 1 on p. 11.

## spssdictionary Functions

The `spssdictionary` group of functions provides tools for working with the dictionaries of IBM® SPSS® Statistics datasets. It includes functions to read dictionary information from the active dataset and to create the dictionary for a new SPSS Statistics dataset.

### spssdictionary.CloseDataset Function

**spssdictionary.CloseDataset(datasetName).** *Closes the specified dataset.* This function closes a dataset created by the `SetDictionaryToSPSS` function. It cannot be used to close an arbitrary open dataset, only datasets created from R. When used, it must be called prior to `EndDataStep`.

## *spssdictionary.CreateSPSSDictionary Function*

**spssdictionary.CreateSPSSDictionary(var1,...,varN).** *Creates an R data frame representation of a dictionary for use with the SetDictionaryToSPSS function.*

- The arguments *var1,...,varN* specify the variables. Each argument is specified as a vector consisting of the following components (component names are optional), and in the presented order:

  **varName.** The variable name.

  **varLabel.** The variable label.

  **varType.** The variable type; 0 for numeric variables, and an integer equal to the defined length for string variables.

  **varFormat.** The display format of the variable, as a character string. Formats for string variables consisting of standard characters are specified as `Aw` where `w` is the defined length of the string—for example, `A4`. Formats for standard numeric variables are specified as `Fw.d`, where `w` 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) and `d` is an optional integer specifying the number of decimal digits. For example, a format of `F5.2` represents a numeric value with a total width of 5, including two decimal positions and a decimal indicator. A listing of the available formats is shown in the table on p. 52. For more information about formats, 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.

  **varMeasurementLevel.** The measurement level of the variable. Possible values are `"nominal"`, `"ordinal"`, and `"scale"`.

*Note:* Missing values, value labels, and custom variable attributes are set with the `SetUserMissing`, `SetValueLabel`, and `SetVariableAttributes` functions from the `spssdictionary` group of functions.

### *Example*

In this example, we create a new dataset but do not populate the case data. To populate the case data, use the `SetDataToSPSS` function.

```
resp <- c("response","",8,"A8","nominal")
int <- c("intercept","",0,"F8.2","scale")
pred <- c("predictor","",0,"F8.2","scale")
dict <- spssdictionary.CreateSPSSDictionary(resp,int,pred)
spssdictionary.SetDictionaryToSPSS("results",dict)
```

For more information on creating new datasets, see Writing Results to a New IBM SPSS Statistics Dataset on p. 11.

### *Format Types*

| |
|---|
| **A.** Standard characters. |
| **AHEX.** Hexadecimal characters. |

| |
|---|
| **COMMA.** Numbers with commas as the grouping symbol and a period as the decimal indicator. For example: 1,234,567.89. |
| **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. |
| **F.** Standard numeric. |
| **IB.** Integer binary. |
| **PIBHEX.** Hexadecimal of PIB (positive integer binary). |
| **P.** Packed decimal. |
| **PIB.** Positive integer binary. |
| **PK.** Unsigned packed decimal. |
| **RB.** Real binary. |
| **RBHEX.** Hexadecimal of RB (real binary). |
| **Z.** Zoned decimal. |
| **N.** Restricted numeric. |
| **E.** Scientific notation. |
| **DATE.** International date of the general form dd-mmm-yyyy. |
| **TIME.** Time of the general form hh:mm:ss.ss. |
| **DATETIME.** Date and time of the general form dd-mmm-yyyy hh:mm:ss.ss. |
| **ADATE.** American date of the general form mm/dd/yyyy. |
| **JDATE.** Julian date of the general form yyyyddd. |
| **DTIME.** Days and time of the general form dd hh:mm:ss.ss. |
| **WKDAY.** Day of the week. |
| **MONTH.** Month. |
| **MOYR.** Month and year. |
| **QYR.** Quarter and year of the general form qQyyyy. |
| **WKYR.** Week and year. |
| **PCT.** Percentage sign after numbers. |
| **DOT.** Numbers with periods as the grouping symbol and a comma as the decimal indicator. For example: 1.234.567,89. |
| **CCA.** Custom currency format 1. |
| **CCB.** Custom currency format 2. |
| **CCC.** Custom currency format 3. |
| **CCD.** Custom currency format 4. |
| **CCE.** Custom currency format 5. |
| **EDATE.** European date of the general form dd.mm.yyyy. |
| **SDATE.** Sortable date of the general form yyyy/mm/dd. |

## *spssdictionary.EditCategoricalDictionary Function*

**spssdictionary.EditCategoricalDictionary(categoryDictionary,newNames).** *Changes the names of one or more variables in a category dictionary.* This is a utility function for use with category dictionaries. Category dictionaries are for use when retrieving categorical variables (variables with a measurement level of `"nominal"` or `"ordinal"`) into labelled R factors, with the intention of writing those factors to a new dataset. If you rename categorical variables when writing them back to IBM® SPSS® Statistics, you must use this function to change the name in the associated category dictionary.

- The argument *categoryDictionary* specifies the name of a category dictionary created with the `GetCategoricalDictionaryFromSPSS` function.
- The argument *newNames* is a vector specifying the names of the variables in the category dictionary. It replaces the existing names, which are defined in the *name* attribute of the category dictionary, and must be the same length as the *name* attribute. The first element of *newNames* replaces the first element in the *name* attribute of the category dictionary, and so on.

#### Example

In this example, we change the name of the variable *oldname* to *newname* in the category dictionary *catdict*.

```
catdict <- spssdictionary.GetCategoricalDictionaryFromSPSS()
names<-replace(catdict$name,match("oldname",catdict$name),"newname")
catdict<-spssdictionary.EditCategoricalDictionary(catdict, names)
```

## spssdictionary.EndDataStep Function

**spssdictionary.EndDataStep().** *Specifies the end of creating a new SPSS Statistics dataset.* This function should be called after completing the steps to create a new IBM® SPSS® Statistics dataset.

Examples of using the `EndDataStep` function are best understood in the context of creating a new dataset. For more information, see the topic Writing Results to a New IBM SPSS Statistics Dataset in Chapter 1 on p. 11.

## spssdictionary.GetCategoricalDictionaryFromSPSS Function

**spssdictionary.GetCategoricalDictionaryFromSPSS(variables).** *Returns a structure containing the values and value labels of the specified categorical variables from the active dataset.* This is a utility function for use when retrieving categorical variables (variables with a measurement level of `"nominal"` or `"ordinal"`) into labeled R factors (`factorMode="labels"` in `GetDataFromSPSS`), with the intention of writing those factors to a new dataset. This is necessary because labeled factors in R do not preserve the original values. The structure returned by this function is referred to as a **category dictionary**.

- The argument *variables* specifies the set of categorical variables. The argument can be a character vector or list specifying the variable names, a character string consisting of variable names separated by blanks, or a numeric vector or list of integers specifying the index values of the variables (index values represent position in the dataset, starting with 0 for the first variable in file order). Variable names must match case with the names as they exist in the active dataset's dictionary. If the argument is omitted, all categorical variables will be retrieved.

  When specifying variable names, you can use `TO` to indicate a range of variables. For example, `variables=c("year TO cylinder")` specifies all categorical variables between *year* and *cylinder* in the active dataset's dictionary. You can also specify a range of values using index values, as in `variables=c(2:4)`, which specifies the categorical variables with index values in the range 2 through 4.

*Note*: Any scale variables in the specified set of variables are ignored.

#### Example

```
categoryDictionary <- spssdictionary.GetCategoricalDictionaryFromSPSS()
```

## spssdictionary.GetDataFileAttributeNames Function

**spssdictionary.GetDataFileAttributeNames().** *Returns the names of any datafile attributes for the active dataset.*

- The result is a vector of the attribute names.
- If there are no datafile attributes for the active dataset, *NULL* is returned.

