IBM SPSS Modeler Extensions

Note

Before you use this information and the product it supports, read the information in “Notices” on page 55.

Product Information

This edition applies to version 18, release 0, modification 0 of IBM SPSS Modeler and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. Supported languages

IBM® SPSS® Modeler supports R and Apache Spark (via Python). See the following sections for more information.

R

IBM SPSS Modeler supports R. See the following sections for more information.

IBM SPSS Modeler R Nodes

IBM SPSS Modeler R Nodes - Overview

To complement IBM SPSS Modeler and its data mining abilities, the IBM SPSS Modeler R nodes enable expert R users to input their own R script to carry out data processing, model building and model scoring.

If you have a compatible copy of R installed, you can connect to it from IBM SPSS Modeler and carry out model building and model scoring using custom R algorithms that can be deployed in IBM SPSS Modeler. You must also have a copy of IBM SPSS Modeler - Essentials for R installed. IBM SPSS Modeler - Essentials for R provides you with tools you need to start developing custom R applications for use with IBM SPSS Modeler. See the document *IBM SPSS Modeler - Essentials for R: Installation Instructions* for information about installation instructions and version compatibility.

Note: You should instantiate your data in a Type node before using the IBM SPSS Modeler R nodes.

Note: R Nodes do not support variable and value labels; any labels that are included in your data are removed when that data is processed through an R node.

The IBM SPSS Modeler R plug-in contains the following nodes:

The R Transform node enables you to take data from an IBM SPSS Modeler stream and modify the data using your own custom R script. After the data is modified it is returned to the stream. See the topic "R Transform Node" on page 2 for more information.

The R Building node enables you to enter custom R script to perform model building and model scoring deployed in IBM SPSS Modeler. Executing an R Building node generates an R model nugget. See the topic "R Building Node" on page 3 for more information.

The R model nugget resembles a standard IBM SPSS Modeler model nugget (also known as a model applier node), and defines a container for a generated model to be used when the model is added to the IBM SPSS Modeler canvas from the **Models** tab of the manager pane. The R model nugget can be edited to view the various forms of model output. See the topic "R Model Nugget" on page 5 for more information.

The R Output node enables you to analyze data and the results of model scoring using your own custom R script. The output of the analysis can be text or graphical. The output is added to the **Output** tab of the manager pane; alternatively, the output can be redirected to a file. See the topic "R Output Node" on page 7 for more information.

If you are working in distributed mode, and have installed R and IBM SPSS Modeler - Essentials for R on IBM SPSS Modeler Server, you can run the R model nugget against the Netezza, Oracle, or SAP HANA databases. The R Transform node and the R model nugget can be run against IBM SPSS Analytic Server.

R Transform Node

With the R Transform node, you can take data from an IBM SPSS Modeler stream and apply transformations to the data using R scripting. When the data has been modified, it is returned to the stream for further processing, model building and model scoring. The R Transform node makes it possible to transform data using algorithms that are written in R, and enables the user to develop data transformation methods that are tailored to a particular problem.

To use this node, you must have installed IBM SPSS Modeler - Essentials for R. See the document *IBM SPSS Modeler - Essentials for R: Installation Instructions* for installation instructions and compatibility information. You must also have a compatible version of R installed on your computer.

R Transform Node - Syntax Tab: **R Transform syntax** You can enter, or paste, custom R scripting syntax for data transformation into this field.

Note: For more information about the syntax that is supported for use in this field, see the topic "Allowable Syntax" on page 9.

Read data in batches If you are processing a large amount of data, that is too big to fit into the R engine's memory, use this option to break the data down into batches that can be sent and processed individually. Specify the maximum number of data records to be included in each batch.

Starting with SPSS Modeler version 17.1, the addition of a non-batch data transfer mode, in both the R Transform node and the R Scoring nugget, means that you can either span or combine rows in the data in SPSS Modeler Server.

Convert flag fields Specifies how flag fields are treated. There are two options: **Strings to factor, Integers and Reals to double**, and **Logical values (True, False)**. If you select **Logical values (True, False)** the original values of the flag fields are lost. For example, if a field has values Male and Female, these are changed to True and False.

Convert missing values to the R 'not available' value (NA) When selected, any missing values are converted to the R `NA` value. The value `NA` is used by R to identify missing values. Some R functions that you use might have an argument that can be used to control how the function behaves when the data contain `NA`. For example, the function might allow you to choose to automatically exclude records that contain `NA`. If this option is not selected, any missing values are passed to R unchanged, and might cause errors when your R script is executed.

Convert date/time fields to R classes with special control for time zones When selected, variables with date or datetime formats are converted to R date/time objects. You must select one of the following options:

- **R POSIXct** Variables with date or datetime formats are converted to R `POSIXct` objects.
- **R POSIXlt (list)** Variables with date or datetime formats are converted to R `POSIXlt` objects.

Note: The POSIX formats are advanced options. Use these options only if your R script specifies that datetime fields are treated in ways that require these formats. The POSIX formats do not apply to variables with time formats.

R Transform Node - Console Output Tab: The **Console Output** tab contains any output that is received from the R console when the R script in the **R Transform syntax** field on the **Syntax** tab is executed. This output might include R error messages or warnings that are produced when the R script is executed, and text output from the R console. The output can be used, primarily, to debug the R script. The **Console**

Output tab also contains the R script from the **R Transform syntax** field. Every time the R transform script is executed, the content of the **Console Output** tab is overwritten with the output received from the R console. The console output cannot be edited.

R Building Node

With the R Building node, you can carry out model building and model scoring using R scripting within IBM SPSS Modeler. This makes it possible to carry out model building and scoring using algorithms that are written in R, and enables the user to develop modeling methods that are tailored to a particular problem. Execution of an R Building node generates an R model nugget.

To use this node, you must have installed IBM SPSS Modeler - Essentials for R. See the document *IBM SPSS Modeler - Essentials for R: Installation Instructions* for installation instructions and compatibility information. You must also have a compatible version of R installed on your computer.

R Building Node - Syntax Tab: **R model building syntax.** You can enter, or paste, custom R scripting syntax for model building into this field.

R model scoring syntax. You can enter, or paste, custom R scripting syntax for model scoring into this field. When the node is executed, the R script in this field is copied over to the R model nugget that is generated. The script itself is only executed when the R model nugget is executed.

Note: For more information about the syntax that is supported for use in these fields, see the topics “Allowable Syntax” on page 9 and “Model Building Syntax” on page 4.

Run. To create an R model nugget, click **Run**. The R model nugget is added to the Models palette, and optionally to the stream canvas.

R Building Node - Model Options Tab: **Model Name.** When **Auto** is selected, the model name is automatically set to the string "R Syntax". Select **Custom** to specify a custom model name in the adjoining text field. If you have executed the node once, and you do not specify a different model name before you execute the node again, the model from the previous execution will be overwritten.

Read Data Options. With these options, you can specify how missing values, flag fields, and variables with date or datetime formats are handled.

- **Convert flag fields.** Specifies how flag fields are treated. There are two options: **Strings to factor, Integers and Reals to double**, and **Logical values (True, False)**. If you select **Logical values (True, False)** the original values of the flag fields are lost. For example, if a field has values Male and Female, these are changed to True and False.
- **Convert missing values to the R 'not available' value (NA).** When selected, any missing values are converted to the R `NA` value. The value `NA` is used by R to identify missing values. Some R functions that you use might have an argument that can be used to control how the function behaves when the data contain `NA`. For example, the function might allow you to choose to automatically exclude records that contain `NA`. If this option is not selected, any missing values are passed to R unchanged, and might cause errors when your R script is executed.
- **Convert date/time fields to R classes with special control for time zones.** When selected, variables with date or datetime formats are converted to R date/time objects (POSIXt).

  Note: By default, date or datetime variables are not converted and are passed to R as a numeric value. The timestamp variable is a numeric value that represents the number of seconds since midnight January 1st 1970. The R date and time objects (POSIXt) use the R GMT(UTC) time zone. If you convert an R object to an R date or time object with a non-GMT(UTC) time zone, and return the date or time variable to SPSS Modeler, the date or time data may show a time zone difference.
  You can select one of two conversion options:
  - **R POSIXct.** Variables with date or datetime formats are converted to R `POSIXct` objects.
  - **R POSIXlt (list).** Variables with date or datetime formats are converted to R `POSIXlt` objects.

Note: The POSIX formats are advanced options. Use these options only if your R script specifies that datetime fields are treated in ways that require these formats. The POSIX formats do not apply to variables with time formats.

- **Output Options.** With these options, you can specify how output from R is displayed.
  - **Display R graphs as HTML.** When selected, R graphs are displayed in HTML format on the **Graph Output** tab of the R model nugget. The **Graph Output** tab displays only those plots that are generated from executing the R script in the **R model building syntax** field of the **Syntax** tab. See the topic "R Model Nugget - Graph Output Tab" on page 6 for more information.
  - **Display R text output.** When selected, any text output that is produced by executing the R script in the **R model building syntax** field is displayed on the **Text Output** tab of the R model nugget. See the topic "R Model Nugget - Text Output Tab" on page 6 for more information. If you want the text output to be saved to a file, include a call to the R `sink` function in your script. Any output that is produced after the call to the `sink` function is not displayed on the **Text Output** tab. Any R error messages or warnings that result from executing your R model building script are always displayed on the **Console Output** tab of the R Building node.

R Building Node - Console Output Tab: The **Console Output** tab contains any output that is received from the R console when the R script in the **R model building syntax** field on the **Syntax** tab is executed. This output might include R error messages or warnings that are produced when the R script is executed, and text output from the R console. The output can be used, primarily, to debug the R script. The **Console Output** tab also contains the R script from the **R model building syntax** field. Every time the model building script is executed, the content of the **Console Output** tab is overwritten with the output received from the R console. The console output cannot be edited.

If **Display R text output** is selected on the **Model Options** tab, the text output from the R console can instead be viewed on the **Text Output** tab of the R model nugget. Any R error messages or warnings that are produced when the R script is executed will still be displayed on the **Console Output** tab. See the topic "R Model Nugget - Text Output Tab" on page 6 for more information.

Model Building Syntax: In the **R model building syntax** field, you must assign the model object that is generated when your model building script is executed to the R object `modelerModel`. IBM SPSS Modeler retains this model object in the R model nugget to pass back to R when scoring data. The model object `modelerModel` can be referenced in the model scoring script. For more information, see the section "Example: Model building and scoring" on page 10. If you assign more than one model object to `modelerModel` in your model building script, only the last model object is retained for scoring data.

Additionally, there are some R objects that are automatically populated when an R Building node and an R model nugget are used in a stream:

- **modelerData.** This is an R data frame that is automatically populated with the data that flows into the R Building node and R model nugget.
- **modelerDataModel.** This is an R data frame that is automatically populated with the data model that flows into the R Building node and R model nugget. The data model describes the type and structure of the data (that is, the metadata) that flows into the nodes.

Any other R objects that are defined in the R script in the **R model building syntax** field will not be recognized if they are used in the R model scoring script. If you want to make reference to these R objects in your model scoring script, you must redefine them in the R script in the **R model scoring syntax** field.

The R script that is entered into the **R model building syntax** and **R model scoring syntax** fields is used to manipulate the R objects `modelerData` and `modelerDataModel`. For example, you might want to add to the data model, `modelerDataModel`, using your model scoring R script. The data model `modelerDataModel` must be modified to match any changes that were made to the data `modelerData`. When the R Building node is successfully executed, a model is generated and an R model nugget is created. The R object `modelerData` is automatically used as the output data of the R model nugget. The R object `modelerDataModel` is automatically used as the output data model of the R model nugget.

Note: The R scripts related to `modelerDataModel` should not be put in a block, and should be placed at the beginning of the scripts.

Creating a new field in the data model

When a new data field is added to the data `modelerData`, a field that describes the type and structure of the new data field must be added to the data model `modelerDataModel`. The new data model field must have the following R syntax structure:

```
c(fieldName="",fieldLabel="",fieldStorage="",fieldMeasure="",fieldFormat="",fieldRole="")
```

- `fieldName` is the name of the field and is required. Enter a name for the field between the quotes.
- `fieldLabel` is the label for the field and is optional. You can enter a label for the field between the quotes.
- `fieldStorage` is the storage type of the field and is required. Enter one of the following options between the quotes: `integer`, `real`, `string`, `date`, `time`, or `timestamp`.
- `fieldMeasure` is the measurement level of the field and is optional. You can enter one of the following options between the quotes: `nominal`, `ordinal`, `flag`, `discrete`, or `typeless`.
- `fieldFormat` is the format setting of the field and is optional. You can enter one of the following options between the quotes: `standard`, `scientific`, `currency`, `H-M`, `H-M-S`, `M-S`, `D-M-Y`, `M-D-Y`, `Y-M-D`, `Q-Y`, `W-Y`, `D-monthName-Y`, `monthName-Y`, `Y-dayNo`, `dayName`, or `monthName`.
- `fieldRole` is the role of the field and is optional. You can enter one of the following options between the quotes: `input`, `target`, `both`, `partition`, `split`, `freqWeight`, `recordId`, or `none`.

R Model Nugget

The R model nugget is generated and placed on the Models palette after executing the R Building node, which contains the R script that defines the model building and model scoring. By default, the R model nugget contains the R script that is used for model scoring, options for reading the data, and any output from the R console. Optionally, the R model nugget can also contain various other forms of model output, such as graphs and text output. After the R model nugget is generated and added to the stream canvas, an output node can be connected to it. The output node is then used in the usual way within IBM SPSS Modeler streams for obtaining information about the data and models, and for exporting data in various formats.

To use this node, you must have installed IBM SPSS Modeler - Essentials for R. See the document *IBM SPSS Modeler - Essentials for R: Installation Instructions* for installation instructions and compatibility information. You must also have a compatible version of R installed on your computer.

R Model Nugget - Syntax Tab: The **Syntax** tab is always present in the R model nugget.

R model scoring syntax. The R script that is used for model scoring is displayed in this field. By default, this field is enabled but not editable. To edit the R model scoring script, click **Edit**.

Edit. Click **Edit** to make the **R model scoring syntax** field editable. You can then edit your R model scoring script by typing in the **R model scoring syntax** field. For example, you might want to edit your R model scoring script if you identify an error in your model scoring script after you have executed the R model nugget. Any changes that you make to the R model scoring script in the R model nugget will be lost if you regenerate the model by executing the R Building node.

R Model Nugget - Model Options Tab: The **Model Options** tab is always present in the R model nugget.

Read Data Options. With these options, you can specify how missing values, flag fields, and variables with date or datetime formats are handled.

- **Read data in batches.** If you are processing a large amount of data, that is too big to fit into the R engine's memory, use this option to break the data down into batches that can be sent and processed individually. Specify the maximum number of data records to be included in each batch.

  For both the R Transform node and the R Scoring nugget, data passes through the R script (in batch). For this reason, R scripts for model scoring and process nodes that are run in either a Hadoop or Databse environment should not include operations that span or combine rows in the data, such as sorting or aggregation. This limitation is imposed to ensure that data can be split up in a Hadoop environment, and during in-database mining. This limitation does not apply if the scripts for model scoring are run in SPSS Modeler Server. R output and R model building nodes do not have this limitation.
- **Convert flag fields.** Specifies how flag fields are treated. There are two options: **Strings to factor, Integers and Reals to double**, and **Logical values (True, False)**. If you select **Logical values (True, False)** the original values of the flag fields are lost. For example, if a field has values Male and Female, these are changed to True and False.
- **Convert missing values to the R 'not available' value (NA).** When selected, any missing values are converted to the R `NA` value. The value `NA` is used by R to identify missing values. Some R functions that you use might have an argument that can be used to control how the function behaves when the data contain `NA`. For example, the function might allow you to choose to automatically exclude records that contain `NA`. If this option is not selected, any missing values are passed to R unchanged, and might cause errors when your R script is executed.
- **Convert date/time fields to R classes with special control for time zones** When selected, variables with date or datetime formats are converted to R date/time objects. You must select one of the following options:
  - **R POSIXct.** Variables with date or datetime formats are converted to R `POSIXct` objects.
  - **R POSIXlt (list).** Variables with date or datetime formats are converted to R `POSIXlt` objects.

