Java Plug-in User Guide for IBM SPSS Statistics

Note

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

Product Information

This edition applies to version 28, release 0, modification 0 of IBM® SPSS® Statistics and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. Getting started with the Integration Plug-in for Java

The IBM SPSS Statistics - Integration Plug-in for Java™ enables an application developer to create Java applications that can invoke and control the IBM SPSS Statistics processor, or to implement extension commands in Java that can then be run from within IBM SPSS Statistics. Extension commands are IBM SPSS Statistics commands that are implemented in an external language (Python, R or Java) and allow users who are proficient in that language to share external functions with users of standard IBM SPSS Statistics command syntax.

With the IBM SPSS Statistics - Integration Plug-in for Java, you can do the following:

- Execute IBM SPSS Statistics command syntax.
- Read case data from the active dataset.
- Get information about data in the active dataset.
- Add new variables and append cases to the active dataset.
- Get output results from syntax commands.
- Create custom output in the form of pivot tables and text blocks.

The IBM SPSS Statistics - Integration Plug-in for Java is installed with IBM SPSS Statistics and IBM SPSS Statistics Server and requires no separate installation or configuration. For version 28 of IBM SPSS Statistics, the IBM SPSS Statistics - Integration Plug-in for Java supports Java version 7.

The IBM SPSS Statistics - Integration Plug-in for Java is designed to work with Unicode mode. Use of the IBM SPSS Statistics - Integration Plug-in for Java with code page mode is not recommended.

Complete documentation for all of the classes and methods available with the IBM SPSS Statistics - Integration Plug-in for Java is available in the Help system under the heading IBM SPSS Statistics - Integration Plug-in for Java API Reference.

Invoking IBM SPSS Statistics from an external Java application

The interface for invoking IBM SPSS Statistics and controlling the IBM SPSS Statistics processor is provided in the JAR file *spssjavaplugin.jar*, which is installed with your IBM SPSS Statistics product. The JAR file contains the *com.ibm.statistics.plugin* package, which contains the Java classes available with the IBM SPSS Statistics - Integration Plug-in for Java. Following are the locations of *spssjavaplugin.jar* by operating system. Be sure to add *spssjavaplugin.jar* to your Java class path.

- On Windows, *spssjavaplugin.jar* is located in the IBM SPSS Statistics installation directory.
- On Linux and UNIX Server systems, *spssjavaplugin.jar* is located in the *bin* directory under the IBM SPSS Statistics installation directory.
- On macOS, *spssjavaplugin.jar* is located in the *bin* directory under the *Content* directory in the IBM SPSS Statistics application bundle.

Important: Do not move *spssjavaplugin.jar* from its installed location. The IBM SPSS Statistics - Integration Plug-in for Java assumes that *spssjavaplugin.jar* is in the installed location.

Note: For information on deploying your application on end user machines, please see Chapter 8, “Deploying an external Java application,” on page 25.

When invoked from an external Java application, the IBM SPSS Statistics processor runs without an associated instance of the IBM SPSS Statistics client. In this mode, output generated from IBM SPSS Statistics can be managed with parameters specified on the method that starts the processor or through use of the IBM SPSS Statistics Output Management System, which is invoked with the OMS command.

The following is a simple example of using the IBM SPSS Statistics - Integration Plug-in for Java to create a dataset in IBM SPSS Statistics, compute descriptive statistics and generate output. It illustrates the basic features of invoking IBM SPSS Statistics from an external Java application.

```
import com.ibm.statistics.plugin.*;

public class demo {public static void main(String[] args) {

   try {
        StatsUtil.start();
        String[] command={"OMS",
                "/DESTINATION FORMAT=HTML OUTFILE='/output/demo.html'.",
                "DATA LIST FREE /salary (F).",
                "BEGIN DATA",
                "21450",
                "30000",
                "57000",
                "END DATA.",
                "DESCRIPTIVES salary.",
                "OMSEND."};
       StatsUtil.submit(command);
       StatsUtil.stop();
   } catch (StatsException e) {
       e.printStackTrace();
   }
}

}
```

- The statement `import com.ibm.statistics.plugin.*` imports all of the classes in the *com.ibm.statistics.plugin* package.
- The `StatsUtil.start` method starts the IBM SPSS Statistics processor.
- A string array specifies IBM SPSS Statistics command syntax that creates a dataset and runs the DESCRIPTIVES procedure. The command syntax is submitted to IBM SPSS Statistics using the `StatsUtil.submit` method. Output from the DESCRIPTIVES procedure is routed to an HTML file using the OMS command.
- The `StatsUtil.stop` method stops the IBM SPSS Statistics processor and should be called to properly end an IBM SPSS Statistics session.
- The `StatsException` class is a subclass of the native Java `Exception` class, and handles exceptions that are specific to the IBM SPSS Statistics - Integration Plug-in for Java. It can be inherited to define custom exception classes for your application.

Creating IBM SPSS Statistics extension commands in Java

This topic describes aspects of extension commands that are specific to implementing extension commands in Java. Detailed information on creating extension commands is provided in the article *"Writing IBM SPSS Statistics Extension Commands"*, available from the IBM SPSS Predictive Analytics community at https://www.ibm.com/community/spss-statistics .

Implementation code

The implementation code can consist of a JAR file or Java class files.

- When using a JAR file, the name of the JAR file must be the same as the name of the extension command, and in upper case. For multi-word command names, spaces between words should be replaced with underscores when constructing the name of the JAR file. The JAR file must contain a class file with the same name as the JAR file and the class must not be part of a package.
- When using standalone Java class files, there must be one class file with the same name as that of the extension command, and in upper case. For multi-word command names, spaces between words should be replaced with underscores when constructing the name of the Java class file.

Whether using a JAR file or standalone Java class files, the class file with the same name as the extension command should contain the following:

- A constructor method, which does not have a parameter, for the class.
- A public static method named Run with a single Hashtable argument that should be specified as follows:

```
Hashtable<String, Hashtable<String, Object>>
```

- A public method that implements the command.

