

# Introduction to R in IBM SPSS Modeler

Wannes Rosius



 Analytics

Big Data





International Technical Support Organization

**Introduction to R in IBM SPSS Modeler**

October 2016

**Note:** Before using this information and the product it supports, read the information in “Notices” on page v.

**First Edition (October 2016)**

This edition applies to Version 18, Release 03 of IBM SPSS Modeler (product number 5725-A65).

This document was created or updated on October 13, 2016.

**© Copyright International Business Machines Corporation 2016. All rights reserved.**

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Notices</b> .....	v
Trademarks .....	vi
<b>IBM Redbooks promotions</b> .....	vii
<b>Preface</b> .....	ix
Introduction to this paper .....	ix
Authors .....	x
Now you can become a published author, too! .....	x
Comments welcome .....	x
Stay connected to IBM Redbooks .....	xi
<b>Chapter 1. System setup</b> .....	1
1.1 Installing R .....	2
1.2 Enabling the R nodes .....	2
<b>Chapter 2. R basics</b> .....	3
2.1 Getting started with R .....	4
<b>Chapter 3. The basics of R nodes in IBM SPSS Modeler</b> .....	7
3.1 The R nodes .....	8
3.2 Simple R code example .....	8
3.2.1 modelerData .....	9
3.2.2 modelerDataModel .....	12
3.2.3 modelerModel .....	14
3.3 Some general remarks .....	15
3.4 Read data options .....	16
<b>Chapter 4. Custom Dialog Builder</b> .....	19
4.1 About the Custom Dialog Builder .....	20
4.2 Tools .....	20
4.3 Custom dialogs .....	21
4.4 Simple example .....	21
<b>Chapter 5. Tips and tricks</b> .....	29
5.1 R code .....	30
5.1.1 ibmspssc70 library .....	30
5.1.2 Some useful parts of R code .....	31
5.2 Custom Dialog Builder tips .....	33
5.2.1 How to save and share a custom dialog .....	33
5.2.2 Link to dialog and script .....	33
5.3 What about SQL Pushback? Hadoop Pushback? .....	35
5.4 What about real-time scoring? and IBM SPSS Modeler Solution Publisher? .....	36
5.5 More about the metadata in modeler and the consequences on R integration .....	37



# Notices

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

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

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

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

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

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

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

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

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

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

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

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

## COPYRIGHT LICENSE:

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

# Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

developerWorks®


IBM®

IBM PureData®

PureData®

Redbooks®

Redpaper™

Redbooks (logo) ®

SPSS®

WebSphere®

The following terms are trademarks of other companies:

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

Other company, product, or service names may be trademarks or service marks of others.



## Find and read thousands of IBM Redbooks publications

- ▶ Search, bookmark, save and organize favorites
- ▶ Get personalized notifications of new content
- ▶ Link to the latest Redbooks blogs and videos

Get the latest version of the Redbooks Mobile App



## Promote your business in an IBM Redbooks publication

Place a Sponsorship Promotion in an IBM® Redbooks® publication, featuring your business or solution with a link to your web site.

Qualified IBM Business Partners may place a full page promotion in the most popular Redbooks publications. Imagine the power of being seen by users who download millions of Redbooks publications each year!



[ibm.com/Redbooks](http://ibm.com/Redbooks)  
About Redbooks → Business Partner Programs

THIS PAGE INTENTIONALLY LEFT BLANK

# Preface

This IBM® Redpaper™ publication focuses on the integration between IBM SPSS® Modeler and R. The paper is aimed at people who know IBM SPSS Modeler and have only a very limited knowledge of R.

Chapters 2, 3, and 4 provide you with a high level understanding of R integration within SPSS Modeler enabling you to create or recreate some very basic R models within SPSS Modeler, even if you have only a basic knowledge of R.

Chapter 5 provides more detailed tips and tricks. This chapter is for the experienced user and consists of items that might help you get up to speed with more detailed functions of the integration and understand some pitfalls.

## Introduction to this paper

Although there are several very good articles and blogs related to IBM SPSS Modeler, many people still struggle with both R and the integration between IBM SPSS Modeler and R. The goal of this paper is to help with this situation.