#### Example

```
names <- spssdictionary.GetDataFileAttributeNames()
```

## spssdictionary.GetDataFileAttributes Function

**spssdictionary.GetDataFileAttributes(attrName).** *Returns the attribute values 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`.

- The result is a vector of the attribute values.
- If there are no values for the specified attribute *NULL* is returned.

#### Example

```
names <- spssdictionary.GetDataFileAttributeNames()
for(attr in names)
{
    y <- spssdictionary.GetDataFileAttributes(attrName = attr)
    print(y)
}
```

## spssdictionary.GetDictionaryFromSPSS Function

**spssdictionary.GetDictionaryFromSPSS(variables).** *Retrieves variable dictionary information from the active dataset and stores it in an R data frame.* The optional argument *variables* specifies the set of variables whose dictionary information will be retrieved. If the argument is omitted, dictionary information for all variables in the active dataset will be retrieved.

- The argument *variables* can be a character vector or list specifying the variable names, a character string consisting of variable names separated by blanks, or a numeric vector or list of integers specifying the index values of the variables (index values represent position in the dataset, starting with 0 for the first variable in file order). Variable names must match case with the names as they exist in the active dataset's dictionary.

When specifying variable names, you can use `TO` to indicate a range of variables. For example, `variables=c("age TO income")` specifies *age*, *income*, and all variables between them in the active dataset's dictionary. You can also specify a range of values using index values, as in `variables=c(2:4)`, which specifies the variables with index values 2 through 4.

- Each column of the returned data frame contains dictionary information for a single variable. The data frame has the following row labels:

  **varName.** The variable name.

  **varLabel.** The variable label.

  **varType.** The variable type; 0 for numeric variables, and an integer equal to the defined length for string variables.

  **varFormat.** The display format of the variable, as a character string. 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.

  **varMeasurementLevel.** The measurement level of the variable. Possible values are `"nominal"`, `"ordinal"`, `"scale"`, and `"unknown"`. The value "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.

*Note:* Additional variable dictionary information is available from the following functions: `GetUserMissingValues`, `GetValueLabels`, `GetVariableAttributeNames`, and `GetVariableAttributes`.

#### *Example*

```
DATA LIST FREE /id (F4) gender (A1) training (F1).
VARIABLE LABELS id 'Employee ID'
   /training 'Training Level'.
VARIABLE LEVEL id (SCALE)
   /gender (NOMINAL)
   /training (ORDINAL).
VALUE LABELS training 1 'Beginning' 2 'Intermediate' 3 'Advanced'
   /gender 'f' 'Female' 'm' 'Male'.
BEGIN DATA
18 m 1
37 f 2
10 f 3
END DATA.
BEGIN PROGRAM R.
dict <- spssdictionary.GetDictionaryFromSPSS()
print(dict)
END PROGRAM.
```

#### *Result*

```
                              X1        X2             X3
varName                       id    gender       training
varLabel             Employee ID             Training Level
varType                        0         1              0
varFormat                     F4        A1             F1
varMeasurementLevel        scale   nominal        ordinal
```

## *spssdictionary.GetMultiResponseSetNames Function*

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

- The result is a vector of names of multiple response sets.
- If there are no multiple response sets for the active dataset, *NULL* is returned.

#### *Example*

```
names <- spssdictionary.GetMultiResponseSetNames()
```

## *spssdictionary.GetMultiResponseSet Function*

**spssdictionary.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`. If the specified name does not begin with a dollar sign ($), then one is added.

- The result is a list with the following named components: *label* (the label, if any, for the set), *codeAs* (“Dichotomies” or “Categories”), *countedValue* (the counted value—applies only to multiple dichotomy sets), *type* (“Numeric” or “String”), and *vars* (a vector of the elementary variables that define the set).
- If the specified response set does not exist, an exception is thrown.

#### *Example*

```
names <- spssdictionary.GetMultiResponseSetNames()
for(set in names)
{
    y <- spssdictionary.GetMultiResponseSet(mrsetName = set)
    print(y)
}
```

## *spssdictionary.GetUserMissingValues Function*

**spssdictionary.GetUserMissingValues(variable).** *Returns the user-missing values for the specified variable in the active dataset.* The argument can be a character string specifying the variable name or an integer specifying the index value of the variable (index values represent position in the dataset, starting with 0 for the first variable in file order). Variable names must match case with the names as they exist in the active dataset’s dictionary.

- The result is a list. The first element of the list has the name *type* and is a character string specifying the missing value type: 'Discrete' for discrete numeric values, 'Range' for a range of values, 'Range Discrete' for a range of values and a single discrete value, and *NULL* for missing values of a string variable. The second element of the list has the name *missing* and is a vector containing the missing values. The content of the vector for the different missing value types is shown in the following table.

| result$type | result$missing[1] | result$missing[2] | result$missing[3] |
|---|---|---|---|
| 'Discrete' | Discrete value or NaN | Discrete value or NaN | Discrete value or NaN |
| 'Range' | Start point of range | End point of range | NaN |
| 'Range Discrete' | Start point of range | End point of range | Discrete value |
| NULL | Discrete value or NA | Discrete value or NA | Discrete value or NA |

- For string variables, returned values are right-padded to the defined width of the string variable.

#### Example

```
#List all variables without user-missing values
nomissList <- vector()
dict <- spssdictionary.GetDictionaryFromSPSS()
varnames <- dict["varName",]
for (name in varnames){
   vals <- spssdictionary.GetUserMissingValues(name)
   if (is.nan(vals$missing[1]) | is.na(vals$missing[1]))
      nomissList <- c(nomissList,name)
}
{if (length(nomissList) > 0) {
   cat("Variables without user-missing values:\n")
   cat(nomissList,sep="\n")}
else
   cat("All variables have user-missing values")
}
```

## spssdictionary.GetValueLabels Function

**spssdictionary.GetValueLabels(variable).** *Returns the value labels for the specified variable in the active dataset.* The argument can be a character string specifying the variable name or an integer specifying the index value of the variable (index values represent position in the dataset, starting with 0 for the first variable in file order). Variable names must match case with the names as they exist in the active dataset's dictionary.

- The result is a list. The first element of the list is a vector of values that have associated labels. The second element of the list is a vector with the associated labels.
- If there are no value labels, the list contains empty vectors.

#### Example

List all variables without value labels.

```
*R_vars_no_value_labels.sps.
BEGIN PROGRAM R.
novallabelList <- vector()
dict <- spssdictionary.GetDictionaryFromSPSS()
varnames <- dict["varName",]
for (name in varnames){
   if (length(spssdictionary.GetValueLabels(name)$values)==0)
       novallabelList <- append(novallabelList,name)
}
{if (length(novallabelList) > 0) {
   cat("Variables without value labels:\n")
   cat(novallabelList,sep="\n")}
else
   cat("All variables have value labels")
}
END PROGRAM.
```

## *spssdictionary.GetVariableAttributeNames Function*

**spssdictionary.GetVariableAttributeNames(variable).** *Returns the names of any variable attributes for the specified variable in the active dataset.* The argument can be a character string specifying the variable name or an integer specifying the index value of the variable (index values represent position in the dataset, starting with 0 for the first variable in file order). Variable names must match case with the names as they exist in the active dataset's dictionary.

- The result is a vector of the attribute names.
- If there are no attributes for the specified variable, *NULL* is returned.