XML command syntax specification

- For extension commands implemented in Java, the Language attribute of the Command element should be set to "Java".
- For Java extension commands implemented with a JAR file, the Mode attribute of the Command element should be set to "Package".

Sample Java class

The following is an example of a Java class for an extension command named DEMO, which simply takes a variable list as input and prints out the list. It demonstrates the basic structure of the implementation code and the means by which values are passed from the command syntax (submitted by the user) to the method that implements the command. The XML command syntax specification for the DEMO command is also provided.

```
import java.util.Arrays;
import java.util.HashSet;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.List;
import com.ibm.statistics.plugin.*;

public class DEMO {

   public DEMO() {
      System.out.println("This is the constructor method");
   }

   public static void Run(Hashtable<String, Hashtable<String, Object>> args)
   {
           try {
               List<TemplateClass> testList = Arrays.asList(
                      Extension.Template("VARIABLES", "","vars", "existingvarlist", true));

               SyntaxClass oobj = Extension.Syntax(testList);
               Extension.processcmd(oobj, args, "printvars");

           } catch (Exception e) {
               e.printStackTrace();
           }
   }

   public void printvars(@ParamName("vars") ArrayList<String> variables)
   {
      System.out.println("variables = " + variables);
   }
}
```

Briefly, the functions of the Run, Template, Syntax and processcmd methods are as follows:

- IBM SPSS Statistics parses the command syntax entered by the user and passes the specified values to the Run function in a single argument--*args* in this example.
- The Run function contains calls to the Extension.Syntax, Extension.Template, and Extension.processcmd methods, which are designed to work together.

  Extension.Template specifies the details needed to process a specified keyword in the syntax for an extension command. In this example, the extension command contains the single keyword VARIABLES.

  Extension.Syntax validates the values passed to the Run function according to the templates specified for the keywords.

  Extension.processcmd parses the values passed to the Run function and calls the function that will actually implement the command--in this example, the printvars method.

Values of specific keywords from the submitted syntax are mapped to variables in the method that implements the command (printvars in this example) using a Java annotation. In this example, the Extension.Template method specifies that the value of the VARIABLES keyword is associated with the identifier "vars". The argument to the printvars method specifies that this identifier is mapped to the local variable *variables*. This is accomplished with the @ParamName("vars") annotation. You include such an annotation for each keyword in the syntax for the extension command.

The annotation mechanism also requires that arguments to the implementation method are defined as object types, not primitive data types. In particular, you must use the object types Integer, Short, Long, Float, Double, Byte, Character and Boolean instead of the primitive data types int, short, long, float,

double, byte, char and boolean. Note that in the above example, the value passed to the `printvars` method is an array of strings and is defined as an `ArrayList` object.

The XML specification for the DEMO command is as follows:

```
<Command xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="extension.xsd"
 Name="DEMO" Language="Java" Mode="Source">
   <Subcommand Name="">
      <Parameter Name="VARIABLES" ParameterType="VariableNameList"/>
   </Subcommand>
</Command>
```

- The top level element `Command` specifies the name of the extension command and the implementation language. The `Mode` attribute is set to "Source", which specifies that the implementation code is contained in Java class files rather than a JAR file.
- The `Subcommand` element specifies the single subcommand for this extension command. In this simple example, the subcommand is unnamed.
- The `Parameter` element specifies the single parameter for this extension command. The parameter is named *VARIABLES* and it specifies a list of variable names.

Deploying an extension command

To deploy an extension command, it is best to create an extension bundle for the command and add both the implementation code (JAR file or Java class files) and the XML file specifying the command syntax to the bundle. You can distribute the extension bundle (*spe*) file to other users who can then install it and begin using your extension command. For more information, see Core System>Extensions>Creating and editing extension bundles, in the Help system.

Chapter 2. Running IBM SPSS Statistics commands

Running IBM SPSS Statistics commands

The `submit` method from the `StatsUtil` class is used to submit syntax commands to IBM SPSS Statistics for processing. It takes a string that resolves to a complete syntax command, or an array of strings that resolves to one or more complete syntax commands. Output from syntax commands can be written to the standard output stream or to an external file. It can also be directed to an in-memory workspace where it is stored as XML and can then be retrieved using XPath expressions. See the topic “Retrieving output from syntax commands ” on page 15 for more information.

Submitting a single command

You submit a single command to IBM SPSS Statistics by providing a string representation of the command as shown in this example. When submitting a single command in this manner the period (.) at the end of the command is optional.

```
StatsUtil.submit("GET FILE='/data/Employee data.sav'.");
StatsUtil.submit("DESCRIPTIVES SALARY.");
```

- The `submit` method is called twice; first to submit a `GET` command and then to submit a `DESCRIPTIVES` command.

Submitting commands using an array

You can submit multiple commands as an array of strings where each array element is a string representation of a syntax command. The string for each command must be terminated with a period (.) as shown in this example.

```
StatsUtil.submit("GET FILE='/data/Employee data.sav'.");
String[] cmdLines = {"DESCRIPTIVES SALARY SALBEGIN.","FREQUENCIES EDUC JOBCAT."};
StatsUtil.submit(cmdLines);
```

- The `submit` method is called with an array that specifies a `DESCRIPTIVES` and a `FREQUENCIES` command.

You can also use the elements of an array to represent parts of a command so that a single array specifies one or more complete syntax commands. When submitting multiple commands in this manner, each command must be terminated with a period (.) as shown in this example.

```
StatsUtil.submit("GET FILE='/data/Employee data.sav'.");
String[] cmdLines = {"OMS /SELECT TABLES ",
     "/IF COMMANDS = ['Descriptives' 'Frequencies'] ",
     "/DESTINATION FORMAT = HTML ",
     "IMAGES = NO OUTFILE = '/output/stats.html'.",
     "DESCRIPTIVES SALARY SALBEGIN.",
     "FREQUENCIES EDUC JOBCAT.",
     "OMSEND."};
StatsUtil.submit(cmdLines);
```

- The `submit` method is called with an array that specifies an `OMS` command followed by a `DESCRIPTIVES` command, a `FREQUENCIES` command, and an `OMSEND` command. The first four elements of the array are used to specify the `OMS` command.