  Note: The POSIX formats are advanced options. Use these options only if your R script specifies that datetime fields are treated in ways that require these formats. The POSIX formats do not apply to variables with time formats.

The options that are selected for the **Convert flag fields**, **Convert missing values to the R 'not available' value (NA)**, and **Convert date/time fields to R classes with special control for time zones** controls are not recognized when the R model nugget is run against a database. When the node is run against a database, the default values for these controls are used instead:
- **Convert flag fields** is set to **Strings to factor, Integers and Reals to double**.
- **Convert missing values to the R 'not available' value (NA)** is selected.
- **Convert date/time fields to R classes with special control for time zones** is not selected.

R Model Nugget - Graph Output Tab: The **Graph Output** tab is present in the R model nugget if requested by selecting the **Display R graphs as HTML** check box on the **Model Options** tab of the R Building node dialog box. Graphs that result from executing the model building R script can be displayed on this tab. For example, if your R script contains a call to the R `plot` function, the resulting graph is displayed on this tab. If you execute the model building script again, without having first specified a different name for the model, the content of the **Graph Output** tab from the previous execution will be overwritten.

R Model Nugget - Text Output Tab: The **Text Output** tab is present in the R model nugget if requested by selecting the **Display R text output** check box on the **Model Options** tab of the R Building node dialog box. This tab can display only text output. Any text output that is produced by executing your R model building script is displayed on this tab. If you execute the model building script again, without having first specified a different name for the model, the content of the **Text Output** tab from the previous execution will be overwritten. The text output cannot be edited.

If you include a call to the R `sink` function in your script, any output that is produced after this function is saved to the specified file and is not displayed on the **Text Output** tab.

Note: R error messages or warnings that result from executing your R model building script are always displayed on the **Console Output** tab of the R Building node.

R Model Nugget - Console Output Tab: The **Console Output** tab is always present in the R model nugget. It contains any output that is received from the R console when the R script in the **R model scoring syntax** field on the **Syntax** tab of the R model nugget is executed. This output includes any R error messages or warnings that are produced when the R script is executed, and any text output from the R console. The output can be used, primarily, to debug the R script. Every time the model scoring script is executed, the content of the **Console Output** tab is overwritten with the output received from the R console. The console output cannot be edited.

R Output Node

With the R Output node, you can use your own custom R scripts to perform data analysis and to summarize the results of model scoring. You can produce text and graphical output of your analyses. This output can be directed to a file, or viewed in the R Output Node Output Browser. The R Output node makes it possible to analyze data using algorithms that are written in R, and enables the user to develop data analysis methods that are tailored to a particular problem.

To use this node, you must have installed IBM SPSS Modeler - Essentials for R. See the document *IBM SPSS Modeler - Essentials for R: Installation Instructions* for installation instructions and compatibility information. You must also have a compatible version of R installed on your computer.

R Output Node - Syntax Tab: **R Output syntax.** You can enter, or paste, custom R scripting syntax for data analysis into this field.

Note: For more information about the syntax that is supported for use in this field, see "Allowable Syntax" on page 9.

Convert flag fields. Specifies how flag fields are treated. There are two options: **Strings to factor, Integers and Reals to double**, and **Logical values (True, False)**. If you select **Logical values (True, False)** the original values of the flag fields are lost. For example, if a field has values Male and Female, these are changed to True and False.

Convert missing values to the R 'not available' value (NA). When selected, any missing values are converted to the R `NA` value. The value `NA` is used by R to identify missing values. Some R functions that you use might have an argument that can be used to control how the function behaves when the data contain `NA`. For example, the function might allow you to choose to automatically exclude records that contain `NA`. If this option is not selected, any missing values are passed to R unchanged, and might cause errors when your R script is executed.

Convert date/time fields to R classes with special control for time zones. When selected, variables with date or datetime formats are converted to R date/time objects. You must select one of the following options:

- **R POSIXct.** Variables with date or datetime formats are converted to R `POSIXct` objects.
- **R POSIXlt (list).** Variables with date or datetime formats are converted to R `POSIXlt` objects.

Note: The POSIX formats are advanced options. Use these options only if your R script specifies that datetime fields are treated in ways that require these formats. The POSIX formats do not apply to variables with time formats.

Run. To execute the R Output node, click **Run**. The output objects are added to the Output manager, or optionally to the file specified in the **Filename** field on the **Output** tab.

R Output Node - Console Output Tab: The **Console Output** tab contains any output that is received from the R console when the R script in the **R Output syntax** field on the **Syntax** tab is executed. This output might include R error messages or warnings that are produced when the R script is executed. The output can be used, primarily, to debug the R script. The **Console Output** tab also contains the R script from the **R Output syntax** field. Every time the R transform script is executed, the content of the **Console Output** tab is overwritten with the output received from the R console. The console output cannot be edited.

R Output Node - Output Tab: **Output name.** Specifies the name of the output that is produced when the node is executed. When **Auto** is selected, the name of the output is automatically set to "R Output". Optionally, you can select **Custom** to specify a different name.

Output to screen. Select this option to generate and display the output in a new window. The output is also added to the Output manager.

Output to file. Select this option to save the output to a file. Doing so enables the **Output Graph** and **Output File** radio buttons.

Output Graph. Only enabled if **Output to file** is selected. Select this option to save any graphs that result from executing the R Output node to a file. Specify a filename to use for the generated output in the **Filename** field. Use the ellipses (**...**) to specify a specific file and location. Specify the file type in the **File type** drop-down list. The following file types are available:
- Output object (`.cou`)
- HTML (`.html`)

Output Text. Only enabled if **Output to file** is selected. Select this option to save any text output that results from executing the R Output node to a file. Specify a filename to use for the generated output in the **Filename** field. Use the ellipses (**...**) to specify a specific file and location. Specify the file type in the **File type** drop-down list. The following file types are available:
- Output object (`.cou`)
- Text document (`.txt`)
- HTML (`.html`)

R Output Browser: If **Output to screen** is selected on the **Output** tab of the R Output node dialog box, on-screen output is displayed in an output browser window. The output is also added to the Output manager. The output browser window has its own set of menus that allow you to print or save the output, or export it to another format. The **Edit** menu only contains the **Copy** option. The output browser of the R Output node has two tabs; the **Text Output** tab that displays text output, and the **Graph Output** tab that displays graphs and charts.

If **Output to file** is selected on the **Output** tab of the R Output node dialog box, the output browser window is not displayed upon successful execution of the R Output node.

R Output Browser - Text Output Tab: The **Text Output** tab displays any text output that is generated when the R script in the **R Output syntax** field on the **Syntax** tab of the R Output node is executed.

Note: R error messages or warnings that result from executing your R output script are always displayed on the **Console Output** tab of the R Output node.

R Output Browser - Graph Output Tab: The **Graph Output** tab displays any graphs or charts that are generated when the R script in the **R Output syntax** field on the **Syntax** tab of the R Output node is executed. For example, if your R script contains a call to the R `plot` function, the resulting graph is displayed on this tab.

Allowable Syntax

Within the syntax field on the **Syntax** tab of the R Transform, R Building, and R Output nodes, only statements and functions that are recognized by R are allowed.

For the R Transform node and the R Scoring nugget, data passes through the R script (in batch). For this reason, R scripts for model scoring and process nodes should not include operations that span or combine rows in the data, such as sorting or aggregation. This limitation is imposed to ensure that data can be split up in a Hadoop environment, and during in-database mining. R output and R model building nodes do not have this limitation.

Starting with SPSS Modeler version 17.1, the addition of a non-batch data transfer mode, in both the R Transform node and the R Scoring nugget, means that you can either span or combine rows in the data in SPSS Modeler Server.

All R nodes can be seen as independent global R environments. Therefore, using `library` functions within the two separate R nodes requires the loading of the R library in both R scripts.

To display the value of an R object that is defined in your R script, you must include a call to a printing function. For example, to display the value of an R object that is called `data`, include the following line in your R script:

```
print(data)
```

You cannot include a call to the R `setwd` function in your R script because this function is used by IBM SPSS Modeler to control the file path of the R scripts output file.

Stream parameters that are defined for use in CLEM expressions and scripting are not recognized if used in R scripts.

Debugging R Scripts

When working with the R nodes it is possible to do limited debugging of your R script using R commands such as `print()` and `str()` to examine variables and R objects. However, as your scripts become more complex and involve function calling you might want to debug your R script in an interactive R environment. A simple approach would be to have the R node write the data and metadata it is receiving at that point in the stream to a file, for example:

```
save(modelerData, file="data.rda")
save(modelerDataModel, file="metadata.rda")
```

You can then launch R outside of Modeler and load the data and metadata. An R script could then be written, and debugged using standard R debugging functions such as `browser()`, `debug ()` and `traceback ()`. Once the code is working as expected, it can be copied and pasted back into the node.

Examples

Example: Data processing: In this example, the R Transform node is used to implement a custom R algorithm that adds one day to a given date.

1. Add a User Input node, from the Sources palette, to the stream canvas.
2. Double-click the User Input node to open the node dialog box.
3. In the table, enter `dob` in the Field cell, select **Date** in the Storage cell, and enter `2001-01-01` in the Values cell.
4. Click **OK** to close the User Input node.
5. Add an R Transform node, from the Record Ops palette, to the stream canvas and connect it to the User Input node.
6. Double-click the R Transform node to open the node dialog box.
7. In the **R Transform Syntax** field on the **Syntax** tab, enter the following R script:

```
day<-as.Date(modelerData$dob, format="%Y-%m-%d")
next_day<-day + 1
modelerData<-cbind(modelerData,next_day)
var1<-c(fieldName="Next day",fieldLabel="",fieldStorage="date",fieldMeasure="",fieldFormat="",
fieldRole="")
modelerDataModel<-data.frame(modelerDataModel,var1)
```

When the R Transform node is executed, the following R objects are created or updated:

- The R object `modelerData` is automatically populated with the data from the User Input node.
- The R object `day` contains the date in the format stated in the `as.Date` function.
- The R object `next_day` contains the date with one day added to it.
- The R object `modelerData` is updated with an extra field that contains the date held in `next_day`.
- The R object `var1` sets up a new field for the data model that describes the type and structure of the new field in `modelerData`.
- The R object `modelerDataModel` contains the data model for the original data with an extra field for the new field in `modelerData`.

8. Select **Convert date/time fields to R classes with special control for time zones**. Keep the default option **POSIXct** selected.
9. Add a Type node, from the Field Ops palette, to the stream canvas and connect it to the R Transform node.
10. Add a Table node, from the Output palette, to the stream canvas.
11. To see the result of executing the R script in the R Transform node, connect the Table node to the Type node, double-click the Table node, and click **Run**.
12. The table contains the original date, and the new date in the field named *Next day*; this field was created by the R script.

Example: Model building and scoring: In this example, a linear model is fitted to the example data set DRUG1n, using the variable Age as the model input field and the variable Na as the model target field. The linear model is then used to score the same data set.

1. Add a Variable File node, from the Sources palette, to the stream canvas.
2. Double-click the Variable File node to open the node dialog box.
3. Click the ellipsis button (...) to the right of the **File** field to select the DRUG1n data set. The file that contains the DRUG1n data set can be found in the **Demos** folder.
4. Click **OK** to close the Variable File node.
5. Add an R Building node, from the Modeling palette, to the stream canvas and connect it to the Variable File node.
6. Double-click the R Building node to open the node dialog box.
7. In the **R model building syntax** field on the **Syntax** tab, enter the following R script:

   ```
   modelerModel<-lm(Na~Age,data=modelerData)
   plot(x=modelerData$Age,y=modelerData$Na,xlab="Age",ylab="Na")
   cor(modelerData$Na,modelerData$Age)
   ```

   The R object `modelerData` is automatically populated with the DRUG1n data set.

   When the node is executed, the R object `modelerModel` contains the results of the linear model analysis.
8. On the **Model Options** tab, select **Display R graphs as HTML**. When the node is executed, a plot of the target field Na against the input field Age is displayed on the **Graph Output** tab of the R model nugget.
9. On the **Model Options** tab, select **Display R text output**. When the node is executed, the correlation between the target field Na and the input field Age is written to the **Text Output** tab of the R model nugget.
10. In the **R model scoring syntax** field on the **Syntax** tab, enter the following R script:

```
result<-predict(modelerModel,newdata=modelerData)
modelerData<-cbind(modelerData,result)
var1<-c(fieldName="NaPrediction",fieldLabel="",fieldStorage="real",fieldMeasure="",fieldFormat="",
fieldRole="")
modelerDataModel<-data.frame(modelerDataModel,var1)
```

When the R model nugget is executed, the following R objects are created:
- The R object `result` contains the predicted values of the target field, Na, obtained from the model `modelerModel`.
- The R object `modelerData` is a data frame that contains the original data with an extra field that contains the predicted values of the target field.
- The R object `var1` sets up a new field for the data model that describes the type and structure of the predicted values of the target field.
- The R object `modelerDataModel` contains the data model for the original data with an extra field for the predicted values of the target field.

11. Click **Run** to execute the R Building node. An R model nugget is added to the Models palette.
12. Add the R model nugget to the stream canvas.
13. Add a Table node, from the Output palette, to the stream canvas.
14. To see the predicted values of the target field, connect the Table node to the R model nugget, double-click the Table node, and click **Run**.
15. The table contains the predicted values in the field named *NaPrediction*; this field was created by the model scoring R script.

Python for Spark

IBM SPSS Modeler supports Python scripts for Apache Spark.

Note:
- There are no standalone nodes for Python for Spark scripting like there are for R nodes.
- Python nodes created from Custom Dialog Builder depend on the Spark environment.
- Python scripts must use the Spark API because data will be presented in the form of a Spark DataFrame.
- Old nodes created in version 17.1 will still only run against IBM SPSS Analytic Server (the data originates from an IBM SPSS Analytic Server source node and has not been extracted to IBM SPSS Modeler Server). New Python and Custom Dialog Builder nodes created in version 18.0 or later can run against IBM SPSS Modeler Server.
- When installing Python, make sure all users have permission to access the Python installation.
- If you want to use the Machine Learning Library (MLlib), you must install a version of Python that includes NumPy. Then you must configure the IBM SPSS Modeler Server (or the local server in IBM SPSS Modeler Client) to use your Python installation. For details, see "Scripting with Python for Spark."

Scripting with Python for Spark

IBM SPSS Modeler can execute Python scripts using the Apache Spark framework to process data. This documentation provides the Python API description for the interfaces provided.

Prerequisites

- If you plan to execute Python/Spark scripts against IBM SPSS Analytic Server, you must have a connection to Analytic Server, and Analytic Server must have access to a compatible installation of Apache Spark. Refer to your IBM SPSS Analytic Server documentation for details about using Apache Spark as the execution engine.

- If you plan to execute Python/Spark scripts against IBM SPSS Modeler Server (or the local server included with IBM SPSS Modeler Client), you must configure it to use your Python installation by adding the following new option to `options.cfg`:

  ```
  # Set to the full path to the python executable (including the executable name) to enable use of
  PySpark.
  eas_pyspark_python_path, ""
  ```

  For example:

  ```
  eas_pyspark_python_path, "C:/Your_Python_Install/python.exe"
  ```
- The IBM SPSS Modeler installation includes a Spark distribution but not a Python distribution. If you want to use the Machine Learning Library (MLlib), you must install a version of Python that includes NumPy.
- When installing Python, make sure all users have permission to access the Python installation.