### *Example*

```
#Create a list of variables that have a specified attribute
varList <- vector()
attribute <- "demographicvars"
dict <- spssdictionary.GetDictionaryFromSPSS()
varnames <- dict["varName",]
for (name in varnames){
   if (any(attribute==spssdictionary.GetVariableAttributeNames(name)))
       varList <- c(varList,name)
}
{if (length(varList) > 0){
   cat(paste("Variables with attribute ",attribute,":\n"))
   cat(varList,sep="\n")}
else
   cat(paste("No variables have the attribute ",attribute))
}
```

## *spssdictionary.GetVariableAttributes Function*

**spssdictionary.GetVariableAttributes(variable,attrName).** *Returns the value(s) for the specified variable and attribute in the active dataset.* The argument *variable* can be a character string specifying the variable name or an integer specifying the index value of the variable (index values represent position in the dataset, starting with 0 for the first variable in file order). Variable names must match case with the names as they exist in the active dataset's dictionary. The argument

*attrName* is a character string that specifies the name of the attribute—for instance, a name returned by `GetVariableAttributeNames`.

- In the case that the specified attribute is an array, the returned value is a vector containing the attribute values.

#### Example

```
#Create a list of variables with a specified attribute and value
varList <- vector()
dict <- spssdictionary.GetDictionaryFromSPSS()
varnames <- dict["varName",]
attr <- "demographicvartypes"
val <- "2"
for (name in varnames){
   attrNames <- spssdictionary.GetVariableAttributeNames(name)
   if(any(attr==attrNames)){
      if(any(val==spssdictionary.GetVariableAttributes(name,attr)))
         varList <- c(varList,name)
   }
}
{if (length(varList) > 0){
   cat(paste("Variables with attribute value ",val,
             " for attribute ",attr,":\n"))
   cat(varList,sep="\n")}
else
   cat(paste("No variables have the attribute value ",val,
             " for attribute ",attr))
}
```

## spssdictionary.GetVariableCount Function

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

#### Example

```
nvars <- spssdictionary.GetVariableCount()
```

## spssdictionary.GetVariableFormat Function

**spssdictionary.GetVariableFormat(variable).** *Returns a string containing the display format for the specified variable in the active dataset.* The argument is the name of the variable in quotes or 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 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.

  To obtain the type code shown in the table, use the GetVariableFormatType function.

#### *Example*

```
varformat <- spssdictionary.GetVariableFormat(0)
```

## *spssdictionary.GetVariableFormatType Function*

**spssdictionary.GetVariableFormatType(variable).** *Returns the integer type code associated with the display format for the specified variable in the active dataset.* The argument is the name of the variable in quotes or the index value of the variable. Index values represent position in the active dataset, starting with 0 for the first variable in file order.

- To obtain the string associated with the type code shown in the table, use the GetVariableFormat function.

#### *Example*

```
varformat <- spssdictionary.GetVariableFormat(0)
```

## *spssdictionary.GetVariableLabel Function*

**spssdictionary.GetVariableLabel(variable).** *Returns a character string containing the variable label for the specified variable in the active dataset .* The argument is the name of the variable in quotes or the index value of the variable. 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*

```
varlabel <- spssdictionary.GetVariableLabel("mpg")
```

## *spssdictionary.GetVariableMeasurementLevel Function*

**spssdictionary.GetVariableMeasurementLevel(variable).** *Returns a string value that indicates the measurement level for the specified variable in the active dataset.* The argument is the name of the variable in quotes or 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 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*

```
varlevel <- spssdictionary.GetVariableMeasurementLevel(0)
```

## spssdictionary.GetVariableName Function

**spssdictionary.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 of the variable. Index values represent position in the active dataset, starting with 0 for the first variable in file order.

#### Example

```
firstvar <- spssdictionary.GetVariableName(0)
```

## spssdictionary.GetVariableType Function

**spssdictionary.GetVariableType(variable).** *Returns 0 for numeric variables or the defined length for string variables for the specified variable in the active dataset.* The argument is the name of the variable in quotes or 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

```
vartype <- spssdictionary.GetVariableType(0)
```

## spssdictionary.GetWeightVariable Function

**spssdictionary.GetWeightVariable().** *Returns the name of the weight variable, or NULL if unweighted.*

#### Example

```
{if (spssdictionary.IsWeighting())
   cat(paste("The weight variable is: ",spssdictionary.GetWeightVariable()))
else
   cat("Weighting is not in effect")}
```

## spssdictionary.IsWeighting Function

**spssdictionary.IsWeighting().** *Indicates if weighting is in effect for the active dataset.* The result is a logical value—*TRUE* if weighting is in effect, otherwise *FALSE*.

#### Example

```
{if (spssdictionary.IsWeighting())
   cat("Weighting is in effect")
else
   cat("Weighting is not in effect")
}
```

## spssdictionary.SetActive Function

**spssdictionary.SetActive(datasetName).** *Sets the specified dataset as the active dataset.* This function can only be called during the process of creating a new IBM® SPSS® Statistics dataset. When used, this function must be called prior to calling `EndDataStep`.

## spssdictionary.SetDataFileAttributes Function

**spssdictionary.SetDataFileAttributes(datasetName,attr1,...,attrN).** *Sets datafile attributes.* This function is used to define datafile attributes for new IBM® SPSS® Statistics datasets created with the `SetDictionaryToSPSS` function.

- The argument *datasetName* is the name of the SPSS Statistics dataset as specified on the call to the `SetDictionaryToSPSS` function used to create the dataset.
- The arguments *attr1,...,attrN* specify the attributes and are of the form `attrName=attrValue`, where `attrName` is the name of the attribute and `attrValue` is either a single character value or a character vector. Specifying a vector results in an attribute array.
- The `SetDataFileAttributes` function should be called after `SetDictionaryToSPSS` and before calling `EndDataStep`.

### Example

```
spssdictionary.SetDataFileAttributes("results",ver="1",Date="7/20/2007")
```

## spssdictionary.SetDictionaryToSPSS Function

**spssdictionary.SetDictionaryToSPSS(datasetName,x,categoryDictionary,hidden).** *Creates a new IBM® SPSS® Statistics dataset with a specified dictionary.*

- The argument *datasetName* specifies the name of the SPSS Statistics dataset to be created, and cannot be the name of an existing dataset.
- The argument *x* is a data frame representing the dictionary and must be an object created by either the `CreateSPSSDictionary` function or the `GetDictionaryFromSPSS` function.
- The optional argument *categoryDictionary* specifies the name of a category dictionary created with the `GetCategoricalDictionaryFromSPSS` function. Category dictionaries are for use when retrieving categorical variables (variables with a measurement level of `"nominal"` or `"ordinal"`) into labeled R factors (`factorMode="labels"` in `GetDataFromSPSS`), with the intention of writing those factors to a new dataset. The value labels of the original categorical variables are automatically added to the new dataset.
- The optional argument *hidden* specifies whether the Data Editor window associated with the new dataset is hidden—by default, it is displayed. Use `hidden=TRUE` to hide the associated Data Editor window.
- The resulting SPSS Statistics dataset is NOT set to be the active dataset. To make a new dataset the active one, use the `SetActive` function or the `DATASET ACTIVATE` command (outside of the program block that calls `SetDictionaryToSPSS`).

Examples of using the `SetDictionaryToSPSS` function are best understood in the context of creating a new dataset. For more information, see the topic Writing Results to a New IBM SPSS Statistics Dataset in Chapter 1 on p. 11.

## *spssdictionary.SetMultiResponseSet Function*

**spssdictionary.SetMultiResponseSet(datasetName,mrsetName,mrsetLabel,codeAs,countedValue,elementaryVars).** *Defines multiple response sets.* This function is used to define multiple response sets for new IBM® SPSS® Statistics datasets created with the `SetDictionaryToSPSS` function.