At every point in the paper, we try to include R examples you can easily copy into the appropriate R node in SPSS Modeler. Unless specified otherwise, the code snippets are always based on the telco.sav data set which can be found in the demo folder of your SPSS Modeler installation. After the source node, attach a type node, and then the appropriate R node. However, sometimes there are just abstracts of code to show you the idea. We clearly indicate when the code is incomplete. You will find the code backs into several code frames throughout this document.

Some useful web addresses to help you get started:

- ▶ Essentials for R - Installation Instructions

[https://github.com/IBMPredictiveAnalytics/R\\_Essentials\\_Modeler/releases/download/18.0/SPSS\\_Modeler\\_R\\_Essentials\\_18.0\\_Installation\\_Doc\\_ML.zip](https://github.com/IBMPredictiveAnalytics/R_Essentials_Modeler/releases/download/18.0/SPSS_Modeler_R_Essentials_18.0_Installation_Doc_ML.zip)

- ▶ IBM SPSS Modeler Extensions

<ftp://public.dhe.ibm.com/software/analytics/spss/documentation/modeler/18.0/en/ModelerExtensions.pdf>

- ▶ IBM developerWorks® web page IBM SPSS Predictive Analytics Downloads

<https://developer.ibm.com/predictiveanalytics/downloads/>

- ▶ IBM developerWorks blog post - *SPSS Modeler and R integration - Getting started*

<https://developer.ibm.com/predictiveanalytics/2014/11/25/spss-modeler-and-r-integration-getting-started>

## Authors

This paper was produced by the following author:

**Wannes Rosius** is a data scientist based in Brussels, Belgium working for IBM within the center of excellence team of IBM predictive solutions. He has over a decade experience in data science across multiple industry sectors. He has experience in a wide variety of data science tools, including IBM SPSS, SAS, R, Python, and others. He holds Masters degrees in Mathematics and Statistics, and has an in-depth knowledge of applying data mining techniques. He is experienced in a wide range of industry application areas including customer churn, customer profitability, cross-selling, retail demand forecasting, fraud intelligence, CRM, econometric modelling, debt management, behavioral credit risk modelling, and site location.

Thanks to the following people for their contributions to this project:

Martin Keen, LindaMay Patterson  
International Technical Support Organization

## Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks® publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

[ibm.com/redbooks](http://ibm.com/redbooks)

- ▶ Send your comments in an email to:

[redbooks@us.ibm.com](mailto:redbooks@us.ibm.com)

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization  
Dept. HYTD Mail Station P099  
2455 South Road  
Poughkeepsie, NY 12601-5400

## Stay connected to IBM Redbooks

- ▶ Find us on Facebook:  
<http://www.facebook.com/IBMRedbooks>
- ▶ Follow us on Twitter:  
<http://twitter.com/ibmredbooks>
- ▶ Look for us on LinkedIn:  
<http://www.linkedin.com/groups?home=&gid=2130806>
- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:  
<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>
- ▶ Stay current on recent Redbooks publications with RSS Feeds:  
<http://www.redbooks.ibm.com/rss.html>





# System setup

This chapter discusses setting up your system. It is assumed that you have a valid installation of IBM SPSS Modeler on your machine. For more installation topics, see the installation instructions.

This chapter contains the following sections:

- ▶ Installing R
- ▶ Enabling the R nodes

# 1.1 Installing R

Depending on the version of your IBM SPSS Modeler, install the associated version of R as shown in Table 1-1.

Table 1-1 SPSS Modeler version to R version link

SPSS Modeler version	R version and download link
16.02.15.2	Download R 2.15.2 for Windows <a href="https://cran.r-project.org/bin/windows/base/old/2.15.2/">https://cran.r-project.org/bin/windows/base/old/2.15.2/</a>
17.03.1	Download R 3.1.0 for Windows <a href="https://cran.r-project.org/bin/windows/base/old/3.1.0/">https://cran.r-project.org/bin/windows/base/old/3.1.0/</a>
17.13.1	Download R 3.1.0 for Windows <a href="https://cran.r-project.org/bin/windows/base/old/3.1.0/">https://cran.r-project.org/bin/windows/base/old/3.1.0/</a>
18.03.2	Download R 3.2.0 for Windows <a href="https://cran.r-project.org/bin/windows/base/old/3.2.0/">https://cran.r-project.org/bin/windows/base/old/3.2.0/</a>

After you have downloaded and installed R, you have a working R instance on your computer. Similar to SPSS Modeler, you can have several versions of R installed on your computer without any problem.