The IBM SPSS Analytic Server context object

The execution context for a Python/Spark script is defined by an Analytic Server context object. When running against IBM SPSS Modeler Server, the context object is for the embedded version of Analytic Server that is included with the IBM SPSS Modeler Server installation. To obtain the context object, the script must include the following:

```
import spss.pyspark.runtime
asContext = spss.pyspark.runtime.getContext()
```

From the Analytic Server context, you can obtain the Spark context and the SQL context:

```
sparkContext = asc.getSparkContext()
sqlContext = asc.getSparkSqlContext()
```

Refer to your Apache Spark documentation for information about the Spark context and the SQL context.

Accessing data

Data is transferred between a Python/Spark script and the execution context in the form of a Spark SQL DataFrame. A script that consumes data (that is, any node except a source node) must retrieve the data frame from the context:

```
inputData = asContext.getSparkInputData()
```

A script that produces data (that is, any node except a terminal node) must return a data frame to the context:

```
asContext.setSparkOutputData(outputData)
```

You can use the SQL context to create an output data frame from an RDD where required:

```
outputData = sqlContext.createDataFrame(rdd)
```

Defining the data model

A node that produces data must also define a data model that describes the fields visible downstream of the node. In Spark SQL terminology, the data model is the schema.

A Python/Spark script defines its output data model in the form of a `pyspsark.sql.types.StructType` object. A `StructType` describes a row in the output data frame and is constructed from a list of `StructField` objects. Each `StructField` describes a single field in the output data model.

You can obtain the data model for the input data using the `:schema` attribute of the input data frame:

```
inputSchema = inputData.schema
```

Fields that are passed through unchanged can be copied from the input data model to the output data model. Fields that are new or modified in the output data model can be created using the `StructField` constructor:

```
field = StructField(name, dataType, nullable=True, metadata=None)
```

Refer to your Spark documentation for information about the constructor.

You must provide at least the field name and its data type. Optionally, you can specify metadata to provide a measure, role, and description for the field (see “Data metadata” on page 16).

DataModelOnly mode

IBM SPSS Modeler needs to know the output data model for a node, before the node is executed, in order to enable downstream editing. To obtain the output data model for a Python/Spark node, IBM SPSS Modeler executes the script in a special "data model only" mode where there is no data available. The script can identify this mode using the `isComputeDataModelOnly` method on the Analytic Server context object.

The script for a transformation node can follow this general pattern:

```
if asContext.isComputeDataModelOnly():
        inputSchema = asContext.getSparkInputSchema()
        outputSchema = ... # construct the output data model
        asContext.setSparkOutputSchema(outputSchema)
else:
        inputData = asContext.getSparkInputData()
        outputData = ... # construct the output data frame
        asContext.setSparkOutputData(outputData)
```

Building a model

A node that builds a model must return to the execution context some content that describes the model sufficiently that the node which applies the model can recreate it exactly at a later time.

Model content is defined in terms of key/value pairs where the meaning of the keys and the values is known only to the build and score nodes and is not interpreted by Modeler in any way. Optionally the node may assign a MIME type to a value with the intent that Modeler might display those values which have known types to the user in the model nugget.

A value in this context may be PMML, HTML, an image, etc. To add a value to the model content (in the build script):

```
asContext.setModelContentFromString(key, value, mimeType=None)
```

To retrieve a value from the model content (in the score script):

```
value = asContext.getModelContentToString(key)
```

As a shortcut, where a model or part of a model is stored to a file or folder in the file system you can bundle all the content stored to that location in one call (in the build script):

```
asContext.setModelContentFromPath(key, path)
```

Note that in this case there is no option to specify a MIME type because the bundle may contain various content types.

If you need a temporary location to store the content while building the model you can obtain an appropriate location from the context:

```
path = asContext.createTemporaryFolder()
```

To retrieve existing content to a temporary location in the file system (in the score script):

```
path = asContext.getModelContentToPath(key)
```

Error handling

To raise errors, throw an exception from the script and display it to the IBM SPSS Modeler user. Some exceptions are predefined in the module `spss.pyspark.exceptions`. For example:

```
from spss.pyspark.exceptions import ASContextException
if ... some error condition ...:
     raise ASContextException("message to display to user")
```

Analytic Server Context

The Context provides support for the Analytic Server context interface for interaction with IBM SPSS Analytic Server.

AnalyticServerContext Objects

`AnalyticServerContext` Objects set up the context environment which provides several interfaces for interacting with IBM SPSS Analytic Server. An application that wants to construct this context instance must do so using the `spss.pyspark.runtime.getContext()` interface rather than implementing the interface directly.

Returns the Pyspark python `SparkContext` instance:

```
cxt.getSparkContext() : SparkContext
```

Returns the Pyspark python `SQLContext` instance:

```
cxt.getSparkSQLContext() : SQLContext
```

Returns `True` to describe whether the execution is made only to compute the output data model. Otherwise returns `False`:

```
cxt.isComputeDataModelOnly() : Boolean
```

Returns `True` if the script is running in the Spark environment. Currently, it always returns `True`:

```
cxt.isSparkExecution() : Boolean
```

Loads input data from the upstream temporary file and generates the `pyspark.sql.DataFrame` instance:

```
cxt.getSparkInputData() : DataFrame
```

Returns a `pyspark.sql.StructType` instance generated from the input data model. Returns `None` if the input data model does not exist:

```
cxt.getSparkInputSchema() : StructType
```

Serializes the output data frame into Analytic Server context and returns the context:

```
cxt.setSparkOutputData( outDF) : AnalyticServerContext
```

Parameter:

- `outDF (DataFrame)` : The output data frame value

Exceptions:

- `DataOutputNotSupported` : If this interface is invoked in the function `pyspark:buildmodel`
- `ASContextException` : If the output data frame is `None`
- `InconsistentOutputDataModel` : The field names and storage type information common to both objects is inconsistent

Converts the `outSchema StructType` instance into a data model, serializes it into the Analytic Server context, and returns the context:

```
cxt.setSparkOutputSchema(outSchema) : AnalyticServerContext
```

Parameter:

- `outSchema(StructType)` : The output `StructType` object

Exceptions:

- `ASContextException` : If the output schema instance is `None`
- `InconsistentOutputDataModel` : The field names and storage type information common to both objects is inconsistent

Stores the location of model building output to the Analytic Server context and returns the context:

```
cxt.setModelContentFromPath(key, path, mimetype=None) : AnalyticServerContext
```

The path can be a directory path which should use the `cxt.createTemporaryFolder()` API to generate , when everything under the directory is packaged up as model content.

Parameters:

- `key (string)` : key string value
- `path (string)` : location of model building output string path
- `mimetype (string, optional)` : the MIME type of the content

Exceptions:

- `ModelOutputNotSupported` : When not invoking this API from the `pyspark:buildmodel` function
- `KeyError` : If the key attribute is `None` or the string is empty

Stores the model building content, metadata, or other attributes to the Analytic Server context and returns the context:

```
cxt.setModelContentFromString(key, value, mimetype=None) : AnalyticServerContext
```

Parameters:

- `key (string)` : key string value
- `value (string)` : the model metadata string value
- `mimetype (string, optional)` : the MIME type of the content

Exceptions:

- `ModelOutputNotSupported` : When not invoking this API from the `pyspark:buildmodel` function
- `KeyError` : If the key attribute is `None` or the string is empty

Returns the temporary folder location that is managed by Analytic Server; this can be used to store the model content:

```
cxt.createTemporaryFolder() : string
```

Exception:

- `ModelOutputNotSupported` : When not invoking this API from the `pyspark:buildmodel` function

Returns the location of the model which matches the input key:

```
cxt.getModelContentToPath(key) : string
```

Parameter:

- key (string) : key string value

Exceptions:
- ModelInputNotSupported : When not invoking this API from the pyspark:applymodel function
- KeyError : If the key attribute is None or the string is empty
- IncompatibleModelContentType : If the model content type is not a container

Returns the model content, metadata of the model, or other model attributes which match the input key:

```
cxt.getModelContentToString(key) : string
```

Parameter:
- key (string) : key string value

Exceptions:
- ModelInputNotSupported : When not invoking this API from the pyspark:applymodel function
- KeyError : If the key attribute is None, or the string is empty, or the key does not exist
- IncompatibleModelContentType : If the model content type is not consistent

Returns the mime-type assigned to the input key. It returns None if the specified content has no mime type:

```
cxt.getModelContentMimeType(key) : string
```

Parameter:
- key (string) : key string value

Exceptions:
- ModelInputNotSupported : When not invoking this API from the pyspark:applymodel function
- KeyError : If the key attribute is None, or the string is empty, or the key does not exist

Data metadata

This section describes how to set up the data model attributes based on pyspark.sql.StructField.

spss.datamodel.Role Objects

This class enumerates valid roles for each field in a data model.

BOTH: Indicates that this field can be either an antecedent or a consequent.

FREQWEIGHT: Indicates that this field is used to be as frequency weight; this is not displayed to the user.

INPUT: Indicates that this field is a predictor or an antecedent.

NONE: Indicates that this field is not used directly during modeling.

TARGET: Indicates that this field is predicted or a consequent.

PARTITION: Indicates that this field is used to identify the data partition.

RECORDID: Indicates that this field is used to identify the record id.

SPLIT: Indicates that this field is used to split the data.

`spss.datamodel.Measure` Objects

This class enumerates measurement levels for fields in a data model.

`UNKNOWN`: Indicates that the measure type is unknown.

`CONTINUOUS`: Indicates that the measure type is continuous.

`NOMINAL`: Indicates that the measure type is nominal.

`FLAG`: Indicates that the field value is one of two values.

`DISCRETE`: Indicates that the field value should be interpreted as a collection of values.

`ORDINAL`: Indicates that the measure type is ordinal.

`TYPELESS`: Indicates that the field can have any value compatible with its storage.

`pyspark.sql.StructField` Objects

Represents a field in a `StructType`. A `StructField` object comprises four fields:
- `name (string)`: name of a `StructField`
- `dataType (pyspark.sql.DataType)`: specific data type
- `nullable (bool)`: if the values of a `StructField` can contain `None` values
- `metadata (dictionary)`: a python dictionary used to store the option attributes

You can use the metadata dictionary instance to store the measure, role, or label attribute for the specific field. The key words for these attributes are:
- `measure`: the key word for `measure` attribute
- `role`: the key word for `role` attribute
- `displayLabel`: the key word for `label` attribute

Example:
```
from spss.datamodel.Role import Role
from spss.datamodel.Measure import Measure
_metadata = {}
_metadata['measure'] = Measure.TYPELESS
_metadata['role'] = Role.NONE
_metadata['displayLabel'] = "field label description"
StructField("userName", StringType(), nullable=False,
metadata=_metadata)
```

Date, time, timestamp

For operations that use date, time, or timestamp type data, the value is converted to the real value based on the value `1970-01-01:00:00:00` (using Coordinated Universal Time).

For the date, the value represents the number of days, based on the value `1970-01-01` (using Coordinated Universal Time).

For the time, the value represents the number of seconds at 24 hours.

For the timestamp, the value represents the number of seconds based on the value `1970-01-01:00:00:00` (using Coordinated Universal Time).

Exceptions

This section describes possible exception instances.

MetadataException Objects

A subclass of python Exception.

This exception is thrown if an error occurs during operate the metadata object.

`UnsupportedOperationException` Objects

A subclass of python Exception.

This exception is thrown if the specific operation does not allow execution.

InconsistentOutputDataModel Objects

A subclass of python Exception.

This Exception is thrown if both `setSparkOutputSchema` and `setSparkOutputData` are invoked but the field names and storage type information common to both objects are inconsistent.

IncompatibleModelContentType Objects

A subclass of python Exception.

This Exception is thrown during the following scenarios:
- Using `setModelContentFormString` to set model but using `getModelContentToPath` to get value
- Using `setModelContentFormPath` to set model but using `getModelContentToString` to get value

DataOutputNotSupported Objects

A subclass of python Exception.

This exception is raised in `setSparkOutputData` in an execution handled by function `pyspark:buildmodel`.

ModelInputNotSupported Objects

A subclass of python Exception.

This exception is only raised if the script does not invoke the `getModelContentPathByKey` and `getModelContentToString` API in the `pyspark:applymodel` function.

ModelOutputNotSupported Objects

A subclass of python Exception.

This exception is only raised if the script does not invoke the `setModelContentFromPath` and `setModelContentFromString` APIs in the `pyspark:buildmodel` function.

ASContextException Objects

A subclass of python Exception.

This exception is thrown if an unexpected runtime exception occurs.

Examples

This section contains Python for Spark scripting examples.

Basic scripting example for processing data

```
import spss.pyspark.runtime
from pyspark.sql.types import *

cxt = spss.pyspark.runtime.getContext()

if  cxt.isComputeDataModelOnly():
        _schema = cxt.getSparkInputSchema()
        cxt.setSparkOutputSchema(_schema)
else:
        _structType = cxt.getSparkInputSchema()
        df = cxt.getSparkInputData()
        _newDF = df.sample(False, 0.01, 1)
        cxt.setSparkOutputData(_newDF)
```

Example model building script, using the LinearRegressionWithSGD algorithm

```
from pyspark.context import SparkContext
from pyspark.sql.context import SQLContext
from pyspark.sql import Row
from pyspark.mllib.regression import
LabeledPoint,LinearRegressionWithSGD, LinearRegressionModel
from pyspark.mllib.linalg import DenseVector
import numpy
import json

import spss.pyspark.runtime
from spss.pyspark.exceptions import ASContextException

ascontext = spss.pyspark.runtime.getContext()
sc = ascontext.getSparkContext()
df = ascontext.getSparkInputData()

# field settings amd algorithm parameters

target = '%%target_field%%'
predictors = [%%predictor_fields%%]
num_iterations=%%num_iterations%%
prediction_field = "$LR-" + target

# save linear regression model to a filesystem path

def save(model, sc, path):
        data =
sc.parallelize([json.dumps({"intercept":model.intercept,"weights":model.weights.tolist()})])
        data.saveAsTextFile(path)

# print model details to stdout

def dump(model,predictors):
        print(prediction_field+" = " + str(model.intercept))
        weights = model.weights.tolist()
        for i in range(0,len(predictors)):
                print("\t+ "+predictors[i]+"*"+ str(weights[i]))

# check that required fields exist in the input data

input_field_names = [ty[0] for ty in df.dtypes[:]]
if target not in input_field_names:
        raise ASContextException("target field "+target+" not found") for predictor in predictors:
        if predictor not in input_field_names:
                raise ASContextException("predictor field "+predictor+" not found")
```

```
# define map function to convert from dataframe Row objects to mllib LabeledPoint

def row2LabeledPoint(target,predictors,row):
        pvals = []
        for predictor in predictors:
                pval = getattr(row,predictor)
                pvals.append(float(pval))
        tval = getattr(row,target)
        return LabeledPoint(float(tval),DenseVector(pvals))

# convert dataframe to an RDD containing LabeledPoint

training_points = df.rdd.map(lambda row:
row2LabeledPoint(target,predictors,row))

# build the model

model = LinearRegressionWithSGD.train(training_points,num_iterations,intercept=True)

# write a text description of the model to stdout

dump(model,predictors)

# save the model to the filesystem and store into the output model content

modelpath = ascontext.createTemporaryFolder()
save(model,sc,modelpath)
ascontext.setModelContentFromPath("model",modelpath)
```

Example model scoring script, using the LinearRegressionWithSGD algorithm

```
import json
import spss.pyspark.runtime
from pyspark.sql import Row
from pyspark.mllib.regression import
LabeledPoint,LinearRegressionWithSGD, LinearRegressionModel
from pyspark.mllib.linalg import DenseVector
from pyspark.sql.context import SQLContext
import numpy
from pyspark.sql.types import DoubleType, StructField

ascontext = spss.pyspark.runtime.getContext()
sc = ascontext.getSparkContext()

prediction_field = "$LR-" + '%%target_field%%'
predictors = [%%predictor_fields%%]

# compute the output schema by adding the prediction field
outputSchema = ascontext.getSparkInputSchema()
outputSchema.fields.append(StructField(prediction_field,
DoubleType(), nullable=True))

# make a prediction based on a regression model and Dataframe Row object
# return a list containing the input row values and the predicted value
def predict(row,model,predictors,infields,prediction_field_name):
        pvals = []
        rdict = row.asDict()
        for predictor in predictors:
                pvals.append(float(rdict[predictor]))
        estimate = float(model.predict(pvals))
        result = []
        for field in infields:
                result.append(rdict[field])
        result.append(estimate)
        return result

# load a serialized model from the filesystem
```

```
def load(sc, path):
        js = sc.textFile(path).take(1)[0]
        obj = json.loads(js)
        weights = numpy.array(obj["weights"])
        intercept = obj["intercept"]
        return LinearRegressionModel(weights,intercept)

ascontext.setSparkOutputSchema(outputSchema)

if not ascontext.isComputeDataModelOnly():
        # score the data in the input data frame
        indf = ascontext.getSparkInputData()

        model_path = ascontext.getModelContentToPath("model")
        model = load(sc,model_path)

        # compute the scores
        infield_names = [ty[0] for ty in indf.dtypes[:]]
        scores_rdd = indf.rdd.map(lambda row:predict(row,model,predictors,infield_names,prediction_field))

        # create an output DataFrame containing the scores
        sqlCtx = SQLContext(sc)
        outdf = sqlCtx.createDataFrame(scores_rdd,schema=outputSchema)

        # return the output DataFrame as the result
        ascontext.setSparkOutputData(outdf)
```

Chapter 2. Extensions

Extensions are custom components that extend the capabilities of IBM SPSS Modeler. Extensions are packaged in extension bundles (`.mpe` files) and are installed to IBM SPSS Modeler. Extensions can be created by any user and shared with other users by sharing the associated extension bundle.

