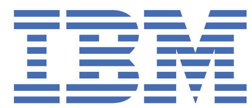
Writing IBM SPSS Statistics Extension Commands



Note

Before using this information and the product it supports, read the information in "Notices" on page 27.

Contents

Chapter 1. Introduction to Extension Commands....... 1

Chapter 2. XML Specification of the Extension Command Syntax....... 3

Chapter 3. Implementation Code....... 7

Chapter 4. Adding help for an extension command....... 11

Chapter 5. Deploying an Extension Command....... 15

Chapter 6. Extension Schema Element Reference....... 17

- Command element....... 17
- Parameter element....... 18
  - EnumValue element....... 22
- Subcommand element....... 22
- Working with Arbitrary Tokens....... 24
- Examples....... 24

Notices....... 27

- Trademarks....... 28

Chapter 1. Introduction to Extension Commands

Extension commands wrap programs that are written in the Python programming language, R, or Java in custom IBM® SPSS® Statistics command syntax. You design the syntax for your extension command based on the parameters that are required by the underlying Python, R, or Java code. When syntax for your command is run, the values that are specified in the syntax are passed to the underlying code. Extension commands require the SPSS Statistics Integration Plug-in for the language in which the command is implemented (Python, R, or Java).

The following figure provides an overview of the extension command mechanism. Briefly, command syntax for an extension command is submitted by a user and is parsed by SPSS Statistics, based on an XML representation of the syntax for the extension command. The parser calls the implementation code (written in Python, R, or Java) to process the parsed syntax and perform the requested actions, which might include generating tabular or graphical output.

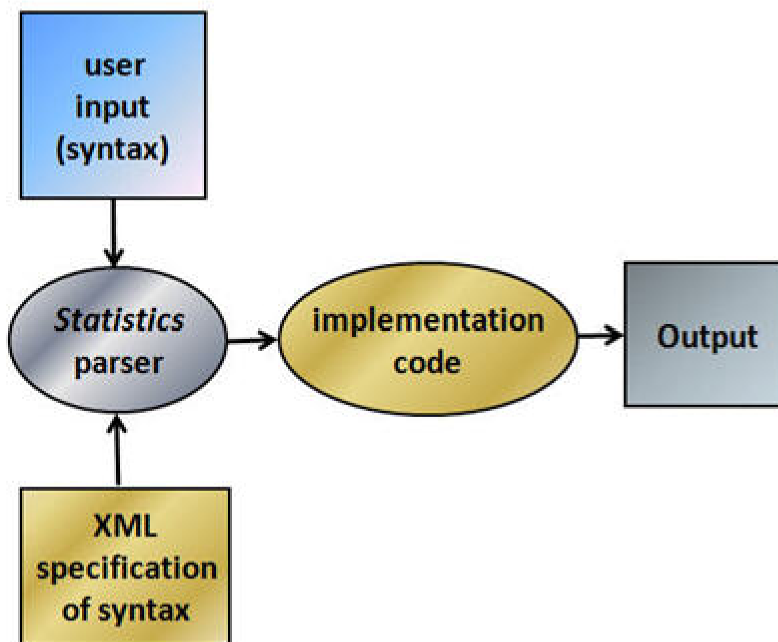


Figure 1. Overview of the extension command mechanism

As an author of an extension command, you are responsible for creating the XML specification of the syntax for the command and the implementation code. If you plan to share the extension command with other users, then you probably want to include documentation for the command. Information on creating the XML specification, the implementation code, and the documentation is provided in the sections that follow.

You can also create a custom dialog that generates the command syntax for your extension command so that users who do not typically use command syntax can easily run your extension command. Extension commands and custom dialogs can be packaged in extension bundles, which can be easily installed by users. For an example of creating a custom dialog for an extension command, which is implemented in R, see the tutorial "Working with R" in the SPSS Statistics Help system.

How you approach the task of creating an extension command depends somewhat on what you are implementing. For example, if you are wrapping an R package, then you might start with the implementation code, and then design the syntax for the extension command based on the parameters that you are supporting in the underlying R functions. However, if you are developing all of the implementation code yourself then you might start at the higher-level of the syntax or custom dialog design. If you do plan to create a custom dialog for your extension command, then you might want to

design the custom dialog before you design the syntax because it is possible to design syntax that is difficult to generate from a custom dialog.

Integration Plug-ins

To develop the implementation code, you need the Integration Plug-in for the programming language in which the extension command is implemented. Information on how to get the Plug-ins for Python and R is available from Core System > Frequently Asked Questions > How to Get Integration Plug-Ins in the SPSS Statistics Help system.

- For version 22 and higher, the Integration Plug-in for Python is installed by default with SPSS Statistics and SPSS Statistics Server, as part of IBM SPSS Statistics - Essentials for Python.
- The Integration Plug-in for Java™ (requires SPSS Statistics version 21 or higher) is installed with SPSS Statistics and SPSS Statistics Server and requires no separate installation or configuration.

Chapter 2. XML Specification of the Extension Command Syntax

To create an extension command, you must specify the syntax for the command. The syntax consists of the subcommands, keywords, and keyword values that define the command. Syntax for extension commands is specified with XML. As an example, consider a Python extension command that gets an Excel file from a user-specified URL and opens a specified sheet from the Excel file in IBM SPSS Statistics.

The inputs to this example command are as follows:

- URL: Required parameter that specifies the URL of the Excel file.
- FILETYPE: Optional parameter that specifies the file type. Possible values are XLS or XLSX, and the default is XLS.
- SHEETNUMBER: Optional parameter that specifies the number of the sheet to open in SPSS Statistics. The default is sheet 1.
- READNAMES: Optional parameter that specifies whether to use values in the first row of the sheet as variable names in the resulting SPSS Statistics dataset. Possible values are "ON" and "OFF", where "ON" specifies to use the values in the first row and is the default.