# 1.2 Enabling the R nodes

You need to install the IBM SPSS Modeler essentials for R. Perform the following steps:

1. Go to the SPSS Community Downloads page to find the essentials, at this web address:  
<https://developer.ibm.com/predictiveanalytics/downloads/>
2. Select option **2 Get Essentials for SPSS** and click **Get R Essentials for SPSS Modeler**.
3. Now you are at github. Select and download the **Modeler 18 Essentials for R** for your particular platform. If you require Essentials for R for earlier Modeler versions, the page provides links to older versions.
4. Execute the installation. The installation asks you the path of your R installation and the path to the bin files of your SPSS Modeler installation.

**Note:** The prefilled path is the default path to a SPSS Modeler server. You need to change this path if you want to configure your client.

This installation places the R nodes in your SPSS Modeler node palette and includes the necessary R libraries in your R installation folder.





## R basics

There are a wide variety of R courses publicly available through several channels. It is not our intent to replace these courses. You do not need to be an R expert to use this document. However, there are some basics of R code and R terminology you need to understand to exploit the integration of R and IBM SPSS Modeler.

This chapter contains the following sections:

- ▶ Getting started with R

## 2.1 Getting started with R

Open R in its original graphical user interface (GUI), by going to the R installation folder and opening the `\bin\x64\RGUI.exe` file.

Figure 2-1 shows the R console.

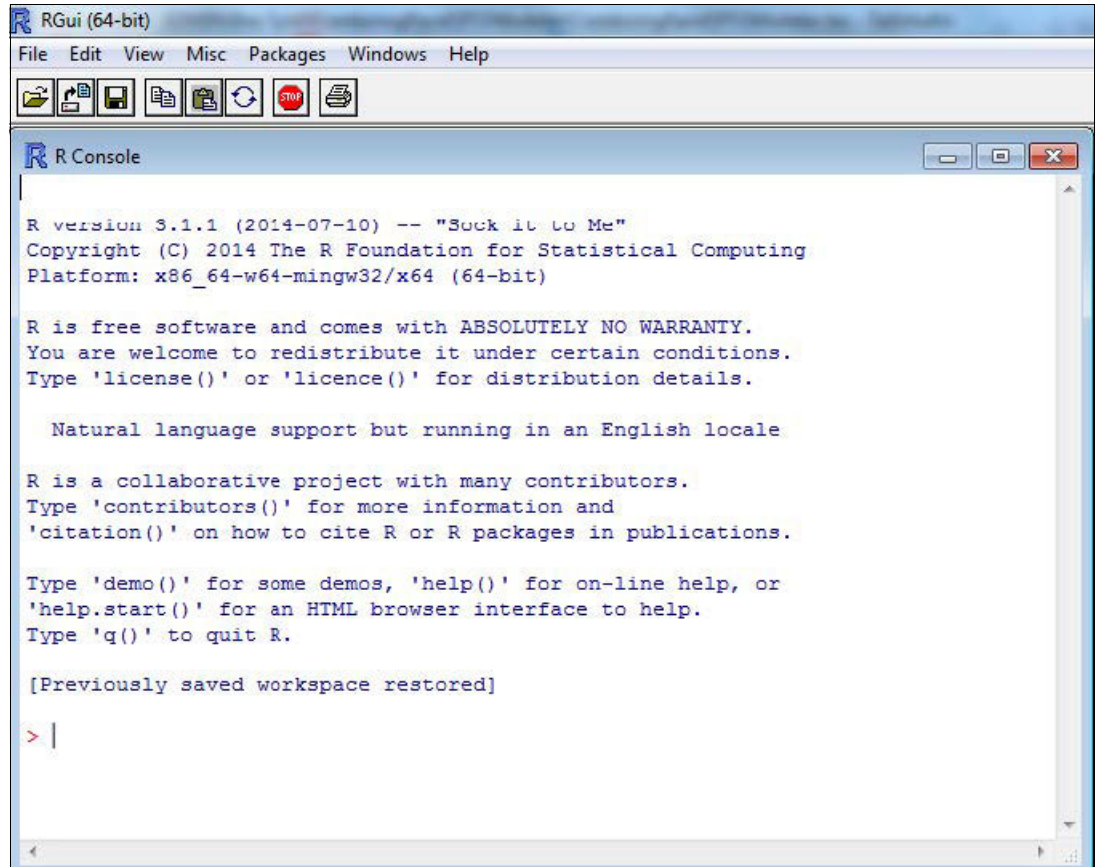


Figure 2-1 R console

The R console is ready to run commands. You might see the term RStudio, which is a development environment on top of this R GUI. You might prefer to use RStudio, which is a powerful and productive user interface for R. Installation of RStudio is not required for this introduction, but might be handy for future use.

