# *IBM SPSS Modeler Extensions*

**Note**

Before using this information and the product it supports, read the information in "Notices" on page 41.

**Product Information**

This edition applies to version 17, release 1, modification 0 of IBM(r) SPSS(r) 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 8.

**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.

From 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 8 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 8.

**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.

From 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 and will 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
- Python scripts must use the Spark API because data will be presented in the form of a Spark DataFrame.

## Scripting with Python for Spark

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

### Prerequisites

To execute a Python/Spark script in IBM SPSS Modeler, you must have a connection to IBM SPSS Analytic Server, and IBM SPSS 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.

### The IBM SPSS Analytic Server context object

The execution context for a Python/Spark script is defined by an Analytic Server context object. 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 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. Creating and managing custom nodes

The Custom Dialog Builder allows you to create and manage nodes to use inside SPSS Modeler streams. Using the Custom Dialog Builder 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 25 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.

**Note:** You cannot create your own version of a node dialog for a standard IBM SPSS Modeler node.

### How to start the Custom Dialog Builder

From the menus, choose **Tools** > **Custom Dialog Builder**

**Note:**

- There are no standalone nodes for Python for Spark scripting like the R nodes.
- Python nodes created from Custom Dialog Builder depend on the Spark environment and will 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
- Python scripts must use the Spark API because data will be presented in the form of a Spark DataFrame.

## 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.

## 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" on page 24 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 29 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 25 for more information.
4. Install the node dialog to IBM SPSS Modeler and/or save the specification for the dialog to a custom dialog package (`.cfd`) file. See the topic "Managing custom node dialogs" on page 27 for more information.

You can preview your node dialog as you're building it. See the topic "Previewing a custom node dialog" on page 27 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. This is the name used to identify the node dialog when installing or uninstalling it. 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 is installed or saved to a custom dialog package file. The Help button on the run-time dialog is hidden if there is no associated help file.

Any supporting files, such as image files and style sheets, must be stored along with the main help file once the custom node dialog has been installed. By default, the specifications for an installed custom node dialog are stored in the `CDB/<Dialog Name>` folder of the installation directory for Windows and Linux. Supporting files should be located in the `Help` sub-folder of this folder. They must be manually added to any custom dialog package files you create for the dialog.

If you have specified alternative locations for installed node dialogs--using the `IBM_SPSS_MODELER_EXTENSION_PATH` environment variable--then store any supporting files under the `Help` sub-folder of the `<Dialog Name>` folder at the appropriate alternate location. See the topic "Managing custom node dialogs" on page 27 for more information.

**Note:** When working with a node dialog opened from a custom dialog package (`.cfd`) file, the Help File property points to a temporary folder associated with the `.cfd` file. Any modifications to the help file should be made to the copy in the temporary folder.

**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.

**Required Add-Ons.** Specifies one or more add-ons that are required in order to run the script that is generated by the node. In IBM SPSS Modeler, custom node dialogs can generate and run scripts that are written in R, for example. In this case, the required plug-in is IBM SPSS Modeler - Essentials for R. Users of your custom node dialog will be warned about required add-ons that are missing when they try to install or run the node.

**Script.** The Script property specifies the model building script template, used to create the R script or Python for Spark script that is generated and run by the node at run-time. Click the ellipsis (...) button to open the Script Template. See the topic "Building the script template" on page 25 for more information.

**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. Select one of the following options: **Model**, **Process**, or **Output**.

**Palette.** Specifies the palette to which the newly created node will be added when you install your node dialog. Select one from the following options: **Modeling**, **Modeling (Analytic Server)**, **Modeling (Classification)**, **Modeling (Association)**, **Modeling (Segmentation)**, **Output**, or **Export**.

**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 three functional columns in which you can place controls.

**Left Column.** The left column is primarily intended for a field chooser control. All controls other than sub-dialog buttons and tabs can be placed in the left column.

**Center Column.** The center column can contain any control other than sub-dialog buttons and tabs.

**Right Column.** The right column can only contain 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. From the menus in the Custom Dialog Builder, choose:

   **Edit** > **Script Template**

   (Or click the ellipsis (...) button in the **Script** property field in Dialog Properties)

2. 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.
3. 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 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 29 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 **Model 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**.