Before you create the XML representation of the syntax, it's always a good idea to construct a syntax chart for your command. The syntax chart specifies the command name, keywords, and keyword values that are available with the command. In that regard, the documentation for every native SPSS Statistics command includes a syntax chart, which provides a quick reference of the specifications for the command.

The syntax chart for the example command is as follows:

```
MYORG GETURL EXCEL URL = "URL specification"
[/OPTIONS ]
   [FILETYPE = {XLS**}]
               {XLSX }
   [SHEETNUMBER = {1**    }]
                  {integer}
   [READNAMES = {ON**}]
                {OFF }
```

- The name of the command is `MYORG GETURL EXCEL`.
- The command contains a single subcommand that is named `OPTIONS` that contains the optional inputs. Subcommands are preceded by a forward slash.
- The `OPTIONS` subcommand includes the following keywords: `FILETYPE`, `SHEETNUMBER`, and `READNAMES`. Curly brackets in syntax charts indicate a set of mutually exclusive choices for a keyword. For example, the `FILETYPE` keyword can have the value XLS or XLSX. A double asterisk in a syntax chart indicates the default value for a keyword. Square brackets in syntax charts indicate optional elements of the syntax.

Notation conventions for syntax charts are described in the Commands topic, under Reference > Command Syntax Reference > Universals, in the SPSS Statistics Help system.

The XML representation of the syntax chart for the `MYORG GETURL EXCEL` extension command is as follows:

```
<Command xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www-01.ibm.com/software/analytics/
spss/xml/extension-1.0.xsd"
Name="MYORG GETURL EXCEL" Language="Python">
    <Subcommand Name="" IsArbitrary="False" Occurrence="Optional">
```

```
        <Parameter Name="URL" ParameterType="TokenList"/>
    </Subcommand>
    <Subcommand Name="OPTIONS" Occurrence="Optional">
        <Parameter Name="FILETYPE" ParameterType="Keyword"/>
        <Parameter Name="SHEETNUMBER" ParameterType="Integer"/>
        <Parameter Name="READNAMES" ParameterType="Keyword"/>
    </Subcommand>
</Command>
```

- The top-level element, `Command`, names the command. Subcommands are children of this element. The `Name` attribute is required and specifies the command name. The command name can consist of up to three words that are separated by spaces, as in `MY EXTENSION COMMAND`, and is not case-sensitive. Command names are restricted to 7-bit ascii characters. The `Language` attribute is optional and specifies the implementation language. The default is the Python programming language (Python 2). The choices for `Language` are Python, R, and Java. For Python code that is implemented in Python 3, specify `Language="Python" LanguageVersion="3"`. Support for Python 3 requires IBM SPSS Statistics release 24 or higher.
- Subcommands are specified with the `Subcommand` element, which is a child of the `Command` element. In this example, the first `Subcommand` element has an empty string for a name. Because it doesn't have a name, it is referred to as the *anonymous* subcommand. A command can have only a single anonymous subcommand. The anonymous subcommand is typically used to specify global keywords for the command. In this example, the anonymous subcommand contains the single keyword `URL` that specifies the URL of the Excel file. Many native SPSS Statistics commands, such as the `FREQUENCIES` command, contain an anonymous subcommand.
- Keywords are specified with the `Parameter` element, which is a child of the `Subcommand` element. The `Parameter` element for the `URL` keyword is specified as a TokenList, which is an arbitrary list of comma separated or blank separated values. Although the actual URL is a single string, a URL that exceeds the allowed length of a string literal needs to be broken up into a set of strings.

A complete specification of the XML schema for defining extension commands is provided in the SPSS Statistics Help system under Reference > Extension Schema. A copy of the extension schema, `extension-1.0.xsd`, is installed with SPSS Statistics, at the root of the installation directory. Numerous XML specification files for extension commands are installed with IBM SPSS Statistics - Essentials for Python and IBM SPSS Statistics - Essentials for R and might be useful as examples. The files are in the location where extension commands are installed on your computer. To view the location, run the `SHOW EXTPATHS` syntax command from within SPSS Statistics. The output displays a list of locations under the heading "Locations for extension commands". The files are installed to the first writable location in the list.

Naming conventions and name conflicts

- Extension commands take priority over built-in command names. For example, if you create an extension command that is named `MEANS`, the built-in `MEANS` command is replaced by your extension. Likewise, if an abbreviation is used for a built-in command and the abbreviation matches the name of an extension command, the extension command is used. Abbreviations are not supported for extension commands.
- To reduce the risk of naming conflicts with built-in commands or commands that are created by other users, you might want to use two- or three-word command names, where the first word specifies your organization.
- There are no naming requirements for the file that contains the XML specification of the syntax. As with choosing the name of the extension command, take care when you choose a name to avoid conflicting XML file names. A useful convention is to use the same name as the Python module, R source file, or Java class file (or JAR file) that implements the command.
- The installed extension commands are read when SPSS Statistics is launched, but they can also be defined during a session with the `EXTENSION` command.

Color coding and auto-completion in the syntax editor

The XML syntax specification file contains all of the information that is needed to provide color coding and auto-completion for your extension command in the SPSS Statistics Syntax Editor. Both color coding and auto-completion are available after the extension command is installed.

Chapter 3. Implementation Code

The implementation code receives the parsed syntax and then performs the requested actions. The code (whether written in Python, R, or Java) must contain a function that is named Run with a single argument that accepts the parsed syntax.