The following utilities are provided for working with extensions:

- The "Extension Hub," which is accessed from **Extensions** > **Extension Hub**, is an interface for searching for, downloading, and installing extensions from the IBM SPSS Predictive Analytics collection on GitHub. From the Extension Hub dialog, you can also view details of the extensions that are installed on your computer, get updates for installed extensions, and remove extensions.
- You can install an extension bundle that is stored on your local computer from **Extensions** > **Install Local Extension Bundle**.
- You can use the Custom Dialog Builder for Extensions to create an extension that includes a user interface, which is referred to as a custom node dialog. Custom node dialogs generate R script or Python for Spark script that carries out the tasks that are associated with the extension. You design the generated script as part of designing the custom dialog.

Extension Hub

From the Extension Hub dialog, you can do the following tasks:

- Explore extensions that are available from the IBM SPSS Predictive Analytics collection on GitHub. You can select extensions to install now or you can download selected extensions and install them later.
- Get updated versions of extensions that are already installed on your computer.
- View details about the extensions that are already installed on your computer.
- Remove extensions that are installed on your computer.

To download or remove extensions:

1. From the menus, choose: **Extensions** > **Extension Hub**
2. Select the extensions that you want to download or remove and click **OK**. All selections that are made on the Explore and Installed tabs are processed when you click **OK**.

By default, the extensions that are selected for download are downloaded and installed on your computer. From the Settings tab, you can choose to download the selected extensions to a specified location without installing them. You can then install them later by choosing **Extensions** > **Install Local Extension Bundle**. For Windows, you can install an extension by double-clicking the extension bundle file.

Important:

- For Windows 7 and later, installing an updated version of an existing extension bundle might require running IBM SPSS Modeler with administrator privileges. You can start IBM SPSS Modeler with administrator privileges by right-clicking the icon for IBM SPSS Modeler and choosing **Run as administrator**. In particular, if you receive an error message that states that one or more extension bundles could not be installed, then try running with administrator privileges.

Note: The license that you agree to when you install an extension can be viewed at any later time by clicking **More info...** for the extension on the Installed tab.

Explore tab

The Explore tab displays all of the extensions that are available from the IBM SPSS Predictive Analytics collection on GitHub (https:/ibmpredictiveanalytics.github.io/). From the Explore tab, you can select new extensions to download and install, and you can select updates for extensions that are already installed on your computer. The Explore tab requires an internet connection.

- For each extension, the number of the latest version and the associated date of that version are displayed. A brief summary of the extension is also provided. For extensions that are already installed on your computer, the installed version number is also displayed.
- You can view detailed information about an extension by clicking **More info**. When an update is available, **More info** displays information about the update.
- You can view the prerequisites for running an extension, such as whether the IBM SPSS Modeler - Integration Plug-in for R is required, by clicking **Prerequisites**. When an update is available, **Prerequisites** displays information about the update.

Refine by

You can refine the set of extensions that are displayed. You can refine by general categories of extensions, the language in which the extension is implemented, the type of organization that provided the extension, or the state of the extension. For each group, such as Category, you can select multiple items by which to refine the displayed list of extensions. You can also refine by search terms. Searches are not case-sensitive, and the asterisk (*) is treated as any other character and does not indicate a wildcard search.

- To refine the displayed list of extensions, click **Apply**. Pressing the enter key when the cursor is in the **Search** box has the same effect as clicking **Apply**.
- To reset the list to display all available extensions, delete any text in the **Search** box, deselect all items, and click **Apply**.

How to Get Integration Plug-Ins

To get the IBM SPSS Modeler - Integration Plug-in for R:

Install IBM SPSS Modeler - Essentials for R, available from https://github.com/IBMPredictiveAnalytics/R_Essentials_Modeler/releases/ or the IBM SPSS Predictive Analytics community at https://developer.ibm.com/predictiveanalytics/predictive-extensions/. IBM SPSS Modeler - Essentials for R includes the IBM SPSS Modeler - Integration Plug-in for R. Essentials for R does not include the R programming language. Before installing IBM SPSS Modeler - Essentials for R you will need to install R version 3.2 if it is not already installed. It is available from http://www.r-project.org/. It is recommended to download and install R 3.2.2.

Note: If you are installing Essentials for R on a computer that does not have internet access and you plan to use the R scripts that are included with Essentials for R, then you must obtain any R packages that are required by those scripts and manually install them in R. To determine which R packages are required for a specific R script, open the Extension Hub dialog (**Extensions** > **Extension Hub**), click the **Installed** tab, and then click **More info** for the desired extension. The required R packages are listed on the Extension Details dialog. R packages can be obtained from any of the R CRAN mirror sites, which are accessed from http://www.r-project.org/. Be sure to obtain the versions of the packages that match your R version. The version-specific packages are available from links on the "Contributed Packages" page of the CRAN mirror site.

Installed tab

The Installed tab displays all of the extensions that are installed on your computer. From the Installed tab, you can select updates for installed extensions that are available from the IBM SPSS Predictive Analytics collection on GitHub and you can remove extensions. To get updates for installed extensions, you must have an internet connection.

- For each extension, the installed version number is displayed. When an internet connection is available, the number of the latest version and the associated date of that version are displayed. A brief summary of the extension is also provided.
- You can view detailed information about an extension by clicking **More info**. When an update is available, **More info** displays information about the update.
- You can view the prerequisites for running an extension, such as whether the IBM SPSS Modeler - Integration Plug-in for R is required, by clicking **Prerequisites**. When an update is available, **Prerequisites** displays information about the update.

Refine by

You can refine the set of extensions that are displayed. You can refine by general categories of extensions, the language in which the extension is implemented, the type of organization that provided the extension, or the state of the extension. For each group, such as Category, you can select multiple items by which to refine the displayed list of extensions. You can also refine by search terms. Searches are not case-sensitive, and the asterisk (*) is treated as any other character and does not indicate a wildcard search.

- To refine the displayed list of extensions, click **Apply**. Pressing the enter key when the cursor is in the **Search** box has the same effect as clicking **Apply**.
- To reset the list to display all available extensions, delete any text in the **Search** box, deselect all items, and click **Apply**.

Private extensions

Private extensions are extensions that are installed on your computer but are not available from the IBM SPSS Predictive Analytics collection on GitHub. The features for refining the set of displayed extensions and for viewing the prerequisites to run an extension are not available for private extensions.

Note: When using the Extension Hub without an internet connection, some of the features of the Installed tab might not be available.

Settings

The Settings tab specifies whether extensions that are selected for download are downloaded and then installed or downloaded but not installed. This setting applies to new extensions and updates to existing extensions. You might choose to download extensions without installing them if you are downloading extensions to distribute to other users within your organization. You might also choose to download, but not install, extensions if you don't have the prerequisites for running the extensions but plan to get the prerequisites.

If you choose to download extensions without installing them, you can install them later by choosing **Extensions** > **Install Local Extension Bundle**. For Windows, you can install an extension by double-clicking the extension bundle file.

Extension Details

The Extension Details dialog box displays the information that was provided by the author of the extension. In addition to required information, such as Summary, and Version, the author might have included URLs to locations of relevance, such as the author's home page. If the extension was downloaded from the Extension Hub, then it includes a license that can be viewed by clicking **View license**.

Custom Nodes. The Custom Nodes table lists the custom node dialogs that are included in the extension.

Note: Installing an extension that contains a custom node dialog might require a restart of IBM SPSS Modeler to see the entry for the node dialog in the Custom Nodes table.

Dependencies. The Dependencies group lists add-ons that are required to run the components included in the extension.

- **Integration Plug-In for R.** The components for an extension may require the Integration Plug-in for R.
- **R packages.** Lists any R packages that are required by the extension. See the topic "Required R packages" on page 27 for more information.

Installing local extension bundles

To install an extension bundle that is stored on your local computer:

1. From the menus choose:

   Extensions > **Install Local Extension Bundle...**
2. Select the extension bundle. Extension bundles have a file type of `mpe`.

Important: For users of Windows 7 and later versions of Windows, installing an updated version of an existing extension bundle might require running IBM SPSS Modeler with administrator privileges. You can start IBM SPSS Modeler with administrator privileges by right-clicking the icon for IBM SPSS Modeler and choosing **Run as administrator**. In particular, if you receive an error message that states that one or more extension bundles could not be installed, then try running with administrator privileges.

Installation locations for extensions

By default, extensions are installed to a general user-writable location for your operating system.

You can override the default location by defining a path with the `IBM_SPSS_MODELER_EXTENSIONS_PATH` environment variable. The specified location must exist on the target computer. After you set `IBM_SPSS_MODELER_EXTENSIONS_PATH`, you must restart IBM SPSS Modeler for the changes to take effect.

To create an environment variable on Windows, from the Control Panel:

Windows 7

1. Select User Accounts.
2. Select **Change my environment variables**.
3. Click **New**, enter the name of the environment variable (for instance, `IBM_SPSS_MODELER_EXTENSIONS_PATH`) in the **Variable name** field and enter the path or paths in the Variable value field.

Windows 8 or Later

1. Select System.
2. Select the Advanced tab and click **Environment Variables**. The Advanced tab is accessed from Advanced system settings.
3. In the User variables section, click **New**, enter the name of the environment variable (for instance, `IBM_SPSS_MODELER_EXTENSIONS_PATH`) in the **Variable name** field and enter the path or paths in the Variable value field.

Important: For users of Windows 7 and later versions of Windows, installing an updated version of an existing extension bundle might require running IBM SPSS Modeler with administrator privileges. You can start IBM SPSS Modeler with administrator privileges by right-clicking the icon for IBM SPSS Modeler and choosing **Run as administrator**. In particular, if you receive an error message that states that one or more extension bundles could not be installed, then try running with administrator privileges.

Required R packages

If you do not have internet access, you will need to obtain any required R packages for a particular extension, that are not found on your computer, from someone who does. You can view the list of required R packages from the Extension Details dialog box, once the extension is installed. See the topic "Extension Details" on page 25 for more information. Packages can be downloaded from and then installed from within R. For details, see the *R Installation and Administration* guide, distributed with R.

Note: For UNIX (including Linux) users, packages are downloaded in source form and then compiled. This requires that you have the appropriate tools installed on your machine. See the *R Installation and Administration* guide for details. In particular, Debian users should install the `r-base-dev` package from `apt-get install r-base-dev`.

Creating and managing custom nodes

The Custom Dialog Builder for Extensions creates nodes to use inside SPSS Modeler streams.

Using the Custom Dialog Builder for Extensions you can:

- Create a custom node dialog for executing a node that is implemented in R, or in Apache Spark (via Python). See the topic "Building the script template" on page 29 for more information.
- Open a file containing the specification for a custom node dialog--perhaps created by another user--and add the dialog to your installation of IBM SPSS Modeler, optionally making your own modifications.
- Save the specification for a custom node dialog so that other users can add it to their installations of IBM SPSS Modeler.

In the Custom Dialog Builder for Extensions, you create or modify custom node dialogs within extensions. When you open the Custom Dialog Builder for Extensions, a new extension that contains an empty custom node dialog is created. When you save or install custom node dialogs from the Custom Dialog Builder for Extensions, they are saved or installed as part of an extension.

Note:

- You cannot create your own version of a node dialog for a standard IBM SPSS Modeler node.
- Scripting is not supported for nodes that are created with Custom Dialog Builder, including Custom Dialog Builder R nodes and Custom Dialog Builder Python nodes.

How to start the Custom Dialog Builder for Extensions

From the menus, choose **Extensions** > **Custom Node Dialog Builder**

Note:

- There are no standalone nodes for Python for Spark scripting like there are for R nodes.
- Python nodes created from Custom Dialog Builder depend on the Spark environment.
- Python scripts must use the Spark API because data will be presented in the form of a Spark DataFrame.
- Old nodes created in version 17.1 will still only run against IBM SPSS Analytic Server (the data originates from an IBM SPSS Analytic Server source node and has not been extracted to IBM SPSS Modeler Server). New Python and Custom Dialog Builder nodes created in version 18.0 or later can run against IBM SPSS Modeler Server.
- When installing Python, make sure all users have permission to access the Python installation.
- If you want to use the Machine Learning Library (MLlib), you must install a version of Python that includes NumPy. Then you must configure the IBM SPSS Modeler Server (or the local server in IBM SPSS Modeler Client) to use your Python installation. For details, see "Scripting with Python for Spark" on page 11.

Custom Dialog Builder layout

Dialog Canvas

The dialog canvas is the area of the Custom Dialog Builder where you design the layout of your node dialog.

Properties Pane

The properties pane is the area of the Custom Dialog Builder where you specify properties of the controls that make up the node dialog as well as properties of the dialog itself, such as the node type.

Tools Palette

The tools palette provides the set of controls that can be included in a custom node dialog. You can show or hide the Tools Palette by choosing Tools Palette from the **View** menu.

Script Template

The Script Template specifies the R script or Python for Spark script that is generated by the custom node dialog. You can move the Script Template pane to a separate window by clicking **Move to New Window**. To move a separate Script Template window back into the Custom Dialog Builder, click **Restore to Main Window**.

Building a custom node dialog

The basic steps involved in building a custom node dialog are:

1. Specify the properties of the node dialog itself, such as the title that appears when the node dialog is launched and the location of the new node within the IBM SPSS Modeler palettes. See the topic "Dialog Properties" for more information.
2. Specify the controls, such as field choosers and check boxes, that make up the node dialog and any sub-dialogs. See the topic "Control types" on page 31 for more information.
3. Create the script template that specifies the R code or Python for Spark code that is generated by the node dialog. See the topic "Building the script template" on page 29 for more information.
4. Specify properties of the extension that contains your nodedialog. See the topic "Extension Properties" on page 48 for more information.
5. Install the extension that contains the node dialog to IBM SPSS Modeler and/or save the extension to an extension bundle (`.mpe`) file. See the topic "Managing custom node dialogs" on page 51 for more information.

You can preview your node dialog as you're building it. See the topic "Previewing a custom node dialog" on page 31 for more information.