- The argument *datasetName* is the name of the SPSS Statistics dataset as specified on the call to the `SetDictionaryToSPSS` function used to create the dataset.
- The argument *mrsetName* is the name of the multiple response set and is a string of maximum length 63 bytes that must follow 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.
- The optional argument *mrsetLabel* is a string specifying a label for the set, and cannot be wider than the limit for SPSS Statistics variable labels.
- The argument *codeAs* specifies the variable coding and must be “Dichotomies” (multiple dichotomy set) or “Categories” (multiple category set).
- The argument *countedValue* specifies 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 can be an integer or a string respresentation of an integer. If the set type is string, the value, after trimming trailing blanks, cannot be wider than the narrowest elementary variable. *countedValue* is required if *codeAs* has the value “Dichotomies” and is ignored otherwise.
- The argument *elementaryVars* is a character vector specifying the list of elementary variables that define the set (the list must include at least two variables). Variables can be specified by name or index value (index values represent position in the dataset, starting with 0 for the first variable in file order). When specifying variable names, you can use `TO` to indicate a range of variables—for example, `elementaryVars=c("age TO income")` specifies *age*, *income*, and all variables between them in the active dataset’s dictionary. Variable names must match case with the names as they exist in the active dataset’s dictionary. All specified variables must be of the same type (numeric or string).
- The `SetMultiResponseSet` function should be called after `SetDictionaryToSPSS` and before calling `EndDataStep`.

#### *Example*

```
var1 <- c("Newspaper","",0,"F1","scale")
var2 <- c("TV","",0,"F1","scale")
var3 <- c("Web","",0,"F1","scale")
dict <- spssdictionary.CreateSPSSDictionary(var1,var2,var3)
spssdictionary.SetDictionaryToSPSS("results",dict)
spssdictionary.SetMultiResponseSet(datasetName="results",
                                   mrsetName="$mltnews",
                                   mrsetLabel="News Sources",
                                   codeAs="Dichotomies",
                                   countedValue=1,
```

```
                              elementaryVars=c("Newspaper","TV","Web"))
spssdictionary.EndDataStep()
```

## *spssdictionary.SetUserMissing Function*

**spssdictionary.SetUserMissing(datasetName,variable,format,missings).** *Sets user-missing values for a specified variable.* This function is used to define user-missing values for new IBM® SPSS® Statistics datasets created with the `SetDictionaryToSPSS` function.

- The argument *datasetName* is the name of the SPSS Statistics dataset as specified on the call to the `SetDictionaryToSPSS` function used to create the dataset.
- The argument *variable* can be a character string specifying the variable name or an integer specifying the index value of the variable (index values represent position in the dataset, starting with 0 for the first variable in file order). Variable names must match case with the names as they exist in the active dataset's dictionary.
- The argument *format* specifies the missing value type: missingFormat[‘Discrete’] for discrete values, missingFormat[‘Range’] for a range of numeric values, and missingFormat[‘Range Discrete’] for a range of numeric values and a single discrete numeric value.
- The argument *missings* is a vector specifying the missing values. The content of the vector for the different missing value types is shown in the following table.

| format | missingvals[1] | missingvals[2] | missingvals[3] |
|---|---|---|---|
| missingFormat[‘Discrete’] | Discrete value (optional) | Discrete value (optional) | Discrete value (optional) |
| missingFormat[‘Range’] | Start point of range | End point of range | Not applicable |
| missingFormat[‘Range Discrete’] | Start point of range | End point of range | Discrete value |

- Missing values for string variables cannot exceed 8 bytes. (There is no limit on the defined width of the string variable, but defined missing values cannot exceed 8 bytes.)
- The `SetUserMissing` function should be called after `SetDictionaryToSPSS` and before calling `EndDataStep`.

#### *Examples*

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

```
spssdictionary.SetUserMissing("results","newvar",
                              missingFormat["Discrete"],c(0,9,99))
```

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

```
spssdictionary.SetUserMissing("results","newvar",
                              missingFormat["Range"],c(9,99))
```

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

```
spssdictionary.SetUserMissing("results","newvar",
                              missingFormat["Range Discrete"],c(9,99,0))
```

Specify missing values for a new string variable.

```
spssdictionary.SetUserMissing("results","newvar",
                              missingFormat["Discrete"],c(' ','NA'))
```

## spssdictionary.SetValueLabel Function

**spssdictionary.SetValueLabel(datasetName,variable,values,labels).** *Sets the value labels for a specified variable.* This function is used to define value labels for new IBM® SPSS® Statistics datasets created with the `SetDictionaryToSPSS` function.

- The argument *datasetName* is the name of the SPSS Statistics dataset as specified on the call to the `SetDictionaryToSPSS` function used to create the dataset.
- The argument *variable* can be a character string specifying the variable name or an integer specifying the index value of the variable (index values represent position in the dataset, starting with 0 for the first variable in file order). Variable names must match case with the names as they exist in the active dataset's dictionary.
- The argument *values* is a vector specifying the values for which labels will be set.
- The argument *labels* is a vector specifying the labels corresponding to the elements of *values*.
- The `SetValueLabel` function should be called after `SetDictionaryToSPSS` and before calling `EndDataStep`.

### Example

```
values <- c(0,1)
labels <- c("m","f")
spssdictionary.SetValueLabel("results","newvar",values,labels)
```

## spssdictionary.SetVariableAttributes Function

**spssdictionary.SetVariableAttributes(datasetName,variable.attr1,...,attrN).** *Sets the variable attributes for a specified variable.* This function is used to define custom variable attributes for new IBM® SPSS® Statistics datasets created with the `SetDictionaryToSPSS` function.

- The argument *datasetName* is the name of the SPSS Statistics dataset as specified on the call to the `SetDictionaryToSPSS` function used to create the dataset.
- The argument *variable* can be a character string specifying the variable name or an integer specifying the index value of the variable (index values represent position in the dataset, starting with 0 for the first variable in file order). Variable names must match case with the names as they exist in the active dataset's dictionary.
- The arguments *attr1,...,attrN* specify the attributes and are of the form `attrName=attrValue`, where `attrName` is the name of the attribute and `attrValue` is either a single character value or a character vector. Specifying a vector results in an attribute array.
- The `SetVariableAttributes` function should be called after `SetDictionaryToSPSS` and before calling `EndDataStep`.

### Example

```
spssdictionary.SetVariableAttributes("results","Gender",DemographicVars="1",
```

```
Binary="Yes")
```

## spsspivottable.Display Function

The `spsspivottable.Display` function provides the ability to render tabular output from R as a pivot table that can be displayed in the IBM® SPSS® Statistics Viewer or written to an external file using the SPSS Statistics Output Management System.

- By default, the name that appears in the outline pane of the Viewer associated with the pivot table is *R*. You can customize the name and nest multiple pivot tables under a common heading by wrapping the pivot table generation in a `StartProcedure-EndProcedure` block. For more information, see the topic spsspkg.StartProcedure Function on p. 72.
- The `spsspivottable.Display` function is limited to pivot tables with one row dimension and one column dimension. To create more complex pivot tables, use the BasePivotTable class.

**spsspivottable.Display(x,title,templateName,outline,caption,isSplit,rowdim,coldim,hiderowdimtitle, hiderowdimlabel,hidecoldimtitle,hidecoldimlabel,rowlabels,collabels,format).** *Creates a pivot table with one row dimension and one column dimension.* All arguments other than *x* are optional.