The contents of the Python module for the example `MYORG GETURL EXCEL` extension command, which was discussed in the preceding section, is shown in the code sample that follows. The code includes `import` statements for the `spssaux` and `extension` Python modules that are used by the code and that are installed with IBM SPSS Statistics - Essentials for Python. The Python module itself must be named `MYORG_GETURL_EXCEL.py`, as required by the naming conventions that are described in what follows.

The module contains two functions: Run and `geturlexcel`. The Run function takes the parsed syntax, validates it, extracts the values of the keywords, and then calls `geturlexcel` to implement the requested actions. Separating the implementation code into a Run function that processes the submitted syntax and a main implementation function that performs the requested actions is a general feature of all extension commands that are created by IBM SPSS.

```
import spssaux
from extension import Template, Syntax, processcmd

def Run(args):
    oobj = Syntax([
        Template("URL", subc="",  ktype="literal", var="url", islist=True),
        Template("FILETYPE", subc="OPTIONS", ktype="str", var="filetype",
vallist=["xls","xlsx"]),
        Template("SHEETNUMBER", subc="OPTIONS", ktype="int", var="sheetnumber"),
        Template("READNAMES", subc="OPTIONS", ktype="str", var="readnames",
vallist=["on","off"])])
    args = args[args.keys()[0]]
    processcmd(oobj, args, geturlexcel)

def geturlexcel(url, filetype="xls", sheetnumber=1, readnames="ON"):
    kwargs = {}
    url = "".join(url)
    kwargs["filetype"] = filetype
    kwargs["sheetid"] = sheetnumber
    kwargs["readnames"] = readnames
    spssaux.openDataFileFromUrl(url, **kwargs)
```

The Run function uses the `Template` class, the `Syntax` class, and the `processcmd` function. These functions are designed to be used together and greatly simplify the task of working with the parsed syntax, which is contained in the *args* parameter to the Run function.

- The `Template` class specifies how to process a particular keyword in the syntax for an extension command. Each keyword of each subcommand must have an associated instance of the `Template` class.
  - The first argument to the `Template` class constructor is the name of the keyword.
  - The argument *subc* specifies the name of the subcommand that contains the keyword. If the keyword belongs to the anonymous subcommand, the argument *subc* can be omitted or set to the empty string as shown here for the URL keyword.
  - 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 argument *var* specifies the name of the argument to the implementation function (`geturlexcel` in this example) that receives the value that is specified for the keyword.
  - The argument *vallist*, that is used for `FILETYPE` and `READNAMES`, specifies the list of allowed values for the keyword. For keywords that are specified as `ktype="str"` in the `Template` constructor, submitted values of the keyword are converted to lowercase before validation, so the list of allowed values is specified in lowercase.

- The `Syntax` class validates the syntax that is specified by the `Template` objects. The class is instantiated with a sequence of one or more `Template` objects as shown in this example.
- The `processcmd` function extracts the values of the keywords from the submitted syntax and then calls the implementation function to carry out the requested actions.
  – The first argument to the `processcmd` function is the `Syntax` object for the command.
  – For Python, the parsed syntax is passed to the Run function in a complex nested Python dictionary that has a single top-level entry. The second argument to `processcmd` is this top-level entry, which is given by the expression `args[args.keys()[0]]`.
  – The third argument to `processcmd` is the name of the implementation function. The values of the keywords that are specified by the `Template` objects are passed to the implementation function as a set of keyword arguments. In this example, the implementation function `geturlexcel` is called with the following signature:

```
geturlexcel(URL=<URL>,filetype=<FILETYPE>,sheetnumber=<SHEETNUMBER>,readnames=<READNAMES>)
```

  where <URL> is the value that is specified for the URL keyword, and likewise for the other keywords.

  Note:

  - If a Python exception is raised in the implementation function, the Python traceback is suppressed, but the error message is displayed. To display tracebacks, set the *SPSS_EXTENSIONS_RAISE* environment variable to "true".
  - If the signature of the implementation function does not have a default value for a parameter, then an error is raised if the submitted syntax does not include a value for the parameter.

The `geturlexcel` function receives the values that were specified for the keywords in the submitted syntax, and then opens the requested Excel file. The function calls the `openDataFileFromUrl` function from the `spssaux` module, which is installed with IBM SPSS Statistics - Essentials for Python, to open the file.

Help for the `Template` class, the `Syntax` class, and the `processcmd` function is provided with the `extension` module, which is installed with IBM SPSS Statistics - Essentials for Python. You can access the help from `help(extension)` after you import the `extension` module.

Although the preceding example is for Python, the same approach for creating the implementation code can be used for extension commands in R or Java.

- For an example of creating an extension command in R, see the tutorial "Working with R" in the IBM SPSS Statistics Help system. Help for the `spsspkg.Template`, `spsspkg.Syntax`, and `spsspkg.processcmd` functions in R (the equivalents of the Python functions used in the previous example) is available under Integration Plug-in for R Help > R Integration Package for IBM SPSS Statistics, in the Help system.
- For Java, see the topic "Creating IBM SPSS Statistics extension commands in Java" under Integration Plug-in for Java User Guide > Getting started with the Integration Plug-in for Java, in the SPSS Statistics Help system. Help for the `Template`, `Syntax`, and `processcmd` functions in Java is included in the help for the `Extension` class under Integration Plug-in for Java API Reference, in the Help system.
- Numerous implementation code files for extension commands are installed with IBM SPSS Statistics - Essentials for Python and IBM SPSS Statistics - Essentials for R, and might be useful as examples. The files are in the location where extension commands are installed on your computer. To view the location, run the `SHOW EXTPATHS` syntax command from within SPSS Statistics. The output displays a list of locations under the heading "Locations for extension commands". The files are installed to the first writable location in the list.