Dialog Properties

The Custom Dialog Builder window shows the properties for the node dialog and for the selected user interface control. To view and set Dialog Properties, click on the canvas in an area outside of any controls. With no controls on the canvas, Dialog Properties are always visible.

Dialog Name. The Dialog Name property is required and specifies a unique name to associate with the node dialog. To minimize the possibility of name conflicts, you may want to prefix the name with an identifier for your organization, such as a URL.

Title. The Title property specifies the text to be displayed in the title bar of the node dialog box.

Help File. The Help File property is optional and specifies the path to a help file for the node dialog. This is the file that will be launched when the user clicks the **Help** button on the dialog. Help files must be in HTML format. A copy of the specified help file is included with the specifications for the node dialog when the node dialog is installed or saved. The Help button on the run-time dialog is hidden if there is no associated help file.

- Localized versions of the help file that exist in the same directory as the help file are automatically added to the node dialog when you add the help file. Localized versions of the help file are named `<Help File>_<language identifier>.htm`. For more information, see the topic "Creating Localized Versions of Custom Node Dialogs" on page 53.
- Supporting files, such as image files and style sheets, can be added to the node dialog by first saving the node dialog. You then manually add the supporting files to the node dialog file (`.cfe`). For information about accessing and manually modifying custom node dialog files, see the section that is titled "To localize dialog strings" in the topic "Creating Localized Versions of Custom Node Dialogs" on page 53.

Script Type. Specifies the type of script that can be used to build the Script Template. In IBM SPSS Modeler, R scripting or Python for Spark scripting can be used.

Score from the Model. Specifies whether the model that is built using the model building script is to be used for scoring.

Node Type. Specifies the type of node that will be created when you install your node dialog.

Palette. Specifies the palette to which the newly created node will be added when you install your node dialog.

Node Icon. Click the ellipsis (...) button to select an image to be used as the node icon for the newly created node. The image that you choose must be a `.gif` file.

Laying out controls on the dialog canvas

You add controls to a custom node dialog by dragging them from the tools palette onto the dialog canvas. To ensure consistency with built-in node dialogs, the dialog canvas is divided into four functional columns in which you can place controls.

- The first (leftmost) column is primarily intended for a Field Chooser control.
- Sub-dialog buttons must be in the rightmost column (for example, the third column if only three columns are used) and no other controls can be in the same column as Sub-dialog buttons. In that regard, the fourth column can contain only Sub-dialog buttons.

Although not shown on the dialog canvas, when the node dialog is installed into IBM SPSS Modeler, the appropriate buttons are added to the dialog (for example: **OK**, **Cancel**, **Apply**, **Reset** and, if appropriate, **Help** and **Run**). The presence and locations of these buttons is automatic. However, the **Help** button is hidden if there is no help file associated with the node dialog (as specified by the Help File property in Dialog Properties).

You can change the vertical order of the controls within a column by dragging them up or down, but the exact position of the controls is determined automatically for you. At run-time, controls resize in appropriate ways when the dialog itself is resized. Controls such as field choosers automatically expand to fill the available space below them.

Building the script template

The script template specifies the R script or Python for Spark script that the custom node dialog will generate. A single custom node dialog can be used to specify one or more operations which will run in sequence.

The script template might consist of *static text*. Static text is different to the static text control; it is R code or Python for Spark code that is always generated when the node runs. For example, command names and subcommand specifications that don't depend on user input are static text. The script template might also consist of control identifiers that are replaced at run-time with the values of the associated custom node dialog controls. For example, the set of fields specified in a field chooser is represented with the control identifier for the field chooser control.

To build the Script Template

1. For static text that does not depend on user-specified values, enter the R script or Python for Spark script as you would in, for example, the **R model building syntax** field of the R Build node.
2. Add control identifiers of the form `%%Identifier%%` at the locations where you want to insert R script or Python for Spark script generated by controls, where `Identifier` is the value of the Identifier property for the control.
   - You can insert a control identifier by selecting a row in the table of identifiers, right-clicking and selecting **Add to script template**. You can also insert a control identifier by right-clicking a control on the canvas and selecting **Add to script template**.
   - You can also select from a list of available control identifiers by pressing Ctrl+Spacebar. The list contains the control identifiers followed by the items available with the script auto-completion feature.

   If you manually enter identifiers, retain any spaces, since all spaces in identifiers are significant.

   At run-time, and for all controls other than check boxes, check box groups, and the static text control, each identifier is replaced with the current value of the **Script** property of the associated control. If the control is empty at run-time, it does not generate any script. For check boxes and check box groups, the identifier is replaced by the current value of the Checked R Script or Unchecked R Script property of the associated control, depending on the current state of the control--checked or unchecked. See the topic "Control types" on page 31 for more information.

Example: Including run-time values in an R script template

In this example, the custom node dialog will generate and run R script to build and score a linear regression model, using a call to the R `lm` function with the signature shown here.

`lm(`*`formula`*`,`*`data`*`)`

- *`formula`* specifies an expression, such as `Na~Age`, where `Na` is the target field of the model, and the input field of the model is `Age`.
- *`data`* is a data frame containing the values of the fields that are specified in the formula.

Consider a custom node dialog with a single field chooser control that allows the user to choose the input field of the linear model. The script template to generate and run the R script that builds the model is entered on the **Script** tab, and might look like this:

```
modelerModel <- lm(Na~%%input%%,data=modelerData)
```

- `%%input%%` is the value of the Identifier property for the field chooser control. At run-time it will be replaced by the current value of the **Script** property of the control.
- Defining the **Script** property of the field chooser control to be `%%ThisValue%%` specifies that at run-time the current value of the property will be the value of the control, which is the field that is chosen from the field chooser.

Suppose the user of the custom node dialog selects the Age field as the input field of the model. The following R script is then generated by the node dialog:

```
modelerModel <- lm(Na~Age,data=modelerData)
```

The script template to generate and run the R script that scores the model is entered on the **Score Script** tab, and might look like this:

```
result <- predict(modelerModel,newdata=modelerData)
var1 <-c(fieldName="predicted", fieldLabel="",fieldStorage="real",fieldMeasure="",fieldFormat="",
fieldRole="")
modelerDataModel<-data.frame(modelerDataModel,var1)
```

This R script does not depend on any user-specified values, only on the model that is built using the model building R script. Therefore, the model scoring R script is entered as it would be in the **R model scoring syntax** field of the R Build node.

Previewing a custom node dialog

You can preview the node dialog that is currently open in the Custom Dialog Builder. The dialog appears and functions as it would when run from a node within IBM SPSS Modeler.

- Field choosers are populated with dummy fields.
- The **OK** button closes the preview.
- If a help file is specified, the **Help** button is enabled and will open the specified file. If no help file is specified, the help button is disabled when previewing, and hidden when the actual dialog is run.

To preview a custom node dialog, from the menus in the Custom Dialog Builder, choose **File** > **Preview Dialog**.

Control types

The tools palette provides all of the standard controls that might be needed in a custom node dialog.

- **Field Chooser:** A list of all the fields from the active dataset. See the topic "Field Chooser" on page 32 for more information.
- **Check Box:** A single check box. See the topic "Check Box" on page 33 for more information.
- **Combo Box:** A combo box for creating drop-down lists. See the topic "Combo Box" on page 33 for more information.
- **List Box:** A list box for creating single selection or multiple selection lists. See the topic "Combo Box" on page 33 for more information.
- **Text control:** A text box that accepts arbitrary text as input. See the topic "Text control" on page 36 for more information.
- **Number control:** A text box that is restricted to numeric values as input. See the topic "Number Control" on page 37 for more information.
- **Date control:** A spinner control for specifying date/time values, which include dates, times, and datetimes. See the topic "Date control" on page 38 for more information.
- **Secured Text:** A text box that masks user entry with asterisks. See the topic "Secured Text" on page 39 for more information.
- **Static Text control:** A control for displaying static text. See the topic "Static Text Control" on page 40 for more information.
- **Color Picker:** A control for specifying a color and generating the associated RGB value. See the topic "Color Picker" on page 40 for more information.
- **Table control:** A table with a fixed number of columns and a variable number of rows that are added at run time. See the topic "Table Control" on page 41 for more information.
- **Item Group:** A container for grouping a set of controls, such as a set of check boxes. See the topic "Item Group" on page 43 for more information.
- **Radio Group:** A group of radio buttons. See the topic "Radio Group" on page 43 for more information.
- **Check Box Group:** A container for a set of controls that are enabled or disabled as a group, by a single check box. See the topic "Check Box Group" on page 45 for more information.
- **File Browser:** A control for browsing the file system to open or save a file. See the topic "File Browser" on page 45 for more information.
- **Tab:** A single tab. See the topic "Tab" on page 46 for more information.

- **Sub-dialog Button:** A button for launching a sub-dialog. See the topic "Sub-dialog button" on page 47 for more information.

Field Chooser

The Field Chooser control displays the list of fields that are available to the end user of the node dialog. You can display all fields from the active dataset (the default) or you can filter the list based on type and measurement level--for instance, numeric fields that have a measurement level of scale. You can also specify any other Field Chooser as the source of fields for the current Field Chooser. The Field Chooser control has the following properties:

Identifier. The unique identifier for the control.

Title. An optional title that appears above the control. For multi-line titles, use `\n` to specify line breaks.

Title Position. Specifies the position of the title relative to the control. Values are Top and Left where Top is the default. This property only applies when the chooser type is set to select a single field.

ToolTip. Optional ToolTip text that appears when the user hovers over the control. The specified text only appears when hovering over the title area of the control. Hovering over one of the listed fields displays the field name and label.

Mnemonic Key. An optional character in the title to use as a keyboard shortcut to the control. The character appears underlined in the title. The shortcut is activated by pressing Alt+[mnemonic key].

Chooser Type. Specifies whether the Field Chooser in the custom node dialog can be used to select a single field or multiple fields from the field list.

Separator Type. Specifies the delimiter between the selected fields in the generated script. The allowed separators are a blank, a comma, and a plus sign (+). You can also enter an arbitrary single character to be used as the separator.

Minimum Fields. The minimum number of fields that must be specified for the control, if any.

Maximum Fields. The maximum number of fields that can be specified for the control, if any.

Required for Execution. Specifies whether a value is required in this control in order for execution to proceed. If **True** is specified, the user of the node dialog must specify a value for the control otherwise clicking the **OK** button will generate an error. If **False** is specified, the absence of a value in this control has no effect on the state of the **OK** button.

Variable Filter. Allows you to filter the set of fields that are displayed in the control. You can filter on field type and measurement level, and you can specify that multiple response sets are included in the field list. Click the ellipsis (...) button to open the Filter dialog. You can also open the Filter dialog by double-clicking the Field Chooser control on the canvas. See the topic "Filtering Field Lists" on page 33 for more information.

Field Source. Specifies that another Field Chooser is the source of fields for the current Field Chooser. When the Field Source property is not set, the source of fields is the active dataset. Click the ellipsis (...) button to open the dialog box and specify the field source.

Script. Specifies the script that is generated and run by this control at run-time and can be inserted in the script template.

- You can specify any valid R script or Python for Spark script. For multi-line scripts or long scripts, click the ellipsis (...) button and enter your script in the Script Property dialog.
- The value `%%ThisValue%%` specifies the run-time value of the control, which is the list of fields. This is the default.

Enabling Rule. Specifies a rule that determines when the current control is enabled. Click the ellipsis (...) button to open the dialog box and specify the rule. The Enabling Rule property is visible only when other controls that can be used to specify an enabling rule exist on the canvas.

Specifying the Field Source for a Field Chooser: The Field Source dialog box specifies the source of the fields that are displayed in the Field Chooser. The source can be any other Field Chooser. You can choose to display the fields that are in the selected control or the fields, from the active dataset, that are not in the selected control.

Filtering Field Lists

The Filter dialog box, associated with field chooser controls, allows you to filter the types of fields from the active dataset that can appear in the lists. You can also specify whether multiple response sets associated with the active dataset are included. Numeric fields include all numeric formats except date and time formats.

Check Box

The Check Box control is a simple check box that can generate and run different R scripts or Python for Spark scripts for the checked versus the unchecked state. The Check Box control has the following properties:

Identifier. The unique identifier for the control.

Title. An optional title that appears above the control. For multi-line titles, use `\n` to specify line breaks.

ToolTip. Optional ToolTip text that appears when the user hovers over the control.

Mnemonic Key. An optional character in the title to use as a keyboard shortcut to the control. The character appears underlined in the title. The shortcut is activated by pressing Alt+[mnemonic key].

Default Value. The default state of the check box--checked or unchecked.

Checked/Unchecked Script. Specifies the R script or Python for Spark script that is generated and run when the control is checked and when it is unchecked. To include the script in the script template, use the value of the Identifier property. The generated script, whether from the Checked Script or Unchecked Script property, will be inserted at the specified position(s) of the identifier. For example, if the identifier is *checkbox1*, then at run time, instances of `%%checkbox1%%` in the script template will be replaced by the value of the Checked Script property when the box is checked and the Unchecked Script property when the box is unchecked.

- You can specify any valid R script or Python for Spark script. For multi-line scripts or long scripts, click the ellipsis (...) button and enter your script in the Script Property dialog.

Enabling Rule. Specifies a rule that determines when the current control is enabled. Click the ellipsis (...) button to open the dialog box and specify the rule. The Enabling Rule property is visible only when other controls that can be used to specify an enabling rule exist on the canvas.

Combo Box

The Combo Box control allows you to create a drop-down list that can generate and run R script or Python for Spark script specific to the selected list item. It is limited to single selection. The Combo Box control has the following properties:

Identifier. The unique identifier for the control. This is the identifier to use when referencing the control in the script template.

Title. An optional title that appears above the control. For multi-line titles, use `\n` to specify line breaks.

Title Position. Specifies the position of the title relative to the control. Values are Top and Left where Top is the default.

ToolTip. Optional ToolTip text that appears when the user hovers over the control.

List Items. Click the ellipsis (...) button to open the List Item Properties dialog box, which allows you to specify the list items of the control. You can also open the List Item Properties dialog by double-clicking the Combo Box control on the canvas.

Mnemonic Key. An optional character in the title to use as a keyboard shortcut to the control. The character appears underlined in the title. The shortcut is activated by pressing Alt+[mnemonic key].

Editable. Specifies whether the Combo Box control is editable. When the control is editable, a custom value can be entered at run time.

Script. Specifies the R script or Python for Spark script that is generated by this control at run time and can be inserted in the script template.

- The value `%%ThisValue%%` specifies the run time value of the control and is the default. If the list items are manually defined, the run time value is the value of the Script property for the selected list item. If the list items are based on a target list control, the run time value is the value of the selected list item. For multiple selection list box controls, the run time value is a blank-separated list of the selected items. See the topic "Specifying list items for combo boxes and list boxes" for more information.
- You can specify any valid R script or Python for Spark script. For multi-line scripts or long scripts, click the ellipsis (...) button and enter your script in the Script Property dialog.

Quote Handling. Specifies handling of quotation marks in the run time value of `%%ThisValue%%` when the Script property contains `%%ThisValue%%` as part of a quoted string. In this context, a quoted string is a string that is enclosed in single quotation marks or double quotation marks. Quote handling applies only to quotation marks that are the same type as the quotation marks that enclose `%%ThisValue%%`. The following types of quote handling are available.

Python
Quotation marks in the run time value of %%ThisValue%% that match the enclosing quotation marks are escaped with the backslash character (\). For example, if the Script property is '%%ThisValue%%' and the run time value of the combo box is `Combo box's value`, then the generated script is `'Combo box\'s value'`. Note that quote handling is not done when %%ThisValue%% is enclosed in triple quotation marks.

R Quotation marks in the run time value of %%ThisValue%% that match the enclosing quotation marks are escaped with the backslash character (\). For example, if the Script property is '%%ThisValue%%' and the run time value of the combo box is `Combo box's value`, then the generated script is `'Combo box\'s value'`.