R is a powerful programming language and environment for statistical computing and graphics. R is a programming language, unlike IBM SPSS Modeler. It is built on objects that are defined by the user. Example 2-1 shows R code you can type in the R console to see the R outputs.

*Example 2-1 R code*

```
x <- 1+1
y <- 2*x
xyVector <- c(x,y)
z <- mean(xyVector)
print(z)
```

In Example 2-1, *x* is an object. This statement fills the object *x* with the value of the evaluated formula  $1 + 1$  equals 2. Whenever the program refers to *x*, it is interpreted as 2. In the second line, *y* is defined as twice the value of *x*. In the third line, a vector is created containing the content of *x* and *y*, calculating the mean of these two objects and placing the result in an object *z*. The operator “<-” could also be replaced by “=”, but for various reasons many R users prefer to write the equation this way. Actually it is not exactly the same, but that is ignored for the purpose of this paper. If you feel more comfortable using “=”, then do so.

Similar to the way *x*, *y*, and *z* were populated with some numbers, any R object can be filled with a variety of types. The following list shows important types:

► Vector

Vector is a sequence of data elements of the same type (for example, numeric or character). This type includes vectors of length 1, which can be interpreted as just being numbers. You can create a vector with the R function `c()`. Example 2-1 on page 4 shows all the values of *x*, *y* and *z* are vectors of length one. *xyVector* is a vector of length 2, containing the values of (the vector) *x* followed by (the vector) *y*. Trying to link it back to SPSS Modeler, you can interpret a vector as the values of a single data column.

► Data frame

Data frame is a list of vectors of equal length. If you look at a vector as the values of a variable, a data frame could be interpreted as a two dimensional data set with columns (the number of vectors) and lines (the size of each vector). Example 2-2 shows the use of a data frame.

*Example 2-2 Using a data frame*

---

```
n <- c(2, 3, 5, 3, 9) #A first vector of 5 numeric values
n2 <- c(1, 3, 2, 5, 4) #A second vector of 5 numeric values
s <- c("aa", "bb", "cc", "aa", "zz") #A third vector of 5 string values
b <- c(TRUE, FALSE, TRUE, TRUE, TRUE) #A fourth vector of 5 flag values
Data <- data.frame(n, s, b, New = n+n2) #A data frame containing 4 vectors
#Note n+n2 will be a new vector called "New" with the sum of the n + n2: c(3,
6, 7, 8, 13)
dim(Data) #Will show you it is a 5x4 dataset.
Data[2,4] #Will give back the value on the 2nd line, the 3rd column
colnames(Data) #Will give the column names as a vector ("n","s","b", "New")
Data$n[1] #Will give back the first value of the vector n within the data
frame.
iris #predefined data frame.
```

---

There are also several predefined data frames installed within R. One of them is called *iris*.

**Note:** A data frame is a special type of a list where all the elements are vectors of equal sizes.

► Model class

Model class is actually a specific list containing predefined objects defining a statistical model. For example, a linear model class can be a list containing, among others, the coefficients of the regression model.

► List

List is an ordered collection of objects. As an example, you can have a list where the first element is a vector, the second is a data frame, and the third is a model.





## The basics of R nodes in IBM SPSS Modeler

This chapter discusses the use of R with IBM SPSS Modeler.

This chapter contains the following sections:

- ▶ The R nodes
- ▶ Simple R code example
- ▶ Some general remarks
- ▶ Read data options

## 3.1 The R nodes

After R essentials is installed, you see three new nodes in your node palettes. There is also a fourth R node, which is the R nugget. The four objects have these characteristics:

- ▶ **Output**  
The output node causes data to be sent to R. Output never goes back to SPSS Modeler because it is a terminal node. The only thing that can go back to SPSS is the outputs generated by R presented within an SPSS output window.
- ▶ **Transform**  
For the transform node, data goes from SPSS to R and goes back to SPSS enabling the SPSS process to continue.
- ▶ **Model**  
The model node is similar to the output node because it is a terminal node, which means that data does not go back to SPSS. However, there is a reusable R object created within a nugget.
- ▶ **Nugget**  
The nugget node is similar to the transform node. The nugget is a reusable R object that can be used in the R code.