Displaying command syntax generated by the submit method

For debugging purposes, it is convenient to see the completed syntax passed to IBM SPSS Statistics by any calls to the `submit` method. This is enabled through command syntax with `SET PRINTBACK ON MPRINT ON`.

```
String[] cmdLines = {"SET PRINTBACK ON MPRINT ON.",
"GET FILE='/data/Employee data.sav'."};
StatsUtil.submit(cmdLines);
String varName;
varName = StatsUtil.getVariableName(1);
StatsUtil.submit("FREQUENCIES /VARIABLES=" + varName + ".");
```

The generated command syntax shows the completed FREQUENCIES command as well as the GET command. In the present example the variable with index value 1 in the dataset has the name *gender*.

```
M>  GET FILE='c:/data/Employee data.sav'.
M>  FREQUENCIES /VARIABLES=gender.
```

Chapter 3. Retrieving dictionary information

Retrieving dictionary information

The `Cursor`, `DataUtil` and `StatsUtil` classes provide a number of methods for retrieving dictionary information from the active dataset. The following information is available:

- **Cursor.getDataFileAttributes.** The attribute values for a specified datafile attribute.
- **Cursor.getDataFileAttributesNames.** Names of any datafile attributes for the active dataset.
- **Cursor.getMultiResponseSet.** The details of a specified multiple response set.
- **Cursor.getMultiResponseSetNames.** The names of any multiple response sets for the active dataset.
- **Cursor.getNumericMissingValues.** The user-missing values, if any, for a specified numeric variable.
- **Cursor.getNumericValueLabels.** The value labels, if any, for a specified numeric variable.
- **Cursor.getStringMissingValues.** The user-missing values, if any, for a specified string variable.
- **Cursor.getStringValueLabels.** The value labels, if any, for a specified string variable.
- **Cursor.getVariableAttributeNames.** Names of any custom variable attributes for a specified variable.
- **Cursor.getVariableAttributes.** The attribute values for a specified attribute of a specified variable.
- **Cursor.getVariableCount.** The number of variables in the active dataset.
- **Cursor.getVariableFormat.** The display format for a specified variable; for example, *F8.2*.
- **Cursor.getVariableLabel.** The variable label, if any, for a specified variable.
- **Cursor.getVariableMeasurementLevel.** The measurement level for a specified variable.
- **Cursor.getVariableName.** The variable name for a variable specified by its index position. Index positions start with 0 for the first variable in file order.
- **Cursor.getVariableRole.** The variable role (for example, INPUT or TARGET) for a specified variable.
- **Cursor.getVariableType.** The variable type (numeric or string) for a specified variable.
- **DataUtil.getVariableIndex.** The index position, in the active dataset, of a specified variable.
- **DataUtil.getVariableNames.** The names of the variables in the active dataset.
- **StatsUtil.getSplitVariableNames.** The names of the split variables, if any.
- **StatsUtil.getWeightVariable.** The name of the weight variable, if any.

Example

Consider the common scenario of running a particular block of command syntax only if a specific variable exists in the dataset. For example, you are processing many datasets containing employee records and want to split them by gender--if a gender variable exists--to obtain separate statistics for the two gender groups. We will assume that if a gender variable exists, it has the name *gender*, although it may be spelled in upper case or mixed case. The following sample code illustrates the approach:

```
StatsUtil.submit("GET FILE='/data/Employee data.sav'.");
DataUtil datautil = new DataUtil();
String[] varnames = datautil.getVariableNames();
datautil.release();
for(String name: varnames){
     if(name.toLowerCase().equals("gender")){
         String[] command={"SORT CASES BY " + name + ".",
                           "SPLIT FILE LAYERED BY " + name + "."};
         StatsUtil.submit(command);
     }
}
```

Chapter 4. Working with case data in the active dataset

Working with case data in the active dataset

The IBM SPSS Statistics - Integration Plug-in for Java provides the ability to read case data from the active dataset, create new variables in the active dataset, and append new cases to the active dataset. The functionality for reading from and writing to the active dataset is provided in the `Cursor` class. An instance of the `Cursor` class creates an open cursor, which provides access to the active dataset. The following rules apply to the use of cursors:

- You cannot use the `submit` method from the `StatsUtil` class while a data cursor is open. You must close the cursor first using the `close` method. In particular, if you need to save changes made to the active dataset to an external file, then use the `submit` method to submit a SAVE command after closing the cursor.
- Only one data cursor can be open at any point in an application. To define a new data cursor, you must first close the previous one.

While the `Cursor` class provides the full set of methods for accessing the active dataset, the simpler `DataUtil` class is a wrapper for the `Cursor` class and provides the ability to read cases, create new variables and append new cases. The examples in this section use the `DataUtil` class. Because the `DataUtil` class is a wrapper for the `Cursor` class, the above limitations on cursors also apply to `DataUtil` objects. The following apply:

- You cannot use the `submit` method from the `StatsUtil` class while a `DataUtil` object exists. You must release the resources associated with the object with the `release` method. As with cursors, if you need to save changes made to the active dataset to an external file, then use the `submit` method to submit a SAVE command after releasing the `DataUtil` object.
- Only one `DataUtil` object can exist at a time. To create a new `DataUtil` object, you must first release the previous one.

Note: To create a new dataset, use the `submit` method in the `StatsUtil` class to submit a `DATA LIST` command.

Reading case data

You retrieve cases using the `fetchCases` method from the `DataUtil` class. You can retrieve cases one at a time in sequential order or you can retrieve multiple cases (including all cases) with a single call to the `fetchCases` method.

- System-missing values are always returned as the Java *null* value, however you can specify whether user-missing values are treated as valid or also returned as *null*. See the example on missing data.
- By default, data retrieved from a variable representing a date, or a date and a time, is given as the number of seconds from October 14, 1582. You can specify that values read from IBM SPSS Statistics variables with date or datetime formats be converted to Java Calendar objects with the `setConvertDateTypes` method as shown in the following example.
- When retrieving cases, any case filtering specified with the `FILTER` or `USE` commands is honored.