- **x.** The data to be displayed as a pivot table. It may be a data frame, matrix, table, or any R object that can be converted to a data frame.
- **title.** A character string that specifies the title that appears with the table. The default is "Rtable".
- **templateName.** A character 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 bytes. The default is "Rtable". Unless you are routing this pivot table with OMS and need to distinguish subtypes you do not need to specify a value.

  When routing pivot table output from R using OMS, the command name associated with this output is `R` by default, as in `COMMANDS=['R']` for the `COMMANDS` keyword on the `OMS` command. If you wrap the pivot table output in a `StartProcedure-EndProcedure` block, then use the name specified in the `StartProcedure` call as the command name. For more information, see the topic spsspkg.StartProcedure Function on p. 72.
- **outline.** A character 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.
- **caption.** A character string that specifies a table caption.
- **isSplit.** A logical value (*TRUE* or *FALSE*) specifying whether to enable split file processing for the table. The default is *TRUE*. Split file processing refers to whether results from different split groups are displayed in separate tables or in the same table but grouped by split, and is controlled by the `SPLIT FILE` command.

  When retrieving data with `spssdata.GetSplitDataFromSPSS`, call `spsspivottable.Display` with the results for each split group. The results from each split group are accumulated and the subsequent table(s) are displayed when `spssdata.CloseDataConnection` is called.

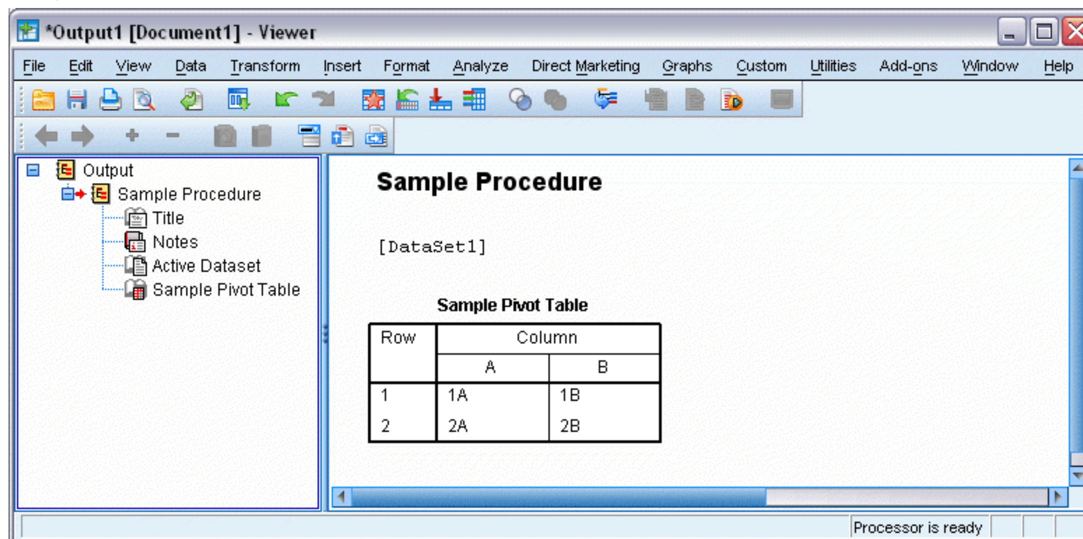
- **rowdim.** A character string specifying a title for the row dimension. The default is “row”.
- **coldim.** A character string specifying a title for the column dimension. The default is “column”.
- **hiderowdimtitle.** A logical value (*TRUE* or *FALSE*) specifying whether to hide the row dimension title. The default is *TRUE*.
- **hiderowdimlabel.** A logical value specifying whether to hide the row labels. The default is *FALSE*.
- **hidecoldimtitle.** A logical value specifying whether to hide the column dimension title. The default is *TRUE*.
- **hidecoldimlabel.** A logical value specifying whether to hide the column labels. The default is *FALSE*.
- **rowlabels.** A numeric or character vector specifying the row labels. If provided, the length of the vector must equal the number of rows in the argument *x*. If omitted, the row names of *x* will be used. If *x* does not have row names, the labels “row1”, “row2”, etc., will be used. If a numeric vector is provided, the row labels will have the format specified by the argument *format*.
- **collabels.** A numeric or character vector specifying the column labels. If provided, the length of the vector must equal the number of columns in the argument *x*. If omitted, the column names of *x* will be used. If *x* does not have column names, the labels “col1”, “col2”, etc., will be used. If a numeric vector is provided, the column labels will have the format specified by the argument *format*.
- **format.** Specifies the format to be used for displaying numeric values, including cell values, row labels, and column labels. The argument *format* is of the form `formatSpec.format` where `format` is one of those listed in the table below—for example, `formatSpec.Correlation`. You can also specify an integer code for *format*, as in the value `3` for `Correlation`. The default format is `GeneralStat`.

#### *Example*

```
BEGIN PROGRAM R.
demo <- data.frame(A=c("1A","2A"),B=c("1B","2B"),row.names=c(1,2))
spsspivottable.Display(demo,
                       title="Sample Pivot Table",
                       rowdim="Row",
                       hiderowdimtitle=FALSE,
                       coldim="Column",
                       hidecoldimtitle=FALSE)
END PROGRAM.
```

### *Result*

Figure 2-10
*Sample Pivot Table*




### *Numeric Formats*

| Format name | Code |
|---|---|
| Coefficient | 0 |
| CoefficientSE | 1 |
| CoefficientVar | 2 |
| Correlation | 3 |
| GeneralStat | 4 |
| Count | 6 |
| Percent | 7 |
| PercentNoSign | 8 |
| Proportion | 9 |
| Significance | 10 |
| Residual | 11 |

### *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.
- `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.

# spsspkg.EndProcedure Function

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

- The `EndProcedure` function must be called to end output initiated with the StartProcedure function.

# spsspkg.GetOutputLanguage Function

**spsspkg.GetOutputLanguage().** *Returns the current IBM® SPSS® Statistics output language.* The value is one of the following strings: "English", "French", "German", "Italian", "Japanese", "Korean", "Polish", "Russian", "SChinese" (Simplified Chinese), "Spanish", "TChinese" (Traditional Chinese), or "BPortugu" (Brazilian Portuguese).

### Example

```
lang <- spsspkg.GetOutputLanguage()
```

# spsspkg.GetSPSSLocale Function

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

### Example

```
locale <- spsspkg.GetSPSSLocale()
```

# spsspkg.GetSPSSPlugInVersion Function

**spsspkg.GetSPSSPlugInVersion().** *Returns the current version of the IBM® SPSS® Statistics - Integration Plug-In for R.*

### Example

```
pluginVer <- spsspkg.GetSPSSPlugInVersion()
```

# spsspkg.GetSPSSVersion Function

**spsspkg.GetSPSSVersion().** *Returns the current IBM® SPSS® Statistics version.*

### Example

```
version <- spsspkg.GetSPSSVersion()
```

# spsspkg.GetStatisticsPath Function

**spsspkg.GetStatisticsPath().** *Returns the path to the installation location of the current instance of IBM® SPSS® Statistics.*

#### *Example*

```
path <- spsspkg.GetStatisticsPath()
```

# spsspkg.processcmd Function

**spsspkg.processcmd(oobj,myArgs,f,excludedargs).** *Parses the values passed to the Run function associated with an extension command, and executes the implementation function.*

- The argument *oobj* is a value returned by the Syntax function. It contains all of the information needed to parse the values passed to the `Run` function associated with an extension command.
- The argument *myArgs* should be specified as:

  ```
  myArgs = args[[2]]
  ```

  where *args* is the argument passed to the `Run` function. *myArgs* contains the run-time values of the keywords associated with the extension command.
- The argument *f* is the name of a function that implements the extension command.
- The argument *excludedargs* is an optional list of arguments to be ignored when checking for arguments required by the implementation function *f*.