## Managing custom node dialogs

The Custom Dialog Builder allows you to manage custom node dialogs created by you or by other users. You can install, uninstall, or modify installed node dialogs; and you can save specifications for a custom node dialog to an external file or open a file containing the specifications for a custom node dialog in order to modify it. Custom node dialogs must be installed to all instances of SPSS Modeler Client or SPSS Modeler Batch where they are needed before 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.

You can install the node dialog that is currently open in the Custom Dialog Builder to a palette. Re-installing an existing node dialog will replace the existing version on the palette. In an open stream, the existing version of the node dialog 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.

To install the currently open node dialog, from the menus in the Custom Dialog Builder, choose **File** > **Install**.

For Windows XP, installing a node dialog requires write permission to the IBM SPSS Modeler installation directory (for Windows 8, Windows 7, and Windows Vista, node dialogs are installed to a general user-writable location). If you do not have write permissions to the required location or would like to store installed node dialogs elsewhere, you can specify one or more alternate locations by defining the *IBM_SPSS_MODELER_EXTENSION_PATH* environment variable. When present, the paths specified in *IBM_SPSS_MODELER_EXTENSION_PATH* take precedence over the default location. Custom node dialogs will be installed to the first writable location. For multiple locations, separate each with a semicolon on Windows. The specified locations must exist on the target machine. After setting *IBM_SPSS_MODELER_EXTENSION_PATH*, you must restart IBM SPSS Modeler for the changes to take effect.

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

### Windows XP and Windows 8

1. Select System.
2. Select the Advanced tab and click **Environment Variables**. For Windows 8, the Advanced tab is accessed from Advanced system settings.
3. In the User variables section, click **New**, enter *IBM_SPSS_MODELER_EXTENSION_PATH* in the Variable name field and the path(s) in the Variable value field.

### Windows Vista or Windows 7

1. Select User Accounts.
2. Select Change my environment variables.
3. Click **New**, enter *IBM_SPSS_MODELER_EXTENSION_PATH* in the Variable name field and the path(s) in the Variable value field.

## Opening an installed custom node dialog

You can open a custom node dialog that you have already installed, allowing you to modify it and/or save it externally so that it can be distributed to other users.

From the menus in the Custom Dialog Builder, choose **File** > **Open Installed**.

**Note:** If you are opening an installed node dialog in order to modify it, choosing **File** > **Install** will re-install 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.

## Uninstalling a custom node dialog

From the menus in the Custom Dialog Builder, choose **File** > **Uninstall**.

The node for the custom dialog will be removed the next time you start IBM SPSS Modeler.

## Saving to a custom dialog package file

You can save the specifications for a custom node dialog to an external file, allowing you to distribute the node dialog to other users or save specifications for a node dialog that is not yet installed. Specifications are saved to a custom dialog package (`.cfd`) file.

From the menus in the Custom Dialog Builder, choose **File** > **Save**.

**Note:** The default folder in which the `.cfd` file is saved is: `C:\ProgramData\IBM\SPSS\Modeler\<version number>\CDB`. Because the `ProgramData` folder is a system folder that is hidden by default, before you save you must either deselect the **`Hide System Folders`** option in your Folder Options, or type the direct path in the toolbar to navigate to the default folder.

## Opening a custom dialog package file

You can open a custom dialog package file containing the specifications for a custom node dialog, allowing you to modify the dialog and re-save it or install it.

From the menus in the Custom Dialog Builder, choose **File** > **Open**.

## Manually copying an installed custom node dialog

By default, the specifications for an installed custom node dialog are stored in the `CDB/<Dialog Name>` folder of the installation directory. You can copy this folder to the same relative location in another instance of IBM SPSS Modeler and it will be recognized as an installed node dialog the next time that instance is launched.

- If you specify alternative locations for installed nodedialogs--using the *IBM_SPSS_MODELER_EXTENSION_PATH* environment variable--then copy the `<Dialog Name>` folder from the appropriate alternate location.
- If alternative locations for installed node dialogs have been defined for the instance of IBM SPSS Modeler you are copying to, then you can copy to any one of the specified locations and the node dialog will be recognized as an installed node dialog the next time that instance is launched.

### Custom node dialogs in SPSS Modeler Batch or in IBM SPSS Collaboration and Deployment Services

To use custom dialog nodes 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 custom dialog nodes. For more information, see the previous section: Manually copying an installed custom node dialog. 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 custom dialog was created.