Example

```
StatsUtil.submit("GET FILE='/data/demo.sav'.");
DataUtil datautil = new DataUtil();
datautil.setConvertDateTypes(true);
Case[] data = datautil.fetchCases(false, 0);
Double numvar;
String strvar;
Calendar datevar;
for(Case onecase: data){
   for(int i = 0;i<onecase.getCellNumber();i++){
```

```
      CellValueFormat format = onecase.getCellValueFormat(i);
      if(format == CellValueFormat.DOUBLE){
         numvar = onecase.getDoubleCellValue(i);
      }
      else if(format == CellValueFormat.STRING){
         strvar = onecase.getStringCellValue(i);
      }
      else if(format == CellValueFormat.DATE){
           datevar = onecase.getDateCellValue(i);
      }
   }
}
datautil.release();
```

- You first create an instance of the `DataUtil` class. In this example, the variable `datautil` is a `DataUtil` object.
- The `setConvertDateTypes` method specifies that values read from IBM SPSS Statistics variables with date or datetime formats will be converted to Java Calendar objects.
- The `fetchCases` method retrieves all cases from the active dataset. The first argument specifies that user-missing values will be treated as missing and thus converted to the Java *null* value. The second argument specifies that all cases, starting with case 0, will be retrieved from the active dataset. You can retrieve cases starting from an arbitrary case number by specifying a different value for the second argument. You can also retrieve a specified number of cases, using an overloaded form of `fetchCases` with a third argument specifying the number of cases to retrieve.
- The `fetchCases` method returns a `Case` object, which represents an array of cases. The items in a given element of the array correspond to the values of the variables in a particular case of data from the active dataset, in file order. You can get the number of items in each case from the `getCellNumber` method of the `Case` object.
- The type of value in each item of a case is available from the `getCellValueFormat` method of the `Case` object. Values are retrieved from the `Case` object with methods specific to each type of value, as shown here for numeric, string and date values.

Retrieving data for a subset of variables

You can specify a subset of variables for which data will be retrieved. You can specify the set of variables by name or by their index position in the active dataset. Index positions start with 0 for the first variable in file order.

```
StatsUtil.submit("GET FILE='/data/employee data.sav'.");
DataUtil datautil = new DataUtil();
String[] varNames = new String[]{"id","educ","salary"};
datautil.setVariableFilter(varNames);
Case[] data = datautil.fetchCases(false, 0);
```

The `setVariableFilter` method specifies the subset of variables for which data will be retrieved. In this example, only data for the variables *id*, *educ* and *salary* will be retrieved.

Missing data

The first argument to the `fetchCases` method specifies whether user-missing values are converted to the Java *null* value or treated as valid data. System-missing values are always converted to the Java *null* value.

```
String[] command={"DATA LIST LIST (',') /numVar (f) stringVar (a4).",
"BEGIN DATA",
"1,a",
",b",
"3,,",
"9,d",
"END DATA.",_
"MISSING VALUES numVar (9) stringVar (' ')."}
StatsUtil.submit(command);
DataUtil datautil = new DataUtil();
Case[] data = datautil.fetchCases(false, 0);
datautil.release();
```

Setting the first argument to `fetchCases` to *false* specifies that user-missing values are converted to the Java *null* value. The values read from IBM SPSS Statistics and stored in the variable *data* are:

```
1      a
null   b
3      null
null   d
```

You can specify that user-missing values be treated as valid data by setting the first argument to the `fetchCases` method to `true`. The values of *data* are now:

```
1      a
null   b
3
9      d
```

Handling Data with Splits

The `getSplitIndex` method, from the `DataUtil` class, allows you to detect split changes when reading from datasets that have splits.

```
String[] command={"DATA LIST FREE /salary (F) jobcat (F).",
"BEGIN DATA",
"21450 1",
"45000 1",
"30000 2",
"30750 2",
"103750 3",
"72500 3",
"57000 3",
"END DATA.",
"SPLIT FILE BY jobcat."};
StatsUtil.submit(command);
DataUtil dautil = new DataUtil();
int splitindex;
splitindex = dautil.getSplitIndex(0);
while(splitindex!=-1){
   System.out.println("A new split begins at case: " + splitindex);
   splitindex = dautil.getSplitIndex(splitindex);
}
dautil.release();
```

- `dautil.getSplitIndex` gets the case number of the first case in the split following the specified case. For the sample dataset used in this example, split boundaries are crossed when reading the 3rd and 5th cases. Case numbers start from 0.
- If there are no split boundaries following the specified case, then `dautil.getSplitIndex` returns -1.

Creating new variables in the active dataset

The `DataUtil` class enables you to add new variables, along with their case values, to the active dataset.

Example

In this example we create a new string variable, a new numeric variable and a new date variable, and populate case values for them. A sample dataset is first created.

```
String[] command={"DATA LIST FREE /case (A5).",
"BEGIN DATA",
"case1",
"case2",
"case3",
"END DATA."};
StatsUtil.submit(command);
Variable numVar = new Variable("numvar",0);
Variable strVar = new Variable("strvar",1);
Variable dateVar = new Variable("datevar",0);
dateVar.setFormatType(VariableFormat.DATE);
double[] numValues = new double[]{1.0,2.0,3.0};
String[] strValues = new String[]{"a","b","c"};
Calendar dateValue = Calendar.getInstance();
dateValue.set(Calendar.YEAR, 2012);
dateValue.set(Calendar.MONTH, Calendar.JANUARY);
dateValue.set(Calendar.DAY_OF_MONTH, 1);
Calendar[] dateValues = new Calendar[]{dateValue};
DataUtil dautil = new DataUtil();
dautil.addVariableWithValue(numVar, numValues, 0);
dautil.addVariableWithValue(strVar, strValues, 0);
dautil.addVariableWithValue(dateVar, dateValues, 0);
dautil.release();
```

- The `Variable` class creates the specification for a new variable to be added to the active dataset. The first argument to the constructor is the name of the variable and the second argument is an integer specifying the variable type. Numeric variables have a variable type of 0 and string variables have a variable type equal to the defined length of the string (maximum of 32767 bytes).
- The `addVariableWithValue` method of the `DataUtil` class adds a new variable to the active dataset. The first argument to the method is the `Variable` object that specifies the properties of the variable. The second argument is an array that specifies the value of the variable for each case in the

active dataset to be populated. The third argument specifies the index of the case at which to begin populating the variable values. Case indexes start with 0 for the first case in the active dataset.