Table 3-1 shows basic information about various R nodes.

Table 3-1 Node details

Node	R Output	R Transform	R Model	R Syntax
Name	R output node	R transform node	R model node	R syntax node
Palette tab	Output	Record Ops	Model	N/A
Data back to SPSS	No	Yes	No	Yes
Reusable R object	No	No	Create	Use

## 3.2 Simple R code example

All the examples in this section are intentionally kept simple to explain the interaction in a functional and structured way. The examples are simple enough for non R programmers to understand. Most of the R code snippets in this chapter could easily be implemented using standard SPSS Modeler functionality.

There are three important and reserved R objects that you should keep in mind when you use the SPSS Modeler R integration:

- ▶ `modelerData`  
The `modelerData` object is a R data frame filled by the data entering in this R node. This data frame can be used and changed within your R code. Eventually, it is this data frame that will be send back to SPSS Modeler as a data set. It only contains the content of the data, not (necessarily) the data column names and other metadata items.

► `modelerDataModel`

The `modelerDataModel` object is a R data frame that contains the metadata of the data that is sent to R and back to SPSS Modeler. It contains most of the information that you might expect within an SPSS Modeler type node. This object might seem strange for experienced R users.

► `modelerModel`

The `modelerModel` R object can be filled by the user with any type of R object. It does not need to have a certain structure. It is calculated in the R model node, after which it is saved within the R nugget, where it can be used in the R-syntax.

**Note:** R code is case sensitive and so are these object names.

In the following sections, the usage of these objects is explained.

### 3.2.1 `modelerData`

If data flows into any R node within SPSS Modeler, the data is converted to a data frame called `modelerData`. You can use this data frame to perform the desired calculations, transformations, and outputs in R.

Example 3-1 shows manipulating tenure data using a `modelerData` R object.

*Example 3-1 Summary results of data using `modelerData`*

---

```
#Print the first 6 lines of the data
head(modelerData)
#Give a summary of the data
summary(modelerData)
#create a histogram of the variable tenure
hist(modelerData$tenure, xlab = "years", main = "Tenure histogram")
#change the tenure unit from months to years
modelerData$tenure <- modelerData$tenure/12
#recreate the histogram, now in months
hist(modelerData$tenure, xlab = "months", main = "Tenure histogram")
```

---

Execution of this node results in an SPSS Modeler output window with all the R outputs assembled. These outputs are always divided in two tabs: Text output (see Figure 3-1 on page 10) and graph output (see Figure 3-2 on page 11). In this case the text output is linked to the code on line 2 and 5. This code prints the first 6 lines (head) of the data, and then provides summary statistics for each column.

Figure 3-1 shows the R text output.