## 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" for more information.
- **Check Box:** A single check box. See the topic "Check Box" on page 30 for more information.
- **Combo Box:** A combo box for creating drop-down lists. See the topic "Combo Box and List Box controls" on page 31 for more information.
- **List Box:** A list box for creating single selection or multiple selection lists. See the topic "Combo Box and List Box controls" on page 31 for more information.
- **Text control:** A text box that accepts arbitrary text as input. See the topic "Text control" on page 32 for more information.
- **Number control:** A text box that is restricted to numeric values as input. See the topic "Number Control" on page 32 for more information.
- **Static Text control:** A control for displaying static text. See the topic "Static Text Control" on page 33 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 33 for more information.
- **Radio Group:** A group of radio buttons. See the topic "Radio Group" on page 34 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 35 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 35 for more information.
- **Tab:** A single tab. See the topic "Tab" on page 36 for more information.
- **Sub-dialog Button:** A button for launching a sub-dialog. See the topic "Sub-dialog button" on page 37 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. 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.

**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.

**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" for more information.

**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 and you can use `\n` for line breaks.
- The value `%%ThisValue%%` specifies the run-time value of the control, which is the list of fields. This is the default.

## Filtering Field Lists

The Filter dialog box, associated with field lists, 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 and you can use `\n` for line breaks.

## Combo Box and List Box controls

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 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 Combo Box and List Box controls have 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 Combo Box control or 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 (List Box only).** 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.

**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 and you can use `\n` for line breaks.

### 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 and you can use `\n` for line breaks.

**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.

## 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.

**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.

**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 and you can use `\n` for line breaks.
- 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.

## 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.

**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.

**Default Value.** The default value, if any.

**Minimum Value.** The minimum allowed value, if any.

**Maximum Value.** The maximum allowed value, 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. 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 and you can use `\n` for line breaks.
- 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.

## 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.

## 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 and you can use `\n` for line breaks.
- 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.

## 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 and you can use `\n` for line breaks.
- 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.

### 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.

**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.

**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 and you can use `\n` for line breaks.
- 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 and you can use `\n` for line breaks.
- 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.

## 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.

**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**.

**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 and you can use `\n` for line breaks.
- 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.

### 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.

## 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].

**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.

**Script.** Click the ellipsis (...) button to open the Script Template. See the topic "Building the script template" on page 25 for more information.

## 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

1. Make a copy of the properties file associated with the nodedialog. The properties file contains all of the localizable strings associated with the node dialog. By default, it is located in the `CDB/<Dialog Name>` folder of the IBM SPSS Modeler installation directory for Windows and Linux. The copy should reside in that same folder and not in a sub-folder.

   If you have specified alternate locations for installed node dialogs--using the *IBM_SPSS_MODELER_EXTENSION_PATH* environment variable--then the copy should reside in the `<Dialog Name>` folder at the appropriate alternate location. See the topic "Managing custom node dialogs" on page 27 for more information.
2. Rename the copy to `<Dialog Name>_<language identifier>.properties`, using the language identifiers in the table below. For example, if the dialog name is *mydialog* and you want to create a Japanese version of the nodedialog, then the localized properties file should be named `mydialog_ja.properties`. Localized properties files must be manually added to any custom dialog package files you create for the node dialog. A custom dialog package file is simply a `.zip` file that can be opened and modified with an application such as WinZip on Windows.
3. Open the new properties file 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*.

When the node dialog is launched, IBM SPSS Modeler will search 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` will be used.

To localize the help file

1. Make a copy of the help file associated with the node dialog and localize the text for the language you want. The copy must reside in the same folder as the help file and not in a sub-folder. The help file should reside in the `CDB/<Dialog Name>` folder of the IBM SPSS Modeler installation directory for Windows and Linux.

   If you have specified alternate locations for installed node dialogs--using the *IBM_SPSS_MODELER_EXTENSION_PATH* environment variable--then the copy should reside in the `<Dialog Name>` folder at the appropriate alternate location. See the topic "Managing custom node dialogs" on page 27 for more information.
2. Rename the copy to `<Help File>_<language identifier>`, 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`. Localized help files must be manually added to any custom dialog package files you create for the nodedialog. A custom dialog package file is simply a `.zip` file that can be opened and modified with an application such as WinZip on Windows.

If there are supplementary files such as image files that also need to be localized, you will need to manually modify the appropriate paths in the main help file to point to the localized versions, which should be stored along with the original versions.

When the node dialog is launched, IBM SPSS Modeler will search 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

# Index

Printed in USA