Naming conventions

The Python module, R source file, or Java class file (or JAR file) that contains the Run function for an extension command must adhere to the following naming conventions:

- **Python.** The Run function must reside in a Python module file with the same name as the command--for instance, in the Python module file *MYCOMMAND.py* for an extension command that is named `MYCOMMAND`. The name of the Python module file must be in upper case, although the command name itself is case insensitive. For multi-word command names, replace the spaces between words with underscores. For example, for an extension command with the name `MY COMMAND`, the associated Python module is *MY_COMMAND.py*.
- **R.** The Run function must reside in an R source file or R package with the same name as the command--for instance, in a source file named *MYRFUNC.R* for an extension command that is named `MYRFUNC`. The name of the R source file or package must be in upper case, although the command name itself is case insensitive. For multi-word command names, replace the spaces between words with underscores for R source files and periods for R packages. For example, for an extension command with the name `MY RFUNC`, the associated R source file is *MY_RFUNC.R*, whereas an R package that implements the command is named *MY.RFUNC.R*. The source file or package should include any `library` function calls required to load R functions that are used by the code.
- **Java.** The Run function must reside in a Java class file or JAR file with the same name as the command--for instance, in a class file named *MYCOMMAND.class* for an extension command that is named `MYCOMMAND`. The name of the Java class file or JAR file must be in upper case, although the command name itself is case insensitive. For multi-word command names, replace spaces between words with underscores when constructing the name of the Java class file or JAR file. For example, for an extension command with the name `MY COMMAND`, the associated Java class file is *MY_COMMAND.class*.

Command syntax errors

Syntax errors, like not providing an integer for a parameter that is specified as `Integer`, are handled by IBM SPSS Statistics and stop the module from running. In that regard, the implementation code does not need to handle deviations from the XML specification of the syntax for the extension command.

Generating output

Generating and sending output to IBM SPSS Statistics is handled by the implementation code.

- For Python and Java, the implementation code is responsible for specifying the procedure name (associated with the extension command) that labels pivot table output in the Viewer. In other words, unlike built-in IBM SPSS Statistics procedures such as `FREQUENCIES`, there is no automatic association of the extension command name with the name that labels pivot table output from the command. For Python, the procedure name is the first argument to the `spss.StartProcedure` function that wraps the statements that generate the output. For Java, the procedure name is the first argument to the `StatsUtil.startProcedure` function that wraps the statements that generate the output.
- For R, the default name that is associated with pivot table output from an extension command is *R*. For IBM SPSS Statistics version 18 and higher, the name can be customized by wrapping the output statements in an `spsspkg.StartProcedure - spsspkg.EndProcedure` block. The procedure name is then the first argument to the `spsspkg.StartProcedure` function.

Globalization

You can globalize messages and output that is produced by the implementation code. For Python, see the topic "Localizing Output from Python Programs" under Integration Plug-in for Python Help > Python Integration Package for IBM SPSS Statistics > Introduction to Python Programs, in the SPSS Statistics Help system. For R, see the topic "Localizing Output from R" under Integration Plug-in for R Help > Using the R Integration Package for IBM SPSS Statistics, in the SPSS Statistics Help system.

Chapter 4. Adding help for an extension command

Providing help for an extension command is optional. However, if you plan to share the command with other users then you probably want to include help for it. Two approaches for including and displaying help for an extension command are presented. The help that is discussed here is independent of any help that you provide for a custom dialog that is associated with the extension command.

Both approaches use the convention of displaying the extension command help (and doing nothing else) when the submitted syntax contains the HELP subcommand. This convention is used by all extension commands that are installed with IBM SPSS Statistics - Essentials for Python and IBM SPSS Statistics - Essentials for R. To implement this convention, the XML specification of the extension command syntax must contain the HELP subcommand.

The modified XML, that includes a HELP subcommand, for the example `MYORG GETURL EXCEL` command is as follows:

```
<Command xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www-01.ibm.com/software/analytics/
spss/xml/extension-1.0.xsd"
Name="MYORG GETURL EXCEL" Language="Python">
    <Subcommand Name="" IsArbitrary="False" Occurrence="Optional">
        <Parameter Name="URL" ParameterType="TokenList"/>
    </Subcommand>
    <Subcommand Name="OPTIONS" Occurrence="Optional">
        <Parameter Name="FILETYPE" ParameterType="Keyword"/>
        <Parameter Name="SHEETNUMBER" ParameterType="Integer"/>
        <Parameter Name="READNAMES" ParameterType="Keyword"/>
    </Subcommand>
    <Subcommand Name="HELP" Occurrence="Optional"/>
</Command>
```

HTML help

You can create an HTML help file for your extension command and include it with the extension bundle for the command. When the extension bundle is installed, the HTML file is installed along with the other files in the bundle. In the implementation code, you can include a simple function that finds and opens the HTML file in the default browser. By convention, the name of the function is `helper` and is defined as follows for Python:

```
def helper():
    import webbrowser, os.path
    path = os.path.splitext(__file__)[0]
    helpspec = "file://" + path + os.path.sep + "markdown.html"
    browser = webbrowser.get()
    if not browser.open_new(helpspec):
        print("Help file not found:" + helpspec)

try:
    from extension import helper
except:
    pass
```

- The `helper` function assumes that the name of the extension bundle (as specified in the Name field on the Create Extension Bundle dialog, or Extension Properties dialog if you created the extension bundle from the Custom Dialog Builder for Extensions in version 24 or higher) is the same as the name of the implementation code file (except that the bundle name might have spaces where the code file has underscores). Using the same name for the extension bundle and the code file is the recommended convention. When the extension bundle is installed, the help file (and any other auxiliary files) is installed in a folder with the same name as the extension bundle, and that folder is directly

under the folder where the implementation code is installed. In this example, the extension bundle is named `MYORG GETURL EXCEL` and the code file is named `MYORG_GETURL_EXCEL.py`, so the help file is installed in a folder that is named `MYORG_GETURL_EXCEL`, which is directly under the folder that contains the code file.