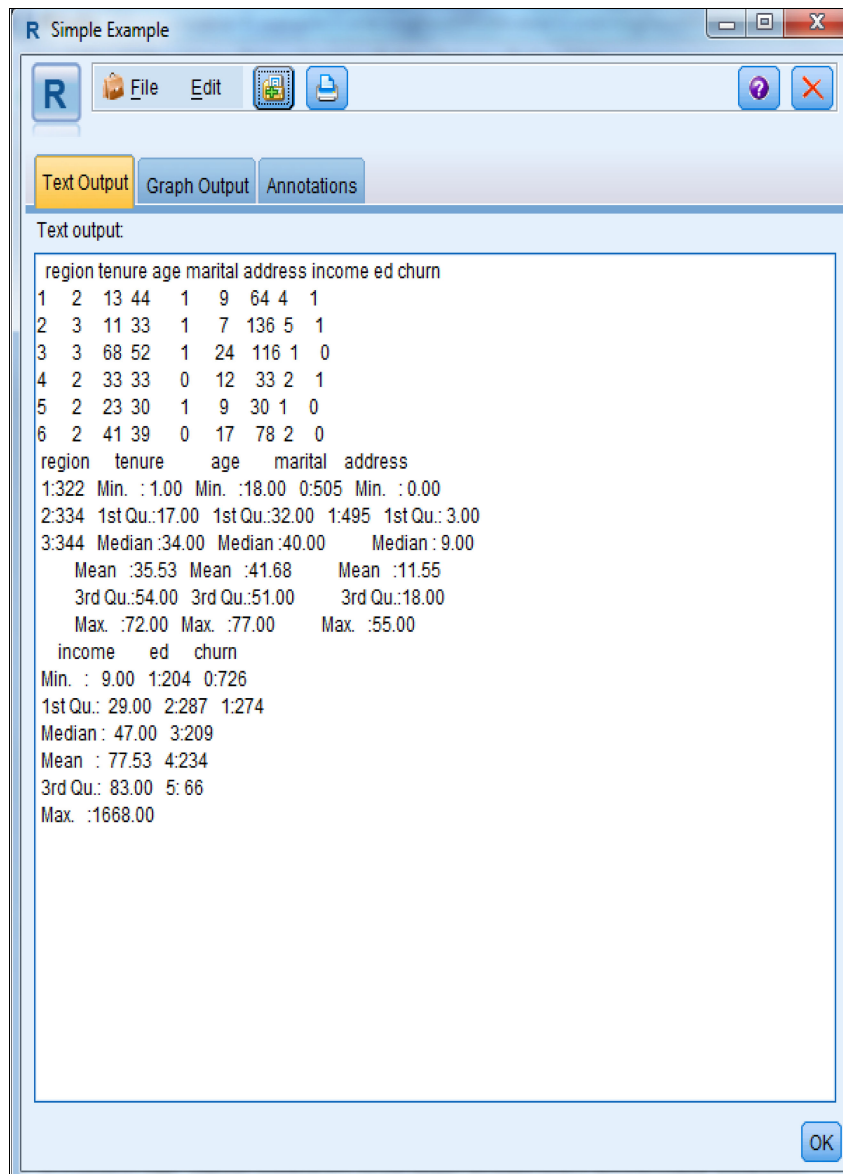


Figure 3-1 R text output



Figure 3-2 shows R graph output.