*Note*: The `spsspkg.processcmd` function is used in conjunction with the implementation code for an extension command implemented in R. It is not for use within a `BEGIN PROGRAM R-END PROGRAM` block. Specifically, it is for use in the `Run` function that implements the extension command.

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

- The article *"Writing IBM SPSS Statistics Extension Commands"*, available from the SPSS community at *http://www.ibm.com/developerworks/spssdevcentral*.
- The chapter on Extension Commands in *Programming and Data Management for IBM SPSS Statistics*, available from the Articles page on the SPSS community at *http://www.ibm.com/developerworks/spssdevcentral*.
- A tutorial on creating extension commands for R is available by choosing "Working with R" from the Help menu in IBM® SPSS® Statistics.

# spsspkg.SetOutput Function

**spsspkg.SetOutput("value").** *Specifies whether output from R is displayed in the IBM® SPSS® Statistics Viewer.* This applies to explicit output from R, such as from the `print` function. It also applies to graphical output, such as from the R `plot` function. The value of the argument is a quoted string: "ON" to display output from R, "OFF" to suppress the display of output from R. The default is "ON".

- The setting persists for the current SPSS Statistics session.
- It is important to note the interaction of the `spsspkg.SetOutput` function and the `spssRGraphics.SetOutput` function. With `spsspkg.SetOutput("ON")` you can turn off display of graphics with `spssRGraphics.SetOutput("OFF")`. `spsspkg.SetOutput("OFF")`, however, applies to all output and overrides `spssRGraphics.SetOutput("ON")`.

#### Example

```
spsspkg.SetOutput("OFF")
```

## spsspkg.SetOutputLanguage Function

**spsspkg.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).

#### Example

```
spsspkg.SetOutputLanguage('German')
```

## spsspkg.StartProcedure Function

**spsspkg.StartProcedure(pName,omsId=pName).** *Signals the beginning of pivot table or text block output.* The `StartProcedure-EndProcedure` block is used to group output under a common heading, as is typical for output associated with a given procedure.

- The argument *pName* is a string and is the name that appears in the outline pane of the Viewer associated with the output. If the optional argument *omsId* is omitted, then *pName* 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 *omsId* 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 *omsId* is omitted, then the value of the *pName* argument is used as the OMS identifier. *omsId* is only necessary when creating procedures with localized output so that the procedure name can be localized but not the OMS identifier. For more information, see the topic Localizing Output from R in Chapter 1 on p. 18.
- You can include multiple pivot tables and text blocks in a given `StartProcedure-EndProcedure` block.
- In order that names associated with output not conflict with names of existing IBM® SPSS® Statistics commands (when working with OMS), consider using names of the form *yourorganization.com.procedurename*.
- Call the `EndProcedure` function to signal the end of pivot table or text block output.

#### *Example*

```
BEGIN PROGRAM R.
spsspkg.StartProcedure("MyProcedure")
demo <- data.frame(A=c("1A","2A"),B=c("1B","2B"),row.names=c(1,2))
spsspivottable.Display(demo,
                       title="Sample Pivot Table",
                       rowdim="Row",
                       hiderowdimtitle=FALSE,
                       coldim="Column",
                       hidecoldimtitle=FALSE)
spsspkg.EndProcedure()
END PROGRAM.
```

## *spsspkg.Syntax Function*

**spsspkg.Syntax(templ).** *Validates values passed to the Run function of an extension command, according to the specified template.* The argument is a list of `TemplateClass` objects, created with the `spsspkg.Template` function. The list should include a `TemplateClass` object for each possible keyword associated with the syntax for the extension command.

#### *Example*

As an example, consider an extension command named `MYCOMMAND` with the following syntax chart:

```
MYCOMMAND VARIABLES=variable list
[/OPTIONS [MISSING={LISTWISE*}] ]
                   {FAIL     }
```

The associated specification for the `spsspkg.Syntax` function is:

```
oobj<-spsspkg.Syntax(templ=list(
   spsspkg.Template("VARIABLES",subc="",var="vars",ktype="existingvarlist",islist=TRUE),
   spsspkg.Template("MISSING",subc="OPTIONS",var="missing",ktype="str")
             ))
```

*Note*: The `spsspkg.Syntax` function is used in conjunction with the implementation code for an extension command implemented in R. It is not for use within a `BEGIN PROGRAM R-END PROGRAM` block. Specifically, it is for use in the `Run` function that implements the extension command.

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

- The article *"Writing IBM SPSS Statistics Extension Commands"*, available from the SPSS community at *http://www.ibm.com/developerworks/spssdevcentral*.
- The chapter on Extension Commands in *Programming and Data Management for IBM SPSS Statistics*, available from the Articles page on the SPSS community at *http://www.ibm.com/developerworks/spssdevcentral*.
- A tutorial on creating extension commands for R is available by choosing "Working with R" from the Help menu in IBM® SPSS® Statistics.

# spsspkg.Template Function

**spsspkg.Template(kwd,subc,var,ktype,islist,vallist).** *Creates the template for a specified keyword associated with the syntax for an extension command.* The template for a keyword specifies the details needed to process the value (of this keyword) passed to the Run function. The result is a `TemplateClass` object for use with the `Syntax` function.

- The argument *kwd* specifies the name of the keyword (in uppercase) as it appears in the command syntax for the extension command.
- The argument *subc* specifies the name of the subcommand (in uppercase) that contains the keyword. If the keyword belongs to the anonymous subcommand, the argument can be omitted or set to the empty string. The anonymous subcommand is an unnamed subcommand (there can be only one per extension command) and is used to handle keywords that are not part of an explicit subcommand.
- The argument *var* specifies the name of the R variable that receives the value specified for the keyword. If *var* is omitted, the lowercase value of *kwd* is used.
- The argument *ktype* specifies the type of keyword, such as whether the keyword specifies a variable name, a string, or a floating point number. The following values are allowed:

  **bool.** Accepts values of true, false, yes, or no and converts them into the corresponding boolean values of *TRUE* or *FALSE*. If the keyword is declared as *LeadingToken* in the XML specification of the extension command and treated as *bool* here, then the presence or absense of the keyword maps to *TRUE* or *FALSE*.

  **str.** A string or quoted literal or list of either. Values are always converted to lower case. You can use the *vallist* argument to specify a set of allowed values.

  **int.** An integer or list of integers with optional range constraints that you can specify in the *vallist* argument.

  **float.** A real number or list of real numbers with optional range constraints that you can specify in the *vallist* argument.

  **literal.** An arbitrary string or list of strings with no case conversion or validation.

  **varname.** An arbitrary, unvalidated variable name (or syntactical equivalent such as a dataset name) or list of variable names with no case conversion.

  **existingvarlist.** A list of variable names that are validated against the variables in the active dataset. Set *islist* to *TRUE* when using *existingvarlist*. Supports the IBM® SPSS® Statistics`TO` and `ALL` keywords for variable lists. To use `TO` and `ALL`, specify the associated keyword as parameter type `TokenList` in the XML specification for the extension command.

- The argument *islist* is a boolean (*TRUE* or *FALSE*) specifying whether the keyword contains a list of values. The default is *FALSE*.
- The argument *vallist* is an R list, specifying a set of allowed values. For a keyword of type *str*, values are checked in uppercase. For keywords of types *int* and *float*, *vallist* is a 2-element list specifying the lower and upper bounds of the range of allowed values. To specify only an upper bound, use a lower bound of `-Inf`. To specify only a lower bound, use an upper bound of `Inf`.