- The `helper` function in this example assumes that the name of the HTML file is `markdown.html`. Given the relationship between the name of the code file and the location of the help file, it is simple to locate the help file, as shown in the example code. In that regard, most extension commands that are installed with IBM SPSS Statistics - Essentials for Python and IBM SPSS Statistics - Essentials for R use the convention of `markdown.html` for the name of the help file and the convention of the equality of the name of the extension bundle and the name of the code file.
- For IBM SPSS Statistics version 23 and higher, the `helper` function is included with IBM SPSS Statistics - Essentials for Python, and is part of the `extension` module. The `try` block in this code segment attempts to import the `helper` function from the `extension` module. If the function is not available in the local copy of the `extension` module, then the version of the function that is included with the implementation code is used.

You can incorporate the `helper` function into the Run function of the implementation code by calling the `helper` function when the HELP subcommand is specified, as shown in the following code sample:

```
    if args.has_key("HELP"):
        helper()
    else:
        processcmd(oobj, args, geturlexcel)
```

For R, the `helper` function is shown in the following code segment:

```
helper = function(cmdname) {
    fn = gsub(" ", "_", cmdname, fixed=TRUE)
    thefile = Find(file.exists, file.path(.libPaths(), fn, "markdown.html"))
    if (is.null(thefile)) {
        print("Help file not found")
    } else {
        browseURL(paste("file://", thefile, sep=""))
    }
}

if (exists("spsspkg.helper")) {
assign("helper", spsspkg.helper)
}
```

- For R, the `helper` function requires the name of the extension command. Although not shown here, the name is given by `args[[1]]`, where *args* is the argument that is passed to the Run function.
- As with Python, the `helper` function in R assumes that the name of the implementation code file is the same as the name of the extension bundle. The code also uses the convention of `markdown.html` for the name of the help file.
- As with Python, you can incorporate the `helper` function into the Run function by calling the `helper` function when the HELP subcommand is specified. In R, you can check for the HELP subcommand as follows:

```
"HELP" %in% attr(args,"names")
```

- For IBM SPSS Statistics version 23 and higher, the `helper` function is included with IBM SPSS Statistics - Essentials for R and is named `spsspkg.helper`. The `if` block that follows the `helper` function attempts to assign the name `helper` to this `spsspkg.helper` function. If the `spsspkg.helper` function is not available, the version of the `helper` function that is included with the implementation code is used.

Note:

- For IBM SPSS Statistics version 23 and higher, the help is displayed by pressing the F1 key in a Syntax Editor window when the cursor is positioned within the associated extension command. The help is also displayed if the submitted syntax contains the HELP subcommand.

- If you have a style sheet for your HTML file, you can include it in the extension bundle with the HTML file. When the extension bundle is installed, the style sheet is installed under the same folder as the HTML file. For version 23 and higher, all extension commands that are installed with IBM SPSS Statistics - Essentials for Python and IBM SPSS Statistics - Essentials for R have HTML help and an associated style sheet that you might want to use. The style sheet is also available from the IBM SPSS Predictive Analytics Community (Docs>SPSS Statistics>Programmability>Extensions, Tools and Utilities for SPSS Statistics>Utilities). The name of the style sheet file is `extsyntax.css`.
- The `helper` function cannot be used in distributed mode, and raises an error message if it is used in distributed mode.

Plain text help

You can embed the help in a string variable that is defined in the implementation code file and then display the string when the HELP subcommand is specified. With this approach, the string is displayed in a Log item in the SPSS Statistics Viewer, and works in distributed mode.

For Python, you can embed the help text in a triple-quoted string to preserve formatting. An example of help text that just contains the syntax chart for the `MYORG GETURL EXCEL` command is as follows:

```
helptext = """MYORG GETURL EXCEL URL = "URL specification"
[/OPTIONS ]
   [FILETYPE = {XLS**}]
               {XLSX }
   [SHEETNUMBER = {1**    }]
                  {integer}
   [READNAMES = {ON**}]
                {OFF }
[/HELP ]
"""
```

An example of code that is added to the Run function to conditionally display the help text when the HELP subcommand is specified is as follows:

```
    if args.has_key("HELP"):
        print helptext
    else:
        processcmd(oobj, args, geturlexcel)
```

For R, you include the help text in a string variable and then conditionally display it with the `writeLines` function.

Help contents

Help for an extension command should contain the following content:

- A brief description of the extension command.
- A syntax chart that specifies the command name, subcommands, keywords, keyword values, and default values of keywords.
- A simple example of the syntax.
- Descriptions for each of the subcommands, keywords, and keyword values.

All of the extension commands that are installed by IBM SPSS Statistics - Essentials for Python and IBM SPSS Statistics - Essentials for R include help and might be useful as examples. For release 22 and earlier, the help is in plain text format and is contained in the implementation code file for the extension command. For release 23 and higher, the help is in HTML format and is contained in the file `markdown.html`. For a particular extension command, the file `markdown.html` is in a folder with the same name as the command and directly under the folder where the implementation code is installed. If you have release 22 or earlier, you can obtain the HTML format help by installing the latest version of the associated extension command from the IBM SPSS Predictive Analytics community.