For numeric variables, cases that are not populated are set to the system-missing value. For string variables, cases that are not populated are set to a blank value. In this example, only the first case is populated for the variable *dateVar*.

- Variables representing a date, or a date and a time, in IBM SPSS Statistics are numeric variables that have a date or datetime format. In the above example, the variable *dateVar* is a numeric variable whose format has been set to DATE with the `setFormatType` method of the associated `Variable` object. When setting the value for such a variable, use a Java Calendar object as shown in this example.

Note: To save the modified active dataset to an external file, use the `submit` method (following the `release` method) to submit a SAVE command, as in:

```
StatsUtil.submit("SAVE OUTFILE='/data/mydata.sav'.")
```

Example: Multiple data passes

Sometimes more than one pass of the data is required, as in the following example involving two data passes. The first data pass is used to read the data and compute a summary statistic. The second data pass is used to add a summary variable to the active dataset.

```
 String[] command={"DATA LIST FREE /var (F).",
"BEGIN DATA",
"40200",
"21450",
"21900",
"END DATA."};
StatsUtil.submit(command);
Double total = 0.0;
DataUtil datautil = new DataUtil();
Case[] data = datautil.fetchCases(false, 0);
for(Case onecase: data){
   total = total + onecase.getDoubleCellValue(0);
}
Double meanval = total/data.length;
Variable mean = new Variable("mean",0);
double[] meanVals = new double[data.length];
for (int i=0;i<data.length;i++){
   meanVals[i]=meanval;
}
datautil.addVariableWithValue(mean, meanVals, 0);
datautil.release();
```

Appending new cases

The `DataUtil` class enables you to append new cases to the active dataset.

Example

In this example a single case is appended to the active dataset.

```
String[] command={"DATA LIST FREE /case (F) value (A1) date(ADATE).",
"BEGIN DATA",
"1 a 01/01/2012",
"END DATA."};
StatsUtil.submit(command);
Case newcase = new Case(3);
DataUtil datautil = new DataUtil();
Calendar date = Calendar.getInstance();
date.set(Calendar.YEAR, 2013);
date.set(Calendar.MONTH, Calendar.JANUARY);
date.set(Calendar.DAY_OF_MONTH, 1);
newcase.setCellValue(0, 2);
newcase.setCellValue(1, "b");
newcase.setCellValue(2, date);
datautil.appendCase(newcase);
datautil.release();
```

- To append a case to the active dataset, you create a `Case` object that represents a single case, and populate it with the values of the variables for the case. The variable values are populated with the `setCellValue` method of the `Case` object. The first argument to this method is the index of the associated variable in the active dataset, starting with 0 for the first variable in file order. In this example, the variable *case* has index 0, the variable *value* has index 1 and the variable *date* has index 2. The second argument to the `setCellValue` method is the value of the variable for the given case.
- To specify the value of variable with a date or datetime format, use a Java Calendar object as shown in this example.

- The appendCase method of the DataUtil object appends the case. Its only argument is the Case object that specifies the case.
- A numeric variable whose value is not specified in the Case object is set to the system-missing value. A string variable whose value is not specified in the Case object will have a blank value. The value will be valid unless you explicitly define the blank value to be missing for that variable. You can also use an overloaded form of the setCellValue method to set the value of a specified cell in the Case object to the Java *null* value, in which case it will be treated the same as if the value of the cell was not specified.

Chapter 5. Retrieving output from syntax commands

Retrieving output from syntax commands

You can retrieve output generated by IBM SPSS Statistics commands without writing the output to an external file. This is accomplished by routing the output 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 IBM SPSS Statistics Output XML Schema. Output is retrieved from this workspace with functions that employ XPath expressions.

Constructing the correct XPath expression (IBM SPSS Statistics currently supports XPath 1.0) requires an understanding of the Output XML schema. The output schema spss-output-1.8.xsd is distributed with IBM SPSS Statistics. Documentation is included in the IBM SPSS Statistics Help system.

Example

In this example, we'll retrieve the mean value of a variable calculated from the Descriptives procedure.

```
String[] command={"GET FILE='/data/Employee data.sav'.",
          "OMS SELECT TABLES ",
          "/IF COMMANDS=['Descriptives'] SUBTYPES=['Descriptive Statistics'] ",
          "/DESTINATION FORMAT=OXML XMLWORKSPACE='desc_table' ",
          "/TAG='desc_out'.",
          "DESCRIPTIVES VARIABLES=salary, salbegin, jobtime, prevexp ",
          "/STATISTICS=MEAN.",
          "OMSEND TAG='desc_out'."};
StatsUtil.submit(command);
String handle = "desc_table";
String context = "/outputTree";
String xpath = "//pivotTable[@subType='Descriptive Statistics']" +
                   "/dimension[@axis='row']" +
                   "/category[@varName='salary']" +
                   "/dimension[@axis='column']" +
                   "/category[@text='Mean']" +
                   "/cell/@text";
String result = StatsUtil.evaluateXPath(handle, context, xpath);
StatsUtil.deleteXPathHandle(handle);
```

- 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 IBM 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 `evaluateXPath` method from the `StatsUtil` class. The method takes an explicit XPath expression, evaluates it against a specified XPath DOM in the XML workspace, and returns the result as an array of string values.
- 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). 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 mean 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='salary']` selects the row for the variable *salary*.

  `/dimension[@axis='column']/category[@text='Mean']` selects the *Mean* column within this row, 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 `deleteXPathHandle` method, whose single argument is the name of the handle associated with the item.

If you're familiar with XPath, note that the mean value of *salary* can also be selected with the following simpler XPath expression:

```
//category[@varName='salary']//category[@text='Mean']/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 syntax command `SHOW OLANG`.

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.

Writing XML workspace contents to a file

When writing and debugging XPath expressions, it is often useful to have a sample file that shows the XML structure. This can be obtained with the `getXMLUTF16` method from the `StatsUtil` class, as well as by an option on the OMS syntax command. The following code recreates the XML workspace for the preceding example and writes the XML associated with the handle *desc_table* to an external file.