*Note*: The `spsspkg.Template` function is used in conjunction with the implementation code for an extension command implemented in R. It is not for use within a `BEGIN PROGRAM R-END PROGRAM` block. Specifically, it is for use in the `Run` function that implements the extension command.

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

- The article *"Writing IBM SPSS Statistics Extension Commands"*, available from the SPSS community at *http://www.ibm.com/developerworks/spssdevcentral*.
- The chapter on Extension Commands in *Programming and Data Management for IBM SPSS Statistics*, available from the Articles page on the SPSS community at *http://www.ibm.com/developerworks/spssdevcentral*.
- A tutorial on creating extension commands for R is available by choosing "Working with R" from the Help menu in SPSS Statistics.

# spsspkg.Version Function

**spsspkg.Version().** *Returns the current version of the IBM® SPSS® Statistics - Integration Plug-In for R, including the bug fix version and system information.*

#### Example

```
spsspkg.Version()
```

# spssRGraphics Functions

The `spssRGraphics` group of functions manage the display of graphical output from R in the IBM® SPSS® Statistics Viewer. By default, graphical output from R—for instance, from the R `plot` function—is displayed in the SPSS Statistics Viewer.

## spssRGraphics.Submit Function

**spssRGraphics.Submit(filename).** *Displays a specified R graphic in the IBM® SPSS® Statistics Viewer.* The argument is a character string specifying the path to a graphic file. The supported formats for the graphic file are: png, jpg and bmp.

#### Example

```
spssRGraphics.Submit("/plots/myplot.jpg")
```

*Note*: Since escape sequences in the R programming language begin with a backslash (\)—such as `\n` for newline and `\t` for tab—it is recommended to use forward slashes (/) in file specifications on Windows. In this regard, SPSS Statistics always accepts a forward slash in file specifications.

### spssRGraphics.SetOutput Function

**spssRGraphics.SetOutput("value").** *Controls whether graphical output from R is displayed in the IBM® SPSS® Statistics Viewer.* The value of the argument is a quoted string: "ON" to display graphical output from R, "OFF" to suppress the display of graphical output from R. The default is "ON".

- It is important to note the interaction of the `spssRGraphics.SetOutput` function and the spsspkg.SetOutput function. With `spsspkg.SetOutput("ON")` you can turn off display of graphics with `spssRGraphics.SetOutput("OFF")`. `spsspkg.SetOutput("OFF")`, however, applies to all output and overrides `spssRGraphics.SetOutput("ON")`.

#### Example

```
spssRGraphics.SetOutput("OFF")
```

## spssxmlworkspace Functions

The `spssxmlworkspace` group of functions provides tools for working with the **XML workspace**—an in-memory workspace where you can store (and then retrieve) XML representations of output from IBM® SPSS® Statistics commands as well as an XML representation of the active dataset's dictionary.

### spssxmlworkspace.CreateXPathDictionary Function

**spssxmlworkspace.CreateXPathDictionary(handle).** *Creates an XPath dictionary DOM, for the active IBM® SPSS® Statistics dataset, that can be accessed with XPath expressions.* The argument is a name used to identify this DOM in any subsequent `EvaluateXPath` and `DeleteXmlWorkspaceObject` function calls.

#### Example

```
handle <- "demo"
spssxmlworkspace.CreateXPathDictionary(handle)
```

- The XPath dictionary DOM for the active dataset is assigned the handle name *demo*. Any subsequent `EvaluateXPath` or `DeleteXmlWorkspaceObject` function calls that reference this dictionary DOM must use this handle name.

### spssxmlworkspace.DeleteXmlWorkspaceObject Function

**spssxmlworkspace.DeleteXmlWorkspaceObject(handle).** *Deletes the XPath dictionary DOM or output DOM with the specified handle name.* The argument is the name associated with this DOM, as defined by a previous `CreateXPathDictionary` function call or IBM® SPSS® Statistics `OMS` command.

#### Example

```
handles <- spssxmlworkspace.GetHandleList()
```

```
for(handle in handles)
   spssxmlworkspace.DeleteXmlWorkspaceObject(handle)
```

### spssxmlworkspace.EvaluateXPath Function

**spssxmlworkspace.EvaluateXPath(handle,context,expression).** *Evaluates an XPath expression against a specified XPath DOM and returns the result as a vector of character strings.* The argument *handle* specifies the particular XPath DOM and must be a valid handle name defined by a previous `CreateXPathDictionary` function call or IBM® SPSS® Statistics `OMS` command. The argument *context* defines the XPath context for the expression and should be set to `"/dictionary"` for a dictionary DOM or `"/outputTree"` for an output XML DOM created by the `OMS` command. The argument *expression* 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"
spssxmlworkspace.CreateXPathDictionary(handle)
context <- "/dictionary"
xpath <- "variable/@name"
varnames <- spssxmlworkspace.EvaluateXPath(handle,context,xpath)
```

*Note*: In the 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`.

Documentation for the output schema and the dictionary schema is available from the Help system.

### spssxmlworkspace.GetHandleList Function

**spssxmlworkspace.GetHandleList().** *Returns the names of the currently defined dictionary and output XML DOMs available for use with EvaluateXPath.*

#### Example

```
handles <- spssxmlworkspace.GetHandleList()
```

## TextBlock Class

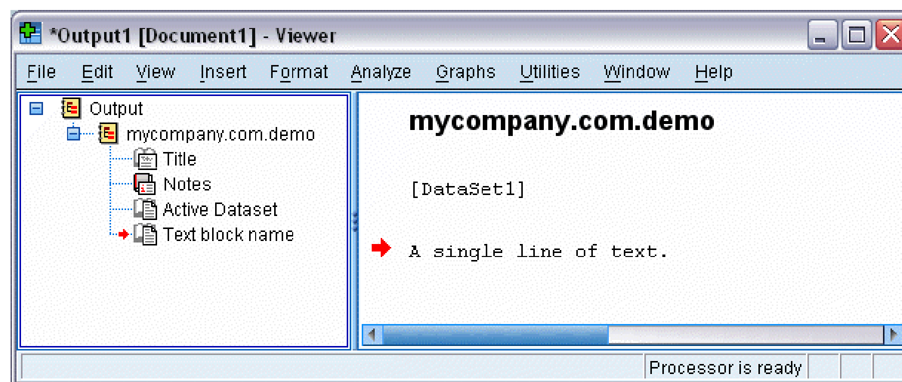
**spss.TextBlock(name,content,outline).** *Creates and populates a text block item in the Viewer.* The argument *name* is a string that specifies the name of this item in the outline pane of the Viewer. The argument *content* is a string that specifies the text, which may include the escape sequence `\n` to specify line breaks. 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.

### *Example*

```
BEGIN PROGRAM R.
spsspkg.StartProcedure("mycompany.com.demo")
textBlock = spss.TextBlock("Text block name",
                           "A single line of text.")
spsspkg.EndProcedure()
END PROGRAM.
```

Figure 2-11
*Sample text block*




- This example shows how to generate a text block within a `StartProcedure-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, which may include the escape sequence `\n` to specify line breaks. 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*