Chapter 5. Deploying an Extension Command

You can easily deploy an extension command (with or without an associated custom dialog) on your computer, or share it so that it can be deployed by other users. To deploy an extension command, you first create an extension bundle that includes the XML specification file and the implementation code. You or your users then install the extension bundle from within IBM SPSS Statistics. For version 24 and higher, install the extension bundle from Extensions> Install Local Extension Bundle. For versions before 24, install the extension bundle from Utilities > Extension Bundles > Install Local Extension Bundle.

- The recommended convention is to use the same name for the extension bundle as the name of the extension command. For example, for an extension command that is named `MYORG GETURL EXCEL`, the value of the **Name** field in the Create Extension Bundle dialog (or Extension Properties dialog if you created the extension bundle from the Custom Dialog Builder for Extensions in version 24 or higher) is also MYORG GETURL EXCEL.
- If you create a custom dialog for your extension command, then be sure that the custom dialog specification file (`.cfe` or `.spd`) is included in the extension bundle.
- If you create an HTML help file for your extension command, then include the file in the extension bundle. If your HTML file uses a style sheet, then include the style sheet in the extension bundle.
- If the implementation code consists of multiple code files (for example, multiple Python modules), then include all of the files in the extension bundle.
- For extension commands that are implemented in R, list the names of any R packages from the CRAN package repository that are required by your code. Required R packages are listed on the Optional tab of the Create Extension Bundle dialog (or Optional tab of the Extension Properties dialog if you created the extension bundle from the Custom Dialog Builder for Extensions in version 24 or higher). When the extension bundle is installed, SPSS Statistics checks if the required R packages exist on the user's computer and attempts to download (from CRAN) and install any that are missing.
- If you are globalizing output and messages for your extension command, then include the folder that contains the translated resources. To add the folder, specify the path to the folder in the Translation Catalogues Folder field on the Optional tab of the Create Extension Bundle dialog (or Optional tab of the Extension Properties dialog if you created the extension bundle from the Custom Dialog Builder for Extensions in version 24 or higher).
- If you or your users plan to run the extension command in distributed mode, then be sure to install the extension bundle on both the client and server computers.

For version 24 and higher, help on creating and installing extension bundles is available under Core System > Extensions, in the SPSS Statistics Help system. For versions before 24, help is available under Core System > Utilities > Extension Bundles. An example of creating an extension bundle is provided in the tutorial "Working with R", in the SPSS Statistics Help system.

Chapter 6. Extension Schema Element Reference

This section provides a reference for all elements in the extension schema. Each topic lists the valid attributes for an element and its parent and child elements.

Command element

The top-level element, also known as the document or root element. The **Command** element contains a complete command syntax specification of an extension command.

Table 1. Command attributes

Attribute	Use	Description	Valid Values
Language	optional	The programming language in which the command is implemented. Defaults to Python.	**Python** **R** **Java**
Mode	optional	Specifies whether the implementation code is contained in an R source file or an R package. Only applies for Language="R". Defaults to **Source**.	**Source** **Package**
Name	**required**	The name of the command. The name can consist of up to three words (case insensitive) separated by spaces. Command names are restricted to 7-bit ASCII characters. For multi-word command names, ensure that the first word as well as the first two words do not match the name of another command. For example, do not use the name **CMD NEW** if there is a command named **CMD**. Likewise, do not use the name **MY CMD NEW** if there is a command named **MY CMD**.	*any*

XML representation

```
<xs:element name="Command" type="command-content">
   <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="Subcommand"></xs:element>
   </xs:sequence>
```

```
    <xs:attribute name="Name" use="required"></xs:attribute>
    <xs:attribute name="Language">
       <xs:simpleType>
          <xs:restriction base="xs:token">
             <xs:enumeration value="Python"></xs:enumeration>
             <xs:enumeration value="R"></xs:enumeration>
             <xs:enumeration value="Java"></xs:enumeration>
          </xs:restriction>
       </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="Mode">
       <xs:simpleType>
          <xs:restriction base="xs:token">
             <xs:enumeration value="Source"></xs:enumeration>
             <xs:enumeration value="Package"></xs:enumeration>
          </xs:restriction>
       </xs:simpleType>
    </xs:attribute>
</xs:element>
```

Child elements

"Subcommand element" on page 22

Parameter element

A parameter controls a specific piece of a command's functionality. There are many parameter types; the **Parameter** element is flexible enough to cover the different types of functionality provided by parameters. Subcommands contain zero or more parameters.

Table 2. Parameter attributes

Attribute	Use	Description	Valid Values
Name	**required**	The name of the parameter. Parameter names are restricted to 7-bit ASCII characters and must start with a letter or one of the characters @, #, or $. Subsequent characters can be any combination of letters, numbers, non-punctuation characters, and a period (.).	*any*
ParameterType	**required**		**DatasetName**. Specifies a dataset name. The value will be checked for syntax correctness (the same rules as for variable names) but not existence. The case is preserved when passed to the **Run** function.

Table 2. Parameter attributes (continued)			
Attribute	**Use**	**Description**	**Valid Values**
			Integer. A number with no fractional part after conversion. Optionally, you can specify a set of allowed keyword values for an **Integer** parameter, using **EnumValue** elements.
			IntegerList. A blank or comma separated list of **Integer** types.
			Keyword. Specifies a value that adheres to the same rules as the **Name** attribute of a **Parameter** element. The value is passed in uppercase to the **Run** function. You can specify the set of allowed values using **EnumValue** elements. **Keyword** type parameters must be assigned values. To specify a keyword without an associated value, use the **LeadingToken** type.
			KeywordList. Specifies a comma or blank separated list of values that adhere to the same rules as the **Name** attribute of a **Parameter** element. To specify a list of values not bound by these rules, use the **TokenList** type. You can specify the set of allowed values using **EnumValue** elements.