None Quotation marks in the run time value of %%ThisValue%% that match the enclosing quotation marks are retained with no modification.

Enabling Rule. Specifies a rule that determines when the current control is enabled. Click the ellipsis (...) button to open the dialog box and specify the rule. The Enabling Rule property is visible only when other controls that can be used to specify an enabling rule exist on the canvas.

Specifying list items for combo boxes and list boxes: The List Item Properties dialog box allows you to specify the list items of a combo box or list box control.

Manually defined values. Allows you to explicitly specify each of the list items.

- **Identifier.** A unique identifier for the list item.

- **Name.** The name that appears in the list for this item. The name is a required field.
- **Default.** For a combo box, specifies whether the list item is the default item displayed in the combo box. For a list box, specifies whether the list item is selected by default.
- **Script.** Specifies the R script or Python for Spark script that is generated when the list item is selected.
- You can specify any valid R script or Python for Spark script. For multi-line scripts or long scripts, click the ellipsis (...) button and enter your script in the Script Property dialog.

Note: You can add a new list item in the blank line at the bottom of the existing list. Entering any of the properties other than the identifier will generate a unique identifier, which you can keep or modify. You can delete a list item by clicking on the *Identifier* cell for the item and pressing delete.

List Box

The List Box control allows you to display a list of items that support single or multiple selection and generate R script or Python for Spark script specific to the selected item(s). The List Box control has the following properties:

Identifier. The unique identifier for the control. This is the identifier to use when referencing the control in the script template.

Title. An optional title that appears above the control. For multi-line titles, use `\n` to specify line breaks.

ToolTip. Optional ToolTip text that appears when the user hovers over the control.

List Items. Click the ellipsis (...) button to open the List Item Properties dialog box, which allows you to specify the list items of the control. You can also open the List Item Properties dialog by double-clicking the List Box control on the canvas.

Mnemonic Key. An optional character in the title to use as a keyboard shortcut to the control. The character appears underlined in the title. The shortcut is activated by pressing Alt+[mnemonic key].

List Box Type. Specifies whether the list box supports single selection only or multiple selection. You can also specify that items are displayed as a list of check boxes.

Separator Type. Specifies the delimiter between the selected list items in the generated script. The allowed separators are a blank, a comma, and a plus sign (+). You can also enter an arbitrary single character to be used as the separator.

Minimum Selected. The minimum number of items that must be selected in the control, if any.

Maximum Selected. The maximum number of items that can be selected in the control, if any.

Script. Specifies the R script or Python for Spark script that is generated by this control at run time and can be inserted in the script template.
- The value `%%ThisValue%%` specifies the run time value of the control and is the default. If the list items are manually defined, the run time value is the value of the Script property for the selected list item. If the list items are based on a target list control, the run time value is the value of the selected list item. For multiple selection list box controls, the run time value is a list of the selected items, delimited by the specified Separator Type (default is blank separated). See the topic "Specifying list items for combo boxes and list boxes" on page 34 for more information.
- You can specify any valid R script or Python for Spark script. For multi-line scripts or long scripts, click the ellipsis (...) button and enter your script in the Script Property dialog.

Quote Handling. Specifies handling of quotation marks in the run time value of `%%ThisValue%%` when the Script property contains `%%ThisValue%%` as part of a quoted string. In this context, a quoted string is a string that is enclosed in single quotation marks or double quotation marks. Quote handling applies only to quotation marks that are the same type as the quotation marks that enclose `%%ThisValue%%`. The following types of quote handling are available.

Python
Quotation marks in the run time value of %%ThisValue%% that match the enclosing quotation marks are escaped with the backslash character (\). For example, if the Script property is '%%ThisValue%%' and the selected list item is `List item's value`, then the generated script is `'List item\'s value'`. Note that quote handling is not done when %%ThisValue%% is enclosed in triple quotation marks.

R Quotation marks in the run time value of %%ThisValue%% that match the enclosing quotation marks are escaped with the backslash character (\). For example, if the Script property is '%%ThisValue%%' and the selected list item is `List item's value`, then the generated script is `'List item\'s value'`.

None Quotation marks in the run time value of %%ThisValue%% that match the enclosing quotation marks are retained with no modification.

Enabling Rule. Specifies a rule that determines when the current control is enabled. Click the ellipsis (...) button to open the dialog box and specify the rule. The Enabling Rule property is visible only when other controls that can be used to specify an enabling rule exist on the canvas.

Text control

The Text control is a simple text box that can accept arbitrary input, and has the following properties:

Identifier. The unique identifier for the control. This is the identifier to use when referencing the control in the script template.

Title. An optional title that appears above the control. For multi-line titles, use `\n` to specify line breaks.

Title Position. Specifies the position of the title relative to the control. Values are Top and Left where Top is the default.

ToolTip. Optional ToolTip text that appears when the user hovers over the control.

Mnemonic Key. An optional character in the title to use as a keyboard shortcut to the control. The character appears underlined in the title. The shortcut is activated by pressing Alt+[mnemonic key].

Text Content. Specifies whether the contents are arbitrary or whether the text box must contain a string that adheres to rules for IBM SPSS Modeler field names.

Default Value. The default contents of the text box.

Width. Specifies the width of the text area of the control in characters. The allowed values are positive integers. An empty value means that the width is automatically determined.

Required for execution. Specifies whether a value is required in this control in order for execution to proceed. If **True** is specified, the user of the node dialog must specify a value for the control otherwise clicking the **OK** button will generate an error. If **False** is specified, the absence of a value in this control has no effect on the state of the **OK** button. The default is **False**.

Script. Specifies the R script or Python for Spark script that is generated and run by this control at run time and can be inserted in the script template.

- You can specify any valid R script or Python for Spark script. For multi-line scripts or long scripts, click the ellipsis (...) button and enter your script in the Script Property dialog.

- The value `%%ThisValue%%` specifies the run time value of the control, which is the content of the text box. This is the default.
- If the Script property includes `%%ThisValue%%` and the run time value of the text box is empty, then the text box control does not generate any script.

Quote Handling. Specifies handling of quotation marks in the run time value of `%%ThisValue%%` when the Script property contains `%%ThisValue%%` as part of a quoted string. In this context, a quoted string is a string that is enclosed in single quotation marks or double quotation marks. Quote handling applies only to quotation marks that are the same type as the quotation marks that enclose `%%ThisValue%%`. The following types of quote handling are available.

Python
Quotation marks in the run time value of `%%ThisValue%%` that match the enclosing quotation marks are escaped with the backslash character (\). For example, if the Script property is `'%%ThisValue%%'` and the run time value of the text control is `Text box's value`, then the generated script is `'Text box\'s value'`. Quote handling is not done when `%%ThisValue%%` is enclosed in triple quotation marks.

R Quotation marks in the run time value of `%%ThisValue%%` that match the enclosing quotation marks are escaped with the backslash character (\). For example, if the Script property is `'%%ThisValue%%'` and the run time value of the text control is `Text box's value`, then the generated script is `'Text box\'s value'`.

None Quotation marks in the run time value of `%%ThisValue%%` that match the enclosing quotation marks are retained with no modification.

Enabling Rule. Specifies a rule that determines when the current control is enabled. Click the ellipsis (...) button to open the dialog box and specify the rule. The Enabling Rule property is visible only when other controls that can be used to specify an enabling rule exist on the canvas.

Number Control

The Number control is a text box for entering a numeric value, and has the following properties:

Identifier. The unique identifier for the control. This is the identifier to use when referencing the control in the script template.

Title. An optional title that appears above the control. For multi-line titles, use `\n` to specify line breaks.

Title Position. Specifies the position of the title relative to the control. Values are Top and Left where Top is the default.

ToolTip. Optional ToolTip text that appears when the user hovers over the control.

Mnemonic Key. An optional character in the title to use as a keyboard shortcut to the control. The character appears underlined in the title. The shortcut is activated by pressing Alt+[mnemonic key].

Numeric Type. Specifies any limitations on what can be entered. A value of Real specifies that there are no restrictions on the entered value, other than it be numeric. A value of Integer specifies that the value must be an integer.

Spin Input. Specifies whether the control is displayed as a spinner. The default is False.

Increment. The increment when the control is displayed as a spinner.

Default Value. The default value, if any.

Minimum Value. The minimum allowed value, if any.

Maximum Value. The maximum allowed value, if any.

Width. Specifies the width of the text area of the control in characters. The allowed values are positive integers. An empty value means that the width is automatically determined.

Required for execution. Specifies whether a value is required in this control in order for execution to proceed. If **True** is specified, the user of the node dialog must specify a value for the control otherwise clicking the **OK** button will generate an error. If **False** is specified, the absence of a value in this control has no effect on the state of the **OK** button. The default is **False**.

Script. Specifies the R script or Python for Spark script that is generated and run by this control at run time and can be inserted in the script template.

- You can specify any valid R script or Python for Spark script. For multi-line scripts or long scripts, click the ellipsis (...) button and enter your script in the Script Property dialog.
- The value `%%ThisValue%%` specifies the run time value of the control, which is the numeric value. This is the default.
- If the Script property includes `%%ThisValue%%` and the run time value of the number control is empty, then the number control does not generate any script.

Enabling Rule. Specifies a rule that determines when the current control is enabled. Click the ellipsis (...) button to open the dialog box and specify the rule. The Enabling Rule property is visible only when other controls that can be used to specify an enabling rule exist on the canvas.

Date control

The Date control is a spinner control for specifying date/time values, which include dates, times, and datetimes. The Date control has the following properties:

Identifier. The unique identifier for the control. This is the identifier to use when referencing the control in the script template.

Title. An optional title that appears above the control. For multi-line titles, use `\n` to specify line breaks.

Title Position. Specifies the position of the title relative to the control. Values are Top and Left where Top is the default.

ToolTip. Optional ToolTip text that appears when the user hovers over the control.

Mnemonic Key. An optional character in the title to use as a keyboard shortcut to the control. The character appears underlined in the title. The shortcut is activated by pressing Alt+[mnemonic key].

Type. Specifies whether the control is for dates, times, or datetime values.

Date The control specifies a calendar date of the form yyyy-mm-dd. The default run time value is specified by the Default Value property.

Time The control specifies the time of day in the form hh:mm:ss. The default run time value is the current time of day.

Datetime
The control specifies a date and time of the form yyyy-mm-dd hh:mm:ss. The default run time value is the current date and time of day.

Default Value. The default run time value of the control when the type is Date. You can specify to display the current date or a particular date.

Script. Specifies the R script or Python for Spark script that is generated and run by this control at run time and can be inserted in the script template.

- You can specify any valid R script or Python for Spark script. For multi-line scripts or long scripts, click the ellipsis (...) button and enter your script in the Script Property dialog.
- The value `%%ThisValue%%` specifies the run time value of the control. This is the default.

Enabling Rule. Specifies a rule that determines when the current control is enabled. Click the ellipsis (...) button to open the dialog box and specify the rule. The Enabling Rule property is visible only when other controls that can be used to specify an enabling rule exist on the canvas.

Note: The Date control is not supported in releases of IBM SPSS Modeler before release 18.

Secured Text

The Secured Text control is a text box that masks user entry with asterisks.

Identifier. The unique identifier for the control. This is the identifier to use when referencing the control in the script template.

Title. An optional title that appears above the control. For multi-line titles, use `\n` to specify line breaks.

Title Position. Specifies the position of the title relative to the control. Values are Top and Left where Top is the default.

ToolTip. Optional ToolTip text that appears when the user hovers over the control.

Mnemonic Key. An optional character in the title to use as a keyboard shortcut to the control. The character appears underlined in the title. The shortcut is activated by pressing Alt+[mnemonic key].

Width. Specifies the width of the text area of the control in characters. The allowed values are positive integers. An empty value means that the width is automatically determined.

Required for execution. Specifies whether a value is required in this control in order for execution to proceed. If **True** is specified, the user of the node dialog must specify a value for the control otherwise clicking the **OK** button will generate an error. If **False** is specified, the absence of a value in this control has no effect on the state of the **OK** button. The default is **False**.

Script. Specifies the R script or Python for Spark script that is generated and run by this control at run time and can be inserted in the script template.

- You can specify any valid R script or Python for Spark script. For multi-line scripts or long scripts, click the ellipsis (...) button and enter your script in the Script Property dialog.
- The value `%%ThisValue%%` specifies the run time value of the control, which is the content of the text box. This is the default.
- If the Script property includes `%%ThisValue%%` and the run time value of the secured text control is empty, then the secured text control does not generate any R script or Python for Spark script.

Quote Handling. Specifies handling of quotation marks in the run time value of `%%ThisValue%%` when the Script property contains `%%ThisValue%%` as part of a quoted string. In this context, a quoted string is a string that is enclosed in single quotation marks or double quotation marks. Quote handling applies only to quotation marks that are the same type as the quotation marks that enclose `%%ThisValue%%` and only when `Encrypt passed value=False`. The following types of quote handling are available.

Python
Quotation marks in the run time value of `%%ThisValue%%` that match the enclosing quotation marks are escaped with the backslash character (\). For example, if the Script property is `'%%ThisValue%%'` and the run time value of the control is `Secured Text's value`, then the generated script is `'Secured Text\'s value'`. Quote handling is not done when `%%ThisValue%%` is enclosed in triple quotation marks.

R Quotation marks in the run time value of `%%ThisValue%%` that match the enclosing quotation marks are escaped with the backslash character (\). For example, if the Script property is `'%%ThisValue%%'` and the run time value of the control is `Secured Text's value`, then the generated script is `'Secured Text\'s value'`.

None Quotation marks in the run time value of `%%ThisValue%%` that match the enclosing quotation marks are retained with no modification.

Enabling Rule. Specifies a rule that determines when the current control is enabled. Click the ellipsis (...) button to open the dialog box and specify the rule. The Enabling Rule property is visible only when other controls that can be used to specify an enabling rule exist on the canvas.

Note: The Secured Text control is not supported in releases of IBM SPSS Modeler before release 18.

Static Text Control

The Static Text control allows you to add a block of text to your node dialog, and has the following properties:

Identifier. The unique identifier for the control.

Title. The content of the text block. For multi-line content, use `\n` to specify line breaks.

Enabling Rule. Specifies a rule that determines when the current control is enabled. Click the ellipsis (...) button to open the dialog box and specify the rule. The Enabling Rule property is visible only when other controls that can be used to specify an enabling rule exist on the canvas.

Color Picker

The Color Picker control is a user interface for specifying a color and generating the associated RGB value. The Color Picker control has the following properties:

Identifier. The unique identifier for the control. This is the identifier to use when referencing the control in the script template.

Title. An optional title that appears above the control. For multi-line titles, use `\n` to specify line breaks.

Title Position. Specifies the position of the title relative to the control. Values are Top and Left where Top is the default.

ToolTip. Optional ToolTip text that appears when the user hovers over the control.

Mnemonic Key. An optional character in the title to use as a keyboard shortcut to the control. The character appears underlined in the title. The shortcut is activated by pressing Alt+[mnemonic key].

Script. Specifies the R script or Python for Spark script that is generated and run by this control at run time and can be inserted in the script template.

- You can specify any valid R script or Python for Spark script. For multi-line scripts or long scripts, click the ellipsis (...) button and enter your script in the Script Property dialog.
- The value `%%ThisValue%%` specifies the run time value of the control, which is the RGB value of the selected color. The RGB value is represented as a blank separated list of integers in the following order: R value, G value, B value.

Enabling Rule. Specifies a rule that determines when the current control is enabled. Click the ellipsis (...) button to open the dialog box and specify the rule. The Enabling Rule property is visible only when other controls that can be used to specify an enabling rule exist on the canvas.

Note: The Color Picker control is not supported in releases of IBM SPSS Modeler before release 18.

Table Control

The Table control creates a table with a fixed number of columns and a variable number of rows that are added at run time. The Table control has the following properties:

Identifier. The unique identifier for the control. This is the identifier to use when referencing the control in the script template.