```
BEGIN PROGRAM R.
spsspkg.StartProcedure("mycompany.com.demo")
textBlock = spss.TextBlock("Text block name",
                           "A single line of text.")
TextBlock.append(textBlock,"A second line of text.")
TextBlock.append(textBlock,"A third line of text preceded by a blank line.",skip=2)
spsspkg.EndProcedure()
END PROGRAM.
```

Appendix

# A

# Notices

This information was developed for products and services offered worldwide.

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, Armonk, NY 10504-1785, U.S.A.*

For license inquiries regarding double-byte character set (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., 1623-14, Shimotsuruma, Yamato-shi, Kanagawa 242-8502 Japan.*

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES 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 states 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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply 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 Software Group, Attention: Licensing, 233 S. Wacker Dr., Chicago, IL 60606, USA.*

© Copyright IBM Corporation 1989, 2011.

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 GNU GENERAL PUBLIC LICENSE Version 2.

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-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

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 the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

### *Trademarks*

IBM, the IBM logo, ibm.com, and SPSS are trademarks of IBM Corporation, registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at *http://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.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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.

Other product and service names might be trademarks of IBM or other companies.

# Index


active dataset
  reading into R, 7, 45
append method
  TextBlock class, 78
Append method, 26
Append method (BasePivotTable class), 31

BasePivotTable class, 24
  Append method, 26, 31
  Caption method, 31
  CategoryFootnotes method, 32
  CellText objects, 40
  DimensionFootnotes method, 32
  Footnotes method, 33
  GetCellValue method, 33
  GetDefaultFormatSpec method, 34
  HideTitle method, 34
  Insert method, 26, 34
  SetCategories method, 27, 35
  SetCellsByColumn method, 28, 36
  SetCellsByRow method, 28, 37
  SetCellValue method, 38
  SetDefaultFormatSpec method, 39
  TitleFootnotes method, 39
  Warnings table, 44
BEGIN PROGRAM (command), 1
browser function, 2

Caption method, 31
case count, 45
CategoryFootnotes method, 32
CellText objects, 40
CellText.Number class, 40
CellText.String class, 42
CellText.VarName class, 42
CellText.VarValue class, 42
CloseDataConnection, 45
CloseDataset, 51
CreateSPSSDictionary, 52
CreateXPathDictionary, 76

data
  creating IBM SPSS Statistics datasets, 51, 63
  reading active dataset into R, 7, 45
  writing data from R to IBM SPSS Statistics, 45
data types, 62
datafile attributes
  adding, 63
dates
  reading datetime values into R, 9
debug function, 2
debugging, 2
DeleteXmlWorkspaceObject, 76
dictionary
  CreateXPathDictionary, 76
DimensionFootnotes method, 32

EditCategoricalDictionary, 53
EndDataStep, 54
EndProcedure, 70
EvaluateXPath, 16, 77
extension commands
  parsing command syntax, 73

file handles, 48
Footnotes method, 33
format of variables, 60–61

GetCaseCount, 45
GetCategoricalDictionaryFromSPSS, 54
GetCellValue method, 33
GetDataFileAttributeNames, 55
GetDataFileAttributes, 55
GetDataFromSPSS, 7, 45
GetDataSetList, 48
GetDefaultFormatSpec method, 34
GetDictionaryFromSPSS, 55
GetFileHandles, 48
GetHandleList, 77
GetMultiResponseSet, 57
GetMultiResponseSetNames, 57
GetOpenedDataSetList, 48
GetOutputLanguage, 70
GetSplitDataFromSPSS, 49
GetSplitVariableNames, 50
GetSPSSLocale, 70
GetSPSSPlugInVersion, 70
GetSPSSVersion, 70
GetStatisticsPath, 71
GetUserMissingValues, 57
GetValueLabels, 58
GetVariableAttributeNames, 59
GetVariableAttributes, 59
GetVariableCount, 60
GetVariableFormat, 60
GetVariableFormatType, 61
GetVariableLabel, 61
GetVariableMeasurementLevel, 61
GetVariableName, 62
GetVariableType, 62
GetWeightVariable, 62
graphical output from R, 15

HideTitle method, 34

Insert method, 26, 34
IsLastSplit, 50
IsWeighting, 62


© Copyright IBM Corporation 1989, 2011.

labels
  variable, 61
legal notices, 79
localizing output, 18

measurement level, 61
missing values
  reading data into R, 9
  retrieving user missing value definitions, 57
  setting missing values, 65

names of variables, 62
number of cases (rows), 45
number of variables, 60
numeric variables, 62

output
  reading output results, 77
OXML
  reading output XML, 77

pivot tables, 13, 24
processcmd, 71

R
  file specifications, 4
  syntax rules, 4
R functions and classes, 23
  spssdata functions, 45
  spssdictionary functions, 51
  spsspivottable.Display, 67
  spssRGraphics functions, 75
  spssxmlworkspace functions, 76
  TextBlock class, 77
R graphics, 15
row count, 45

SetActive, 63
SetCategories method, 27, 35
SetCellsByColumn method, 28, 36
SetCellsByRow method, 28, 37
SetCellValue method, 38
SetDataFileAttributes, 63
SetDataToSPSS, 51
SetDefaultFormatSpec method, 39
SetDictionaryToSPSS, 63
SetMultiResponseSet, 64
SetOutput, 71, 76
SetOutputLanguage, 72
SetUserMissing, 65
SetValueLabel, 66
SetVariableAttributes, 66
split-file processing, 50
  reading datasets with splits in R, 10, 49
  split variables, 50
spssdata functions, 45
  CloseDataConnection, 45
  GetCaseCount, 45
  GetDataFromSPSS, 7, 45
  GetDataSetList, 48
  GetFileHandles, 48
  GetOpenedDataSetList, 48
  GetSplitDataFromSPSS, 49
  GetSplitVariableNames, 50
  IsLastSplit, 50
  SetDataToSPSS, 51
spssdictionary functions, 51
  CloseDataset, 51
  CreateSPSSDictionary, 52
  EditCategoricalDictionary, 53
  EndDataStep, 54
  GetCategoricalDictionaryFromSPSS, 54
  GetDataFileAttributeNames, 55
  GetDataFileAttributes, 55
  GetDictionaryFromSPSS, 55
  GetMultiResponseSet, 57
  GetMultiResponseSetNames, 57
  GetUserMissingValues, 57
  GetValueLabels, 58
  GetVariableAttributeNames, 59
  GetVariableAttributes, 59
  GetVariableCount, 60
  GetVariableFormat, 60
  GetVariableFormatType, 61
  GetVariableLabel, 61
  GetVariableMeasurementLevel, 61
  GetVariableName, 62
  GetVariableType, 62
  GetWeightVariable, 62
  IsWeighting, 62
  SetActive, 63
  SetDataFileAttributes, 63
  SetDictionaryToSPSS, 63
  SetMultiResponseSet, 64
  SetUserMissing, 65
  SetValueLabel, 66
  SetVariableAttributes, 66
spsspivottable.Display, 67
spsspkg functions
  EndProcedure, 70
  GetOutputLanguage, 70
  GetSPSSLocale, 70
  GetSPSSPlugInVersion, 70
  GetSPSSVersion, 70
  GetStatisticsPath, 71
  processcmd, 71
  SetOutput, 71
  SetOutputLanguage, 72
  StartProcedure, 72
  Syntax function, 73
  Template function, 74
  Version, 75

spssRGraphics functions, 75
  SetOutput, 76
  Submit, 75
spssxmlworkspace functions, 76
  CreateXPathDictionary, 76
  DeleteXmlWorkspaceObject, 76
  EvaluateXPath, 77
  GetHandleList, 77
StartProcedure, 72
string variables, 62
Submit, 75
Syntax function, 73

Template function, 74
TextBlock class, 77
  append method, 78
TitleFootnotes method, 39
toNumber method, 43
toString method, 43
trademarks, 80

unknown measurement level, 61

value labels
  adding, 66
  retrieving, 58
variable attributes
  adding, 66
  retrieving, 59
variable count, 60
variable format, 60–61
variable label, 61
variable names, 62
Version, 75

weight variable, 62

XML workspace, 16, 76
XPath expressions, 16, 77