```
String[] command={"GET FILE='/data/Employee data.sav'.",
          "OMS SELECT TABLES ",
          "/IF COMMANDS=['Descriptives'] SUBTYPES=['Descriptive Statistics'] ",
          "/DESTINATION FORMAT=OXML XMLWORKSPACE='desc_table' ",
          "/TAG='desc_out'.",
          "DESCRIPTIVES VARIABLES=salary, salbegin, jobtime, prevexp ",
          "/STATISTICS=MEAN.",
          "OMSEND TAG='desc_out'."};
StatsUtil.submit(command);
String result = StatsUtil.getXMLUTF16("desc_table");
Writer out = new OutputStreamWriter(new FileOutputStream("/output/descriptives_table.xml"));
out.write(result);
out.close();
StatsUtil.deleteXPathHandle("desc_table");
```

The section of the output file that specifies the Descriptive Statistics table, including the mean value of *salary*, is as follows (the output is written in Unicode (UTF-16)):

```
<pivotTable subType="Descriptive Statistics" text="Descriptive Statistics">
  <dimension axis="row" text="Variables">
    <category label="Current Salary" text="Current Salary"
    varName="salary" variable="true">
      <dimension axis="column" text="Statistics">
        <category text="N">
          <cell number="474" text="474"/>
        </category>
        <category text="Mean">
          <cell decimals="2" format="dollar" number="34419.567510548"
          text="$34,419.57"/>
        </category>
      </dimension>
    </category>
```

Retrieving images associated with an output XPath DOM

You can retrieve images associated with output routed to the XML workspace. In this example, we'll retrieve a bar chart associated with output from the Frequencies procedure.

```
String[] command={"GET FILE='/data/Employee data.sav'.",
              "OMS SELECT CHARTS ",
              "/IF COMMANDS=['Frequencies'] ",
              "/DESTINATION FORMAT=OXML IMAGES=YES",
              "CHARTFORMAT=IMAGE IMAGEROOT='myimages' IMAGEFORMAT=JPG XMLWORKSPACE='demo'.",
              "FREQUENCIES VARIABLES=jobcat",
              "  /BARCHART PERCENT",
              "  /ORDER=ANALYSIS.",
              "OMSEND."};
StatsUtil.submit(command);
String handle = "demo";
String context = "/outputTree";
String xpath = "//command[@command='Frequencies']" +
                    "/chartTitle[@text='Bar Chart']" +
                    "/chart/@imageFile";
String[] result = StatsUtil.evaluateXPath(handle, context, xpath);
String imageName = result[0];
BufferedImage imageObj = StatsUtil.getImage(handle, imageName);
File outputFile = new File("/output/barchart.jpg");
ImageIO.write(imageObj, "JPG", outputFile);
StatsUtil.deleteXPathHandle(handle);
```

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

  The first argument to the getImage function specifies the particular XPath DOM and must be a valid handle name defined by a previous IBM SPSS Statistics OMS command.

  The second argument to getImage is the filename associated with the image in the OXML output--specifically, the value of the imageFile attribute of the chart element associated with the image.

Chapter 6. Creating custom output

Creating custom output

The IBM SPSS Statistics - Integration Plug-in for Java provides the ability to create output in the form of custom pivot tables and text blocks. Using the Output Management System (OMS), the output can be rendered in a variety of formats such as HTML, text, or XML that conforms to the IBM SPSS Statistics Output XML Schema. If you are generating output as part of an extension command implemented in Java, then the output will be displayed in the Viewer by default.

Creating pivot tables

The following figure shows the basic structural components of a pivot table. For IBM SPSS Statistics version 28, the IBM SPSS Statistics - Integration Plug-in for Java supports pivot tables with one row dimension and one column dimension. Each dimension contains a set of categories that label the elements of the dimension--for instance, row labels for a row dimension.

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 and *Custodial* for the *Employment Category* dimension.

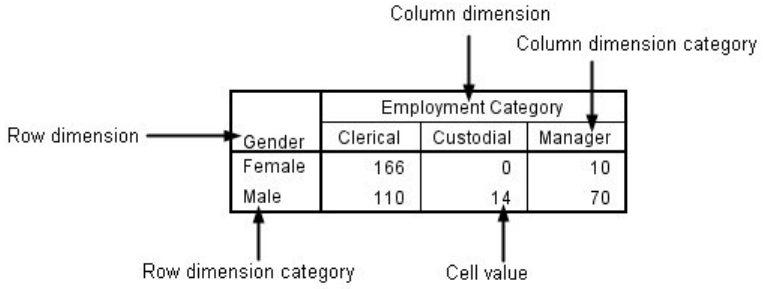

Figure 1. Pivot table structure

Pivot tables are created with the `PivotTable` class, as shown in the following example. This example assumes that you are creating pivot table output for an external Java application that will invoke IBM SPSS Statistics, so that the output is routed with OMS.