Title. An optional title that appears above the control. For multi-line titles, use `\n` to specify line breaks.

ToolTip. Optional ToolTip text that appears when the user hovers over the control.

Mnemonic Key. An optional character in the title to use as a keyboard shortcut to the control. The character appears underlined in the title. The shortcut is activated by pressing Alt+[mnemonic key].

Reorder Buttons. Specifies whether move up and move down buttons are added to the table. These buttons are used at run time to reorder the rows of the table.

Table Columns. Click the ellipsis (...) button to open the Table Columns dialog box, where you specify the columns of the table.

Minimum Rows. The minimum number of rows that must be in the table.

Maximum Rows. The maximum number of rows that can be in the table.

Required for Execution. Specifies whether a value is required in this control in order for execution to proceed. If **True** is specified, the user of the node dialog must specify a value for the control otherwise clicking the **OK** button will generate an error. If **False** is specified, the absence of a value in this control has no effect on the state of the **OK** button.

Script. Specifies the R script or Python for Spark script that is generated by this control at run time and can be inserted in the script template.

- The value `%%ThisValue%%` specifies the run time value of the control and is the default. The run time value is a blank-separated list of the script that is generated by each column in the table, starting with the leftmost column. If the Script property includes `%%ThisValue%%` and none of the columns generate script, then the table as a whole does not generate any script.
- You can specify any valid R script or Python for Spark script. For multi-line scripts or long scripts, click the ellipsis (...) button and enter your script in the Script Property dialog.

Enabling Rule. Specifies a rule that determines when the current control is enabled. Click the ellipsis (...) button to open the dialog box and specify the rule. The Enabling Rule property is visible only when other controls that can be used to specify an enabling rule exist on the canvas.

Note: The Table control is not supported in releases of IBM SPSS Modeler before release 18.

Specifying columns for table controls: The Table Columns dialog box specifies the properties of the columns of the Table control.

Identifier. A unique identifier for the column.

Column Name. The name of the column as it appears in the table.

Contents. Specifies the type of data for the column. The value **Real** specifies that there are no restrictions on the entered value, other than it is numeric. The value **Integer** specifies that the value must be an integer. The value **Any** specifies that there are no restrictions on the entered value. The value **Variable Name** specifies that the value must meet the requirements for a valid variable name in IBM SPSS Statistics.

Default Value. The default value for this column, if any, when new rows are added to the table at run time.

Separator Type. Specifies the delimiter between the values of the column, in the generated script. The allowed separators are a blank, a comma, and a plus sign (+). You can also enter an arbitrary single character to be used as the separator.

Quoted. Specifies whether each value in the column is enclosed in double quotation marks in the generated script.

Quote Handling. Specifies handling of quotation marks in cell entries for the column when the Quoted property is true. Quote handling applies only to double quotation marks in cell values. The following types of quote handling are available.

Python
: Double quotation marks in cell values are escaped with the backslash character (\). For example, if the cell value is `This "quoted" value` then the generated script is `"This \"quoted\" value"`.

R
: Double quotation marks in cell values are escaped with the backslash character (\). For example, if the cell value is `This "quoted" value` then the generated script is `"This \"quoted\" value"`.

None
: Double quotation marks in cell values are retained with no modification.

Width(chars). Specifies the width of the column in characters. The allowed values are positive integers.

Script. Specifies the R script or Python for Spark script that is generated by this column at run time. The generated script for the table as a whole is a blank-separated list of the script that is generated by each column in the table, starting with the leftmost column.
- You can specify any valid R script or Python for Spark script. For multi-line scripts or long scripts, click the ellipsis (...) button and enter your script in the Script Property dialog.
- The value `%%ThisValue%%` specifies the run time value of the column, which is a list of the values in the column, delimited by the specified separator.
- If the Script property for the column includes `%%ThisValue%%` and the run time value of the column is empty, then the column does not generate any script.

Note: You can add a row for a new Table column in the blank line at the bottom of the existing list in the Table Columns dialog. Entering any of the properties other than the identifier generates a unique identifier, which you can keep or modify. You can delete a Table column by clicking the identifier cell for the Table column and pressing delete.

Link to Control

You can link a Table control to a Field Chooser control. When a Table control is linked to a Field Chooser, there is a row in the table for each field in the Field Chooser. Rows are added to the table by adding fields to the Field Chooser. Rows are deleted from the table by removing fields from the Field Chooser. A linked Table control can be used, for example, to specify properties of fields that are selected in a Field Chooser.

To enable linking, the table must have a column with Variable Name for the Contents property and there must be at least one Field Chooser control on the canvas.

To link a Table control to a Field Chooser, specify the Field Chooser from the list of Available controls in the Link to Control group on the Table Columns dialog box. Then, select the table column, referred to as the **Linked column**, that defines the link. When the table is rendered, the linked column displays the current fields in the Field Chooser. You can link only to multi-field Field Choosers.

Item Group

The Item Group control is a container for other controls, allowing you to group and control the script generated from multiple controls. For example, you have a set of check boxes that specify optional settings for a subcommand, but only want to generate the script for the subcommand if at least one box is checked. This is accomplished by using an Item Group control as a container for the check box controls. The following types of controls can be contained in an Item Group: field chooser, check box, combo box, list box, text control, number control, static text, radio group, and file browser. The Item Group control has the following properties:

Identifier. The unique identifier for the control. This is the identifier to use when referencing the control in the script template.

Title. An optional title for the group. For multi-line titles, use `\n` to specify line breaks.

Required for execution. Selecting **True** means that if the user of the node dialog does not specify a value for at least one control in the group, clicking the **OK** button generates an error.

For example, the group consists of a set of check boxes. If Required for execution is set to **True** and all of the boxes are unchecked, then clicking the **OK** button generates an error.

Script. Specifies the R script or Python for Spark script that is generated and run by this control at run time and can be inserted in the script template.
- You can specify any valid R script or Python for Spark script. For multi-line scripts or long scripts, click the ellipsis (...) button and enter your script in the Script Property dialog.
- You can include identifiers for any controls contained in the item group. At run time the identifiers are replaced with the R script or Python script generated by the controls.
- The value `%%ThisValue%%` generates a blank-separated list of the R script or Python script generated by each control in the item group, in the order in which they appear in the group (top to bottom). This is the default. If the Script property includes `%%ThisValue%%` and no script is generated by any of the controls in the item group, then the item group as a whole does not generate any script.

Enabling Rule. Specifies a rule that determines when the current control is enabled. Click the ellipsis (...) button to open the dialog box and specify the rule. The Enabling Rule property is visible only when other controls that can be used to specify an enabling rule exist on the canvas.

Radio Group

The Radio Group control is a container for a set of radio buttons, each of which can contain a set of nested controls. The Radio Group control has the following properties:

Identifier. The unique identifier for the control. This is the identifier to use when referencing the control in the script template.

Title. An optional title for the group. For multi-line titles, use `\n` to specify line breaks.

ToolTip. Optional ToolTip text that appears when the user hovers over the control.

Radio Buttons. Click the ellipsis (...) button to open the Radio Group Properties dialog box, which allows you to specify the properties of the radio buttons as well as to add or remove buttons from the group. The ability to nest controls under a given radio button is a property of the radio button and is set in the Radio Group Properties dialog box. Note that you can also open the Radio Group Properties dialog by double-clicking the Radio Group control on the canvas.

Script. Specifies the R script or Python for Spark script that is generated by this control at run time and can be inserted in the script template.
- You can specify any valid R script or Python for Spark script. For multi-line scripts or long scripts, click the ellipsis (...) button and enter your script in the Script Property dialog.
- The value `%%ThisValue%%` specifies the run time value of the radio button group, which is the value of the Script property for the selected radio button. This is the default. If the Script property includes `%%ThisValue%%` and no script is generated by the selected radio button, then the radio button group does not generate any script.

Enabling Rule. Specifies a rule that determines when the current control is enabled. Click the ellipsis (...) button to open the dialog box and specify the rule. The Enabling Rule property is visible only when other controls that can be used to specify an enabling rule exist on the canvas.

Defining radio buttons: The Radio Button Group Properties dialog box allows you to specify a group of radio buttons.

Identifier. A unique identifier for the radio button.

Column Name. The name that appears next to the radio button. The name is a required field.

ToolTip. Optional ToolTip text that appears when the user hovers over the control.

Mnemonic Key. An optional character in the name to use as a mnemonic. The specified character must exist in the name.

Nested Group. Specifies whether other controls can be nested under this radio button. The default is false. When the nested group property is set to true, a rectangular drop zone is displayed, nested and indented, under the associated radio button. The following controls can be nested under a radio button: field chooser, check box, text control, static text, number control, combo box, list box, and file browser.

Default. Specifies whether the radio button is the default selection.

Enabling Rule. Specifies a rule that determines when the current control is enabled. Click the ellipsis (...) button to open the dialog box and specify the rule. The Enabling Rule property is visible only when other controls that can be used to specify an enabling rule exist on the canvas.

Script. Specifies the R script or Python for Spark script that is generated when the radio button is selected.
- You can specify any valid R script or Python for Spark script. For multi-line scripts or long scripts, click the ellipsis (...) button and enter your script in the Script Property dialog.
- For radio buttons containing nested controls, the value `%%ThisValue%%` generates a blank-separated list of the R script or Python for Spark script generated by each nested control, in the order in which they appear under the radio button (top to bottom).

You can add a new radio button in the blank line at the bottom of the existing list. Entering any of the properties other than the identifier will generate a unique identifier, which you can keep or modify. You can delete a radio button by clicking on the *Identifier* cell for the button and pressing delete.

Check Box Group

The Check Box Group control is a container for a set of controls that are enabled or disabled as a group, by a single check box. The following types of controls can be contained in a Check Box Group: field chooser, check box, combo box, list box, text control, number control, static text, radio group, and file browser. The Check Box Group control has the following properties:

Identifier. The unique identifier for the control. This is the identifier to use when referencing the control in the script template.

Title. An optional title for the group. For multi-line titles, use `\n` to specify line breaks.

Checkbox Title. An optional label that is displayed with the controlling check box. Supports `\n` to specify line breaks.

ToolTip. Optional ToolTip text that appears when the user hovers over the control.

Mnemonic Key. An optional character in the title to use as a keyboard shortcut to the control. The character appears underlined in the title. The shortcut is activated by pressing Alt+[mnemonic key].

Default Value. The default state of the controlling check box--checked or unchecked.

Checked/Unchecked R Script. Specifies the R script that is generated when the control is checked and when it is unchecked. To include the R script in the script template, use the value of the Identifier property. The generated R script, whether from the Checked R Script or Unchecked R Script property, will be inserted at the specified position(s) of the identifier. For example, if the identifier is *checkboxgroup1*, then at run time, instances of `%%checkboxgroup1%%` in the script template will be replaced by the value of the Checked R Script property when the box is checked and the Unchecked R Script property when the box is unchecked.

- You can specify any valid R script or Python for Spark script. For multi-line scripts or long scripts, click the ellipsis (...) button and enter your script in the Script Property dialog.
- You can include identifiers for any controls contained in the check box group. At run time the identifiers are replaced with the R script generated by the controls.
- The value `%%ThisValue%%` can be used in either the Checked R Script or Unchecked R Script property. It generates a blank-separated list of the R script generated by each control in the check box group, in the order in which they appear in the group (top to bottom).
- By default, the Checked R Script property has a value of `%%ThisValue%%` and the Unchecked R Script property is blank.

Enabling Rule. Specifies a rule that determines when the current control is enabled. Click the ellipsis (...) button to open the dialog box and specify the rule. The Enabling Rule property is visible only when other controls that can be used to specify an enabling rule exist on the canvas.

File Browser

The File Browser control consists of a text box for a file path and a browse button that opens a standard IBM SPSS Modeler dialog to open or save a file. The File Browser control has the following properties:

Identifier. The unique identifier for the control. This is the identifier to use when referencing the control in the script template.

Title. An optional title that appears above the control. For multi-line titles, use `\n` to specify line breaks.

Title Position. Specifies the position of the title relative to the control. Values are Top and Left where Top is the default.

ToolTip. Optional ToolTip text that appears when the user hovers over the control.

Mnemonic Key. An optional character in the title to use as a keyboard shortcut to the control. The character appears underlined in the title. The shortcut is activated by pressing Alt+[mnemonic key].

File System Operation. Specifies whether the dialog launched by the browse button is appropriate for opening files or for saving files. A value of Open indicates that the browse dialog validates the existence of the specified file. A value of Save indicates that the browse dialog does not validate the existence of the specified file.

Browser Type. Specifies whether the browse dialog is used to select a file (Locate File) or to select a folder (Locate Folder).

File Filter. Click the ellipsis (...) button to open the File Filter dialog box, which allows you to specify the available file types for the open or save dialog. By default, all file types are allowed. Note that you can also open the File Filter dialog by double-clicking the File Browser control on the canvas.

File System Type. In distributed analysis mode, this specifies whether the open or save dialog browses the file system on which IBM SPSS Modeler Server is running or the file system of your local computer. Select **Server** to browse the file system of the server or **Client** to browse the file system of your local computer. The property has no effect in local analysis mode.

Required for execution. Specifies whether a value is required in this control in order for execution to proceed. If **True** is specified, the user of the node dialog must specify a value for the control otherwise clicking the **OK** button will generate an error. If **False** is specified, the absence of a value in this control has no effect on the state of the **OK** button. The default is **False**.

Default. The default value of the control.

Script. Specifies the R script or Python for Spark script that is generated by this control at run time and can be inserted in the script template.
- You can specify any valid R script or Python for Spark script. For multi-line scripts or long scripts, click the ellipsis (...) button and enter your script in the Script Property dialog.
- The value `%%ThisValue%%` specifies the run time value of the text box, which is the file path enclosed by double quotation marks, specified manually or populated by the browse dialog. This is the default.
- If the Script property includes `%%ThisValue%%` and the run time value of the text box is empty, then the file browser control does not generate any script.

Enabling Rule. Specifies a rule that determines when the current control is enabled. Click the ellipsis (...) button to open the dialog box and specify the rule. The Enabling Rule property is visible only when other controls that can be used to specify an enabling rule exist on the canvas.

File type filter: The File Filter dialog box allows you to specify the file types displayed in the Files of type and Save as type drop-down lists for open and save dialogs accessed from a File System Browser control. By default, all file types are allowed.

To specify file types not explicitly listed in the dialog box:
1. Select Other.
2. Enter a name for the file type.
3. Enter a file type using the form `*.suffix`--for example, `*.xls`. You can specify multiple file types, each separated by a semicolon.

Tab

The Tab control adds a tab to the node dialog. Any of the other controls can be added to the new tab. The Tab control has the following properties:

Identifier. The unique identifier for the control.

Title. The title of the tab.

Position. Specifies the position of the tab on the node dialog, relative to the other tabs on the node dialog.

Script. Specifies the R script or Python for Spark script that is generated and run by this control at run time and can be inserted in the script template.

- You can specify any valid R script or Python for Spark script and you can use `\n` for line breaks.
- The value `%%ThisValue%%` generates a blank-separated list of the R script or Python for Spark script generated by each control in the tab, in the order in which they appear on the tab (top to bottom and left to right). This is the default.
- If the Script property includes `%%ThisValue%%` and no R script or Python for Spark script is generated by any of the controls in the tab, then the tab as a whole does not generate any script.

Enabling Rule. Specifies a rule that determines when the current control is enabled. Click the ellipsis (...) button to open the dialog box and specify the rule. The Enabling Rule property is visible only when other controls that can be used to specify an enabling rule exist on the canvas.

Sub-dialog button

The Sub-dialog Button control specifies a button for launching a sub-dialog and provides access to the Dialog Builder for the sub-dialog. The Sub-dialog Button has the following properties:

Identifier. The unique identifier for the control.