Table 2. Parameter attributes (continued)			
Attribute	**Use**	**Description**	**Valid Values**
			LeadingToken. Specifies a parameter that has a name (given by the **Name** attribute of the **Parameter** element) but no associated value. The name is passed in uppercase to the **Run** function.
			Number. A number, possibly in scientific notation using **e** or **E**. Optionally, you can specify a set of allowed keyword values for a **Number** parameter, using **EnumValue** elements.
			NumberList. A blank or comma separated list of **Number** types.
			QuotedString. A string enclosed in single or double quotes. The case is preserved when passed to the **Run** function. Optionally, you can specify a set of allowed keyword values (unquoted) for a **QuotedString** parameter, using **EnumValue** elements.
			TokenList. Specifies a comma or blank separated list of values. Case is preserved when values are passed to the **Run** function. The **TokenList** type is similar to the **KeywordList** type but **TokenList** values are not bound by the rules required of **KeywordList** values.

Attribute	Use	Description	Valid Values
			`VariableName`. Specifies a variable name. The value will be checked for syntax correctness (see the rules for variable names in the Command Syntax Reference) but not existence. The case is preserved when passed to the **Run** function.
			`VariableNameList`. Specifies a list of variable names. Each name in the list will be checked for syntax correctness but not existence. Case is preserved when values are passed to the **Run** function. The **TO** and **ALL** keywords are not supported.
			`InputFile`. A file specification for an input file. The specified file must exist. The case is preserved when passed to the **Run** function.
			`OutputFile`. A file specification for an output file. The specified directory path must exist and either the specified file does not exist, or if it exists it must be writable. The case is preserved when passed to the **Run** function.

XML representation

```
<xs:element name="Parameter" type="parameter-content">
   <xs:sequence maxOccurs="unbounded" minOccurs="0">
      <xs:element name="EnumValue"></xs:element>
   </xs:sequence>
   <xs:attribute name="Name" use="required"></xs:attribute>
   <xs:attribute name="ParameterType" use="required">
      <xs:simpleType>
         <xs:restriction base="xs:token">
            <xs:enumeration value="DatasetName"></xs:enumeration>
            <xs:enumeration value="Integer"></xs:enumeration>
```

```
            <xs:enumeration value="IntegerList"></xs:enumeration>
            <xs:enumeration value="Keyword"></xs:enumeration>
            <xs:enumeration value="KeywordList"></xs:enumeration>
            <xs:enumeration value="LeadingToken"></xs:enumeration>
            <xs:enumeration value="Number"></xs:enumeration>
            <xs:enumeration value="NumberList"></xs:enumeration>
            <xs:enumeration value="QuotedString"></xs:enumeration>
            <xs:enumeration value="TokenList"></xs:enumeration>
            <xs:enumeration value="VariableName"></xs:enumeration>
            <xs:enumeration value="VariableNameList"></xs:enumeration>
            <xs:enumeration value="InputFile"></xs:enumeration>
            <xs:enumeration value="OutputFile"></xs:enumeration>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
</xs:element>
```

Parent elements

"Subcommand element" on page 22

Child elements

"EnumValue element" on page 22

EnumValue element

EnumValue is used to enumerate a set of allowed values. **EnumValue** elements are ignored except for the **Keyword**, **KeywordList**, **Number**, **Integer**, and **QuotedString** parameters. When used with **Keyword** or **KeywordList** parameters, the specified **EnumValue** elements represent the complete set of allowed values. When used with **Number**, **Integer**, or **QuotedString** parameters, the specified **EnumValue** elements represent a set of valid keywords in addition to the specified type. For example, the value **AUTO** can be specified as an allowed keyword value for an **Integer** parameter. The parameter can then be specified as an integer or the unquoted string **AUTO**.

Table 3. EnumValue attributes

Attribute	Use	Description	Valid Values
Name	**required**	The enumerated value. Case is ignored.	*any*

XML representation

```
<xs:element name="EnumValue">
   <xs:attribute name="Name" use="required"></xs:attribute>
</xs:element>
```

Parent elements

"Parameter element" on page 18

Subcommand element

Subcommands divide a command's functionality into distinct groups. Typical subcommands include **SAVE** for specifying variables to be saved to the active dataset, **PRINT** for specifying tabular output, and **PLOT** for specifying chart output. A subcommand can only be specified once per command. The name of a subcommand must be preceded by a forward slash when specified in command syntax.

Table 4. Subcommand attributes			
Attribute	**Use**	**Description**	**Valid Values**
IsArbitrary	optional	Allows arbitrary tokens on the subcommand. This is useful, for example, for specifying variable lists and model effect lists.	**True** **False** **Yes**. Equivalent to **True**. **No**. Equivalent to **False**.
Name	**required**	The name of the subcommand. Subcommand names are restricted to 7-bit ASCII characters and start with a letter or one of the characters @, #, or $. Subsequent characters can be any combination of letters, numbers, non-punctuation characters, and a period (.). Use Name="" to specify the anonymous subcommand.	*any*
Occurrence	optional	Specifies whether the subcommand must be included in a syntax job for the command to run.	**Required** **Optional**