```
String[] command={
    "OMS SELECT TABLES",
    "/IF SUBTYPES=['pivotTableDemo']",
    "/DESTINATION FORMAT=HTML OUTFILE='/output/pivottable.html'."
};
StatsUtil.submit(command);
Object[] rowLabels = new Object[] {"row1", "row2"};
Object[] colLabels = new Object[] { "columnA", "columnB"};
Object[][] cells = new Object[][] {{"1A","1B"}, {"2A","2B"}};
String title = "Sample pivot table";
String templateName = "pivotTableDemo";
String outline = "";
String caption = "";
String rowDim = "Row dimension";
String columnDim = "Column dimension";
boolean hideRowDimTitle = false;
boolean hideRowDimLabel = false;
boolean hideColDimTitle = false;
boolean hideColDimLabel = false;
PivotTable table = new PivotTable(cells, rowLabels, colLabels,
                        title, templateName, outline, caption, rowDim,
                        columnDim, hideRowDimTitle, hideRowDimLabel,
                        hideColDimTitle, hideColDimLabel, FormatSpec.COEFFICIENT);
table.createSimplePivotTable();
StatsUtil.submit("OMSEND.");
```

Result

Sample pivot table

	Column dimension	
Row dimension	columnA	columnB
row1	1A	1B
row2	2A	2B

Figure 2. Sample pivot table

- In this example, pivot table output is routed to an external HTML file using OMS (Output Management System). The OMS command specifies that only tables with a table subtype of *pivotTableDemo* (to be defined below) will be included in the output. It also specifies the path to the output file and that the output will be rendered in HTML.
- The command syntax for the OMS command is submitted to IBM SPSS Statistics. This starts an OMS session. The session is closed with the OMSEND command, which then writes the output to the destination file.

  Note: As an alternative to routing output with OMS, you can also specify output formats and destinations using an overloaded form of the `StatsUtil.start` method that accepts a string with output specifications. OMS offers greater flexibility but, for simple applications, specifying output on the `start` method may be sufficient.
- To create a pivot table, you create an instance of the `PivotTable` class. The arguments are as follows:

  cells. This argument specifies the values for the cells of the pivot table, and must be given as a 2-dimensional array, where each element of the array specifies the cells in a given row of the pivot table. Only Double and String objects can be specified in this array.

  rowLabels. A 1-dimensional array of categories for the row dimension. Only Double and String objects can be specified in this array.

  colLabels. A 1-dimensional array of categories for the column dimension. Only Double and String objects can be specified in this array.

  title. A string specifying the title of the table.

  templateName A string that specifies the OMS (Output Management System) table subtype for this table. This value must begin with a letter, and have a maximum of 64 bytes. The table subtype can be used on the SUBTYPES keyword of the OMS command, as done here, to include this pivot table in the output for a specified set of subtypes.

  Note: By creating the pivot table instance within a `startProcedure-endProcedure` block, you can associate the pivot table output with a command name, as for pivot table output produced by syntax commands. The command name is the argument to the `startProcedure` function and can used on the COMMANDS keyword of the OMS command to include the pivot table in the output for a specified set of commands (along with optionally specifying a subtype as discussed above).

  outline. A string that specifies an optional outline title for the table. When routing output to an SPV file with OMS, or generating output in the Viewer for an extension command implemented in Java, the pivot table will be nested under an item with the outline name. When output is routed to OMS in OXML format, the outline title specifies a heading tag that will contain the output for the pivot table.

  caption. A string that specifies a table caption.

  rowDim. A string specifying the label for the row dimension.

  columnDim. A string specifying the label for the column dimension.

  hideRowDimTitle. A boolean specifying whether to hide the row dimension title.

  hideRowDimLabels. A boolean specifying whether to hide the row labels.

  hideColDimTitle. A boolean specifying whether to hide the column dimension title.

  hideColDimLabels. A boolean specifying whether to hide the column labels.

format. Specifies the format to be used for displaying numeric values, including cell values, row labels, and column labels. The argument is specified as a `FormatSpec` enum.

Creating text blocks

Text blocks are created with the `addTextBlock` method in the `StatsUtil` class.

```
String[] command={"OMS SELECT TEXTS",
"/IF LABELS = ['Text block name']",
"/DESTINATION FORMAT=HTML OUTFILE='/output/textblock.htm'."
};
StatsUtil.submit(command);
StatsUtil.startProcedure("demo");
StatsUtil.addTextBlock("Text block name", "The first line of text.");
StatsUtil.addTextBlock("Text block name", "The second line of text.",1);
StatsUtil.endProcedure();
StatsUtil.submit("OMSEND.");
```

- In the common case that you're creating text blocks along with other output such as pivot tables, you'll probably be routing the output with OMS (unless you are creating text blocks for an extension command implemented in Java). To route text block output to OMS, you include the TEXTS keyword on the SELECT subcommand of the OMS command, as in this example. As specified on the OMS command, the text block in this example will be routed to an HTML file.
- The `addTextBlock` method must be called within a `startProcedure - endProcedure` block, as shown in this example. `startProcedure - endProcedure` blocks define a set of output (pivot tables and text blocks) that is associated with a name (in this example, *demo*). When routing output to a Viewer (*spv*) file with OMS or creating output for an extension command implemented in Java, `startProcedure - endProcedure` blocks allow you to group output under a common heading, as is done for the set of output generated by an IBM SPSS Statistics command.
- The first argument to the `addTextBlock` method is a string that specifies the name of the text block. The name can be used on the LABELS keyword of the OMS command, as done here, to limit the textual output routed to OMS.
- The second argument to the `addTextBlock` method is the content of the text block as a string.
- You can append additional lines by calling an overloaded version of the `addTextBlock` method, supplying the same text block name, the content for the next line, and an integer specifying the number of lines to skip before the new line. You can also use the escape sequence *n* to specify line breaks, allowing you to specify multiple lines in a single call to `addTextBlock`.

Creating output for extension commands

Pivot tables and text blocks created from extension commands are displayed in the Viewer by default. If you are displaying output in the Viewer then you will probably want to group your output under a common heading. This is done by wrapping the output generation in a `startProcedure - endProcedure` block, which is actually required for text blocks but optional for pivot tables. If you don't wrap your output in a `startProcedure - endProcedure` block, then your output will be grouped under the heading *UserProcedure* in the Viewer. Following is a reworking of the example in “Creating pivot tables ” on page 19, but wrapping the pivot table generation in a `startProcedure - endProcedure` block.