Title. The text that is displayed in the button.

ToolTip. Optional ToolTip text that appears when the user hovers over the control.

Sub-dialog. Click the ellipsis (...) button to open the Custom Dialog Builder for the sub-dialog. You can also open the builder by double-clicking on the Sub-dialog button.

Mnemonic Key. An optional character in the title to use as a keyboard shortcut to the control. The character appears underlined in the title. The shortcut is activated by pressing Alt+[mnemonic key].

Enabling Rule. Specifies a rule that determines when the current control is enabled. Click the ellipsis (...) button to open the dialog box and specify the rule. The Enabling Rule property is visible only when other controls that can be used to specify an enabling rule exist on the canvas.

Note: The Sub-dialog Button control cannot be added to a sub-dialog.

Dialog properties for a sub-dialog: To view and set properties for a sub-dialog:

1. Open the sub-dialog by double-clicking on the button for the sub-dialog in the main dialog, or single-click the sub-dialog button and click the ellipsis (...) button for the Sub-dialog property.
2. In the sub-dialog, click on the canvas in an area outside of any controls. With no controls on the canvas, the properties for a sub-dialog are always visible.

Sub-dialog Name. The unique identifier for the sub-dialog. The Sub-dialog Name property is required.

Note: If you specify the Sub-dialog Name as an identifier in the Script Template--as in `%%My Sub-dialog Name%%`--it will be replaced at run-time with a blank-separated list of the script generated by each control in the sub-dialog, in the order in which they appear (top to bottom and left to right).

Title. Specifies the text to be displayed in the title bar of the sub-dialog box. The Title property is optional but recommended.

Help File. Specifies the path to an optional help file for the sub-dialog. This is the file that will be launched when the user clicks the **Help** button on the sub-dialog, and may be the same help file specified for the main dialog. Help files must be in HTML format. See the description of the Help File property for Dialog Properties for more information.

Specifying an Enabling Rule for a Control

You can specify a rule that determines when a control is enabled. For example, you can specify that a radio group is enabled when a field chooser is populated. The available options for specifying the enabling rule depend on the type of control that defines the rule.

Field Chooser
You can specify that the current control is enabled when a Field Chooser is populated with at least one field (Non-empty). You can alternately specify that the current control is enabled when a Field Chooser is not populated (Empty).

Check Box or Check Box Group
You can specify that the current control is enabled when a Check Box or Check Box Group is checked. You can alternately specify that the current control is enabled when a Check Box or Check Box Group is not checked.

Combo Box or Single-Select List Box
You can specify that the current control is enabled when a particular value is selected in a Combo Box or Single-Select List Box. You can alternately specify that the current control is enabled when a particular value is not selected in a Combo Box or Single-Select List Box.

Multi-Select List Box
You can specify that the current control is enabled when a particular value is among the selected values in a Multi-Select List Box. You can alternately specify that the current control is enabled when a particular value is not among the selected values in a Multi-Select List Box.

Radio Group
You can specify that the current control is enabled when a particular radio button is selected. You can alternately specify that the current control is enabled when a particular radio button is not selected.

Controls for which an enabling rule can be specified have an associated Enabling Rule property.

Note:

- Enabling rules apply whether the control that defines the rule is enabled or not. For example, consider a rule that specifies that a radio group is enabled when a field chooser is populated. The radio group is then enabled whenever the field chooser is populated, regardless of whether the field chooser is enabled.
- When a Tab control is disabled, all controls on the tab are disabled regardless of whether any of those controls have an enabling rule that is satisfied.
- When a Check Box Group is disabled, all controls in the group are disabled regardless of whether the controlling check box is checked.

Extension Properties

The Extension Properties dialog specifies information about the current extension within the Custom Dialog Builder for Extensions, such as the name of the extension and the files in the extension.

- All custom node dialogs that are created in the Custom Dialog Builder for Extensions are part of an extension.
- Fields on the Required tab of the Extension Properties dialog must be specified before you can install an extension and the custom node dialogs that are contained in it.

To specify the properties for an extension, from the menus in the Custom Dialog Builder for Extensions choose:

Extension > **Properties**

Required properties of extensions

Name A unique name to associate with the extension. It can consist of up to three words and is not case sensitive. Characters are restricted to seven-bit ASCII. To minimize the possibility of name conflicts, you may want to use a multi-word name, where the first word is an identifier for your organization, such as a URL.

Summary
A short description of the extension that is intended to be displayed as a single line.

Version
A version identifier of the form x.x.x, where each component of the identifier must be an integer, as in 1.0.0. Zeros are implied if not provided. For example, a version identifier of 3.1 implies 3.1.0. The version identifier is independent of the IBM SPSS Modeler version.

Minimum SPSS Modeler Version
The minimum version of SPSS Modeler required to run the extension.

Files The Files list displays the files that are currently included in the extension. Click **Add** to add files to the extension. You can also remove files from the extension and you can extract files to a specified folder.

- Custom node dialogs have a filetype of `.cfe`.
- Translation files for components of the extension are added from the Localization settings on the Optional tab.
- You can add a readme file to the extension. Specify the filename as `ReadMe.txt`. Users can access the readme file from the dialog that displays the details for the extension. You can include localized versions of the readme file, specified as `ReadMe_<language identifier>.txt`, as in `ReadMe_fr.txt` for a French version.

Optional properties of extensions

General properties

Description
A more detailed description of the extension than that provided for the **Summary** field. For example, you might list the major features available with the extension.

Date An optional date for the current version of the extension. No formatting is provided.

Author
The author of the extension. You might want to include an email address.

Links A set of URLs to associate with the extension; for example, the author's home page. The format of this field is arbitrary so be sure to delimit multiple URLs with spaces, commas, or some other reasonable delimiter.

Keywords
A set of keywords with which to associate the extension.

Platform
Information about any restrictions that apply to using the extension on particular operating system platforms.

Dependencies

Maximum SPSS Modeler Version
The maximum version of IBM SPSS Modeler on which the extension can be run.

Integration Plug-in for R required
Specifies whether the Integration Plug-in for R is required.

If the extension requires any R packages from the CRAN package repository, then enter the names of those packages in the Required R Packages control. Names are case sensitive. To add the first package, click anywhere in the Required R Packages control to highlight the entry field. Pressing **Enter**, with the cursor in a given row, will create a new row. You delete a row by selecting it and pressing **Delete**.

Localization

Custom Nodes

You can add translated versions of the properties file (specifies all strings that appear in the node dialog) for a custom node dialog within the extension. To add translations for a particular nodedialog, select the dialog, click **Add Translations**, and select the folder that contains the translated versions. All translated files for a particular node dialog must be in the same folder. For instructions on creating the translation files, see the topic "Creating Localized Versions of Custom Node Dialogs" on page 53.

Translation Catalogues Folder

You can provide localized versions of the **Summary** and **Description** fields for the extension that are displayed when end users view the extension details from the Extension Hub. The set of all localized files for an extension must be in a folder named `lang`. Browse to the `lang` folder that contains the localized files and select that folder.

To provide localized versions of the **Summary** and **Description** fields, create a file named `<extension name>_<language-identifier>.properties` for each language for which a translation is being provided. At run time, if the `.properties` file for the current user interface language cannot be found, the values of the **Summary** and **Description** fields that are specified on the Required and Optional tabs are used.

- `<extension name>` is the value of the **Name** field for the extension, with any spaces replaced by underscore characters.
- `<language-identifier>` is the identifier for a particular language. Identifiers for the languages that are supported by IBM SPSS Modeler are shown in what follows.

For example, the French translations for an extension named `MYORG MYSTAT` are stored in the file `MYORG_MYSTAT_fr.properties`.

The `.properties` file must contain the following two lines, which specify the localized text for the two fields:

```
Summary=<localized text for Summary field>
Description=<localized text for Description field>
```

- The keywords Summary and Description must be in English and the localized text must be on the same line as the keyword, and with no line breaks.
- The file must be in ISO 8859-1 encoding. Characters that cannot be directly represented in this encoding must be written with Unicode escapes ("`\u`").

The `lang` folder that contains the localized files must have a subfolder named `<language-identifier>` that contains the localized `.properties` file for a particular language. For example, the French `.properties` file must be in the `lang/fr` folder.

Language Identifiers

de. German

en. English

es. Spanish

fr. French

it. Italian

ja. Japanese

ko. Korean

pl. Polish

pt_BR. Brazilian Portuguese

ru. Russian

zh_CN. Simplified Chinese

zh_TW. Traditional Chinese

Managing custom node dialogs

The Custom Dialog Builder for Extensions allows you to manage custom node dialogs, within extensions that are created by you or by other users. Custom node dialogs must be installed to all instances of SPSS Modeler Client or SPSS Modeler Batch where they are neededbefore they can be used. Note that there is nothing that needs to be installed to SPSS Modeler Server to use a custom dialog node in server mode.

Note: You can only modify custom node dialogs that are created in IBM SPSS Modeler.

Opening an extension that contains custom node dialogs

You can open an extension bundle file (`.mpe`) that contains the specifications for one or more custom node dialogs or you can open an installed extension. You can modify any of the nodedialogs in the extension and save or install the extension. Installing the extension installs the node dialogs that are contained in the extension. Saving the extension saves changes that were made to any of the node dialogs in the extension.

To open an extension bundle file, from the menus in the Custom Dialog Builder for Extensions choose:

File > **Open**

To open an installed extension, from the menus in the Custom Dialog Builder for Extensions choose:

File > **Open Installed**

Note: If you are opening an installed extension to modify it, choosing **File** > **Install** reinstalls it, replacing the existing version. Using **Edit** on the context menu of a node created using the Custom Dialog Builder will not open the node dialog in the Custom Dialog Builder.

Saving to an extension bundle file

Saving the extension that is open in the Custom Dialog Builder for Extensions also saves the custom node dialogs that are contained in the extension. Extensions are saved to an extension bundle file (`.mpe`).

From the menus in the Custom Dialog Builder for Extensions, choose:

File > **Save**

Installing an extension

Installing the extension that is open in the Custom Dialog Builder for Extensions also installs the custom node dialogs that are contained in the extension. Installing an existing extension replaces the existing version, which includes replacing all custom node dialogs in the extension that were already installed.

To install the currently open extension, from the menus in the Custom Dialog Builder for Extensions choose:

File > **Install**

By default, extensions are installed to a general user-writable location for your operating system. For more information, see the topic "Installation locations for extensions" on page 26.

Note: In an open stream, the existing versions of the node dialogs that are contained in the extension will not be replaced. When you open a stream that contains a Custom Dialog Builder node that has been re-installed, you will receive a warning message.

Uninstalling an extension

From the menus in the Custom Dialog Builder for Extensions, choose:

File > **Uninstall**

Uninstalling an extension uninstalls all custom node dialogs that are contained in the extension. You can also uninstall extensions from the Extension Hub.

Importing a custom dialog package file

You can import a custom dialog package (`.cfd`) file into the Custom Dialog Builder for Extensions. The `.cfd` file is converted into a `.cfe` file, which is added to a new extension.

From the menus in the Custom Dialog Builder for Extensions, choose:

File > **Import**

You can also add `.cfe` files to an extension from the Extension Properties dialog, which is accessed from **Extension** > **Properties** within the Custom Dialog Builder for Extensions.

Adding a custom node dialog to an extension

You can add a new custom nodedialog to an extension.

From the menus in the Custom Dialog Builder for Extensions, choose:

Extension > **New Dialog**

Switching between multiple custom node dialogs in an extension

If the current extension contains multiple custom node dialogs, you can switch between them.

From the menus in the Custom Dialog Builder for Extensions, choose:

Extension > **Edit Dialog** and select the custom node dialog that you want to work with.

Creating a new extension

When you create a new extension in the Custom Dialog Builder for Extensions, a new empty custom node dialog is added to the extension.

To create a new extension, from the menus in the Custom Dialog Builder for Extensions choose:

File > **New**

Extensions in SPSS Modeler Batch or in IBM SPSS Collaboration and Deployment Services

To use extensions in an SPSS Modeler Batch or IBM SPSS Collaboration and Deployment Services installation, ensure that the environment variable *IBM_SPSS_MODELER_EXTENSION_PATH* is defined on the target environment, and that it points to the location that contains the extensions. If the streams that contain a custom node were stored to the IBM SPSS Collaboration and Deployment Services Repository before the *IBM_SPSS_MODELER_EXTENSION_PATH* environment variable was defined, you must re-store the streams to the repository before they will run successfully.

Note: Ensure that the version of the SPSS Modeler Batch or IBM SPSS Collaboration and Deployment Services adapter for SPSS Modeler matches the version of SPSS Modeler Client where the extension was created.

Creating Localized Versions of Custom Node Dialogs

You can create localized versions of custom node dialogs for any of the languages supported by IBM SPSS Modeler. You can localize any string that appears in a custom node dialog and you can localize the optional help file.

To localize dialog strings

You must create a copy of the properties file associated with the custom node dialog for each language that you plan to deploy. The properties file contains all of the localizable strings associated with the node dialog.

Extract the custom node dialog file (`.cfe`) from the extension by selecting the file in the Extension Properties dialog (within the Custom Dialog Builder for Extensions) and clicking **Extract**. Then, extract the contents of the `.cfe` file. A `.cfe` file is simply a `.zip` file. The extracted contents of a `.cfe` file includes a properties file for each supported language, where the name of the file for a particular language is given by `<Dialog Name>_<language identifier>.properties` (see language identifiers in the table that follows).

1. Open each properties file, that you plan to translate, with a text editor that supports UTF-8, such as Notepad on Windows. Modify the values associated with any properties that need to be localized, but do not modify the names of the properties. Properties associated with a specific control are prefixed with the identifier for the control. For example, the ToolTip property for a control with the identifier *options_button* is *options_button_tooltip_LABEL*. Title properties are simply named *<identifier>_LABEL*, as in *options_button_LABEL*.
2. Add the localized versions of the properties files back to the custom node dialog file (`.cfe`) from the Localization settings on the Optional tab of the Extension Properties dialog. For more information, see the topic “Optional properties of extensions” on page 49.

When the node dialog is launched, IBM SPSS Modeler searches for a properties file whose language identifier matches the current language, as specified by the Language drop-down on the General tab in the Options dialog box. If no such properties file is found, the default file `<Dialog Name>.properties` is used.

To localize the help file

1. Make a copy of the help file associated with the custom nodedialog and localize the text for the language you want.
2. Rename the copy to `<Help File>_<language identifier>.htm`, using the language identifiers in the table below. For example, if the help file is `myhelp.htm` and you want to create a German version of the file, then the localized help file should be named `myhelp_de.htm`.

Store all localized versions of the help file in the same directory as the non-localized version. When you add the non-localized help file from the Help File property of Dialog Properties, the localized versions are automatically added to the node dialog.

If there are supplementary files such as image files that also need to be localized, then you must manually modify the appropriate paths in the main help file to point to the localized versions. Supplementary files, including localized versions, must be manually added to the custom node dialog (`.cfe`) file. See the previous section titled "To localize dialog strings" for information about accessing and manually modifying custom nodedialog files.

When the node dialog is launched, IBM SPSS Modeler searches for a help file whose language identifier matches the current language, as specified by the Language drop-down on the General tab in the Options dialog box. If no such help file is found, the help file specified for the node dialog (the file specified in the Help File property of Dialog Properties) is used.

Language Identifiers

de. German

en. English

es. Spanish

fr. French

it. Italian

ja. Japanese

ko. Korean

pl. Polish

pt_BR. Brazilian Portuguese

ru. Russian

zh_CN. Simplified Chinese

zh_TW. Traditional Chinese

Note: Text in custom node dialogs and associated help files is not limited to the languages supported by IBM SPSS Modeler. You are free to write the node dialog and help text in any language without creating language-specific properties and help files. All users of your node dialog will then see the text in that language.

Notices

Trademarks

Terms and conditions for product documentation

Index

Printed in USA