XML representation

```
<xs:element name="Subcommand" type="subcommand-content">
   <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="Parameter"></xs:element>
   </xs:sequence>
   <xs:attribute name="Name" use="required"></xs:attribute>
   <xs:attribute name="Occurrence">
      <xs:simpleType>
         <xs:restriction base="xs:token">
            <xs:enumeration value="Required"></xs:enumeration>
            <xs:enumeration value="Optional"></xs:enumeration>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="IsArbitrary">
      <xs:simpleType>
         <xs:restriction base="xs:token">
            <xs:enumeration value="True"></xs:enumeration>
            <xs:enumeration value="False"></xs:enumeration>
            <xs:enumeration value="Yes"></xs:enumeration>
            <xs:enumeration value="No"></xs:enumeration>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
</xs:element>
```

Parent elements

"Command element" on page 17

Child elements

"Parameter element" on page 18

Working with Arbitrary Tokens

TokenList parameter types and Subcommand elements with IsArbitrary=True consist of arbitrary tokens, such as character strings, literals (strings delimited by single or double quotes), and operators like '>'. Typical scenarios where arbitrary tokens are required include variable lists that may contain BY or WITH specifications and model effects lists. For example, a subcommand that allows arbitrary tokens for specifying model interaction effects might be given as:

```
/MODEL var1*var2(var3)
```

When parsed, this results in the following list of six tokens passed to the Run function of the implementation code.

```
['var1','*','var2','(','var3',')']
```

Likewise, an anonymous subcommand that allows arbitrary tokens to specify variables with optional BY or WITH keywords might be specified as:

```
DepVar BY A B WITH X Y
```

When parsed, this results in the following list of seven tokens passed to the Run function of the implementation code.

```
['DepVar','BY','A','B','WITH','X','Y']
```

When working with arbitrary tokens you'll want to test the various forms of input that your implementation function will need to handle, since the tokens that result from the specified input may not always be what you expect. For example, a TokenList parameter named TOKENS might be specified as:

```
TOKENS = 1a 2b
```

When parsed, this results in the following list of four tokens passed to the Run function of the implementation code.

```
['1','a','2','b']
```

The result reflects the standard way that IBM SPSS Statistics tokenizes command syntax. In particular, when a set of digits precedes a set of characters, the digits are treated as a separate token. You can force a set of characters to be passed as a single token by enclosing them in quotes. For example, specifying TOKENS = '1a' '2b' results in the token list ['1a','2b']. The same applies if you need to specify multi-word phrases and have each phrase passed as a single token. For example, TOKENS = 'two words' results in the single token 'two words'.

Examples

Using the Keyword Type

The Keyword type is used to specify a parameter that takes a single value. As an example, consider an OPTIONS subcommand with a parameter for controlling missing values, and represented in the syntax diagram as:

```
/OPTIONS MISSING={PAIRWISE}
                 {LISTWISE}
```

The specification of MISSING is best handled with a Keyword type parameter. The XML syntax specification for the OPTIONS subcommand is:

```
<Subcommand Name="OPTIONS">
   <Parameter Name="MISSING" ParameterType="Keyword">
      <EnumValue Name="PAIRWISE"/>
      <EnumValue Name="LISTWISE"/>
   </Parameter>
</Subcommand>
```

An example of command syntax containing the OPTIONS subcommand is:

```
/OPTIONS MISSING=LISTWISE
```

The Keyword type always requires the parameter name, followed by an equals sign, followed by a single value.

Using the KeywordList Type

The KeywordList type is used to specify a parameter that can take on multiple values. As an example, consider an OPTIONS subcommand with a parameter for specifying one or more file types from a fixed set, and represented in the syntax diagram as:

```
/OPTIONS FILETYPES=[SAV SAS STATA]
```

The specification of FILETYPES is best handled with a KeywordList type parameter. The XML syntax specification for the OPTIONS subcommand is:

```
<Subcommand Name="OPTIONS">
   <Parameter Name="FILETYPES" ParameterType="KeywordList">
      <EnumValue Name="SAV"/>
      <EnumValue Name="SAS"/>
      <EnumValue Name="STATA"/>
   </Parameter>
</Subcommand>
```

An example of command syntax containing the OPTIONS subcommand is:

```
/OPTIONS FILETYPES=SAV SAS
```

The KeywordList type always requires the parameter name, followed by an equals sign, followed by one or more values.

Using the LeadingToken Type

The LeadingToken type is used to specify a parameter that has a name but no associated value. As an example, consider a PLOT subcommand for specifying types of plots to include in output, and represented in the syntax diagram as:

```
/PLOT OBSERVED FORECAST FIT
```

The specification of PLOT is best handled with a set of LeadingToken type parameters. The XML syntax specification for the PLOT subcommand is:

```
<Subcommand Name="PLOT">
   <Parameter Name="OBSERVED" ParameterType="LeadingToken"/>
   <Parameter Name="FORECAST" ParameterType="LeadingToken"/>
   <Parameter Name="FIT" ParameterType="LeadingToken"/>
</Subcommand>
```

An example of command syntax containing the PLOT subcommand is:

```
/PLOT OBSERVED FIT
```

Using the TokenList Type

The TokenList type is used to specify a parameter that can take on multiple values. It is similar to the KeywordList type but TokenList values are not bound by the rules required of KeywordList values. As an example, consider a MODEL subcommand with a parameter for specifying model interaction effects, and represented in the syntax diagram as:

```
/MODEL EFFECTS=effect-list
```

The specification of the effects list is handled with a TokenList type parameter. The XML syntax specification for the MODEL subcommand is:

```
<Subcommand Name="MODEL">
   <Parameter Name="EFFECTS" ParameterType="TokenList"/>
</Subcommand>
```

An example of command syntax containing the `MODEL` subcommand is:

```
/MODEL EFFECTS=A*B C(D)
```

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the GNU GENERAL PUBLIC LICENSE Version 2.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBMproducts. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.