```
Object[] rowLabels = new Object[] {"row1", "row2"};
Object[] colLabels = new Object[] { "columnA", "columnB"};
Object[][] cells = new Object[][] {{"1A","1B"}, {"2A","2B"}};
String title = "Sample pivot table";
String templateName = "pivotTableDemo";
String outline = "";
String caption = "";
String rowDim = "Row dimension";
String columnDim = "Column dimension";
boolean hideRowDimTitle = false;
boolean hideRowDimLabel = false;
boolean hideColDimTitle = false;
boolean hideColDimLabel = false;
StatsUtil.startProcedure("Demo");
PivotTable table = new PivotTable(cells, rowLabels, colLabels,
                        title, templateName, outline, caption, rowDim,
                        columnDim, hideRowDimTitle, hideRowDimLabel,
                        hideColDimTitle, hideColDimLabel, FormatSpec.COEFFICIENT);
table.createSimplePivotTable();
StatsUtil.endProcedure();
```

- The argument to the `startProcedure` method specifies the name associated with the generated output, and is the name of the heading under which the output is grouped in the Viewer.
- Although not utilized in this example, the *outline* argument to the `PivotTable` method specifies a heading under which the pivot table will appear. This heading will be nested under the heading specified by the argument to the `startProcedure` method. When the argument is omitted, as in this example, the pivot table appears directly under the heading specified by the argument to the `startProcedure` method.
- The `endProcedure` method must be called to end the block.

Chapter 7. Running Python and R Programs from Java

The IBM SPSS Statistics BEGIN PROGRAM command provides the ability to integrate the capabilities of external programming languages with IBM SPSS Statistics command syntax. The supported languages are the Python programming language and R. This enables you to utilize the extensive set of scientific and statistical programming libraries available with the Python language and R to create custom algorithms that you can apply to your data. Results can be written to a dataset or the XML workspace and then retrieved by your Java program.

Python Programs

Once you have installed the IBM SPSS Statistics - Integration Plug-in for Python, you have full access to the Python programming language by including Python code in a BEGIN PROGRAM PYTHON-END PROGRAM block (within command syntax) and submitting the command syntax with the submit method from the StatsUtil class. For example:

```
String[] command = {"BEGIN PROGRAM PYTHON.",
                    "import spss",
                    "<Python code>",
                    "END PROGRAM."};
StatsUtil.submit(command);
```

- The Python statement import spss imports the Python modules (installed with the IBM SPSS Statistics - Integration Plug-in for Python) that allow the Python processor to interact with IBM SPSS Statistics. Your Python language code follows the import statement.
- You can omit the keyword PYTHON on BEGIN PROGRAM--it is the default.

Complete documentation on the functionality available with the IBM SPSS Statistics - Integration Plug-in for Python is available in the SPSS Statistics Help system under Integration Plug-in for Python.

R Programs

Once you have installed the IBM SPSS Statistics - Integration Plug-in for R, you have full access to the R programming language by including R code in a BEGIN PROGRAM R-END PROGRAM block (within command syntax) and submitting the command syntax with the submit method from the StatsUtil class. For example:

```
String[] command = {"BEGIN PROGRAM R.",
                    "<R code>",
                    "END PROGRAM."};
StatsUtil.submit(command);
```

Complete documentation on the functionality available with the IBM SPSS Statistics - Integration Plug-in for R is available in the SPSS Statistics Help system under Integration Plug-in for R.

Inserting Programs From Command Syntax Files

You can use the IBM SPSS Statistics INSERT command to include BEGIN PROGRAM-END PROGRAM blocks contained in command syntax files. This allows you to store your programs as separate code files and include them as needed. For example:

```
StatsUtil.submit("INSERT FILE='/myprograms/program_block.sps'.")
```

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

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

Chapter 8. Deploying an external Java application

A Java application that externally invokes IBM SPSS Statistics through the IBM SPSS Statistics - Integration Plug-in for Java must be deployed on a machine that has a licensed IBM SPSS Statistics application. For information about licensing or distribution arrangements, please contact IBM Corp. directly. Support for the IBM SPSS Statistics - Integration Plug-in for Java was introduced in version 21.

As discussed in Chapter 1, “Getting started with the Integration Plug-in for Java ,” on page 1, the JAR file containing the Java package for the IBM SPSS Statistics - Integration Plug-in for Java is located in the IBM SPSS Statistics installation directory, so your application must be able to locate this directory. A utility distributed with the IBM SPSS Statistics - Programmability SDK is provided for this purpose. This software development kit (SDK) is available from http://www.ibm.com/developerworks/spssdevcentral.

The utility provides two means for obtaining the IBM SPSS Statistics installation directory. You can either run a script which will write the location of the latest installed version of IBM SPSS Statistics to a *.ini* file, or you can import a JAR file with methods that will allow you to obtain that location or the locations of all installed versions of IBM SPSS Statistics (beginning with version 21). The components available with the utility can be found in the *StatisticsUtil* folder under the location where you extract the ZIP file containing the IBM SPSS Statistics - Programmability SDK.

Using an ini file

To write the location of the latest installed version of IBM SPSS Statistics (beginning with version 21) to a *.ini* file, run *StatisticsUtil.bat* (for Windows platforms) or *StatisticsUtil.sh* (for non-Windows platforms). The script writes the location of IBM SPSS Statistics to the file *Statistics.ini*, located in the folder that contains the script. Specifically, it creates a key-value pair with *STATISTICS_PATH* as the key and the location as the value--for example, *STATISTICS_PATH=C:\Program Files\IBM\SPSS\Statistics\21*. If *Statistics.ini* already exists, the script updates the file. If *Statistics.ini* does not exist, then the script will create it.

Using methods in the JAR file

The file *StatisticsUtil.jar* provides the following two methods in the `Utility` class:

- **getStatisticsLocationLatest.** This method returns a string with the path to the latest installed version of IBM SPSS Statistics, starting with version 21.
- **getStatisticsLocationAll.** This method returns a string array with the paths to all installed versions of IBM SPSS Statistics, starting with version 21.

Notes

- For instances of the client version of SPSS Statistics, the term 'latest' means the highest installed version. For instances of the server version of SPSS Statistics on UNIX platforms, where multiple instances with the same version number may exist on the same machine, the 'latest' version is the one with the highest version number and the most recent time stamp.
- If there is both a client and a server version of SPSS Statistics installed as the latest version, then the path to the server version is the one written to the *.ini* file or returned by the `getStatisticsLocationLatest` method.

Notices

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

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

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

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

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

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

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

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

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

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

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

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

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

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

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

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

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

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

Trademarks

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

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

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

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

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

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

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

Index

D

J

O

P

R

X