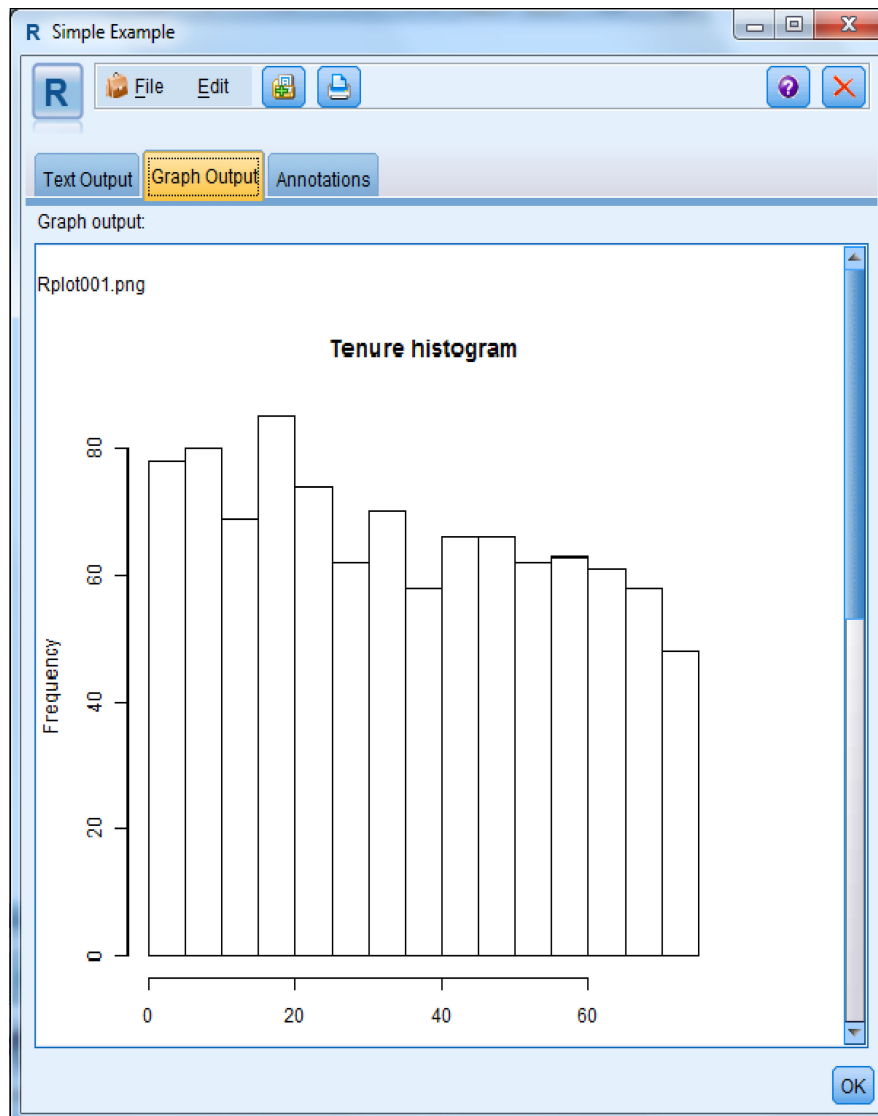


Figure 3-2 R graphic output

The graph output (Figure 3-2) is two histograms. One for the tenure in months. The other is the same column, after dividing the original value by twelve to give the tenure in years (notice the X-axis scale).

As shown in the example stream “Explain modelerData.str”(Example 3-1), you can also copy exactly this same code into a transform node and attach a table node to it. After running this table node, you do not see any R output (as none is expected). That means that, even though the output code has run, no outputs are given. However the data frame of modelerData is sent back to SPSS Modeler. In this case you see the value of tenure being divided by 12.

### 3.2.2 modelerDataModel

Metadata is very important in SPSS Modeler. Within the modeler, metadata is represented by the type node, which specifies the type of each variable in the dataset (such as, numeric, ag, String, and storage). At all times, modeler will know exactly all the metadata at every step in the stream.

R does not handle the metadata in a similar way as SPSS Modeler. The `modelerDataModel` taking over the role of the “type node”. This `modelerDataModel` is a data frame of the following structure (see Table 3-2).

Table 3-2 Data fields

Field	X <sub>1</sub>	X <sub>2</sub>	...	X <sub>n</sub>
fieldName	Region	Tenure	...	Age
fieldLabel	Geographic indicator	Months with services	...	Age in years
fieldStorage	Real	Real	...	Real
fieldMeasure	Nominal	Continuous	...	Continuous
fieldFormat	Standard	Standard	...	Standard
fieldRole	Input	Input	...	Input

This data set always has six lines with fixed names (in R, the lines have names). The thing with this data set is that it is completely the responsibility of the user to align this metadata with the appropriate data. If you would like to add a variable with R, you must manually add a column in `modelerDataModel` to make sure `modelerData` correctly goes back to SPSS Modeler. Example 3-1 on page 9 did not make any changes to the `modelerDataModel` and it was not needed because the metadata did not change (dividing a number by 12 does not change the metadata).

Based on Example 3-1 on page 9, rather than changing the value of tenure in the same data variable, we will create another one. As a result, we have to update the metadata (shown in Example 3-2).

Example 3-2 Tenure in a new variable

---

```
#Create the vector of tenure in years:
Rcolumn <- modelerData$tenure/12
#Paste this vector to the right of the dataset:
modelerData <- cbind(modelerData,Rcolumn)
#create the metadata for the column to add
newVar <- c(fieldName="tenureYears", fieldLabel="",fieldStorage="real",
fieldMeasure="",
fieldFormat="", fieldRole="")
#paste the new column metadata to the existing metadata
modelerDataModel <- cbind(modelerDataModel,newVar)
```

---

Running a table node downstream of this transform node, shows you the new variable with the name tenureYears. Following are some important things to realize about this code change:

- ▶ `fieldName` and `fieldStorage` are the only two required rows that need to be filled in for any new column. All the other lines of code are empty, meaning they are filled in by the stream default. For a list of available values, refer to the *IBM SPSS Modeler Extensions* document at this web address:
   
<ftp://public.dhe.ibm.com/software/analytics/spss/documentation/modeler/18.0/en/ModelerExtensions.pdf>
- ▶ Because `modelerDataModel` is only useful when you go back to SPSS Modeler, you generally only use/change this object in non-terminal R-nodes. It might still be handy to use it in terminal nodes, if the value of the `modelerDataModel` is important for your output. For example, to run a histogram of all continuous variables.
- ▶ When data goes back to SPSS modeler, it is the content of the data frame excluding the column and row names. That means, that even though the column in the `modelerData` is called `Rcolumn`, the name of the column in SPSS Modeler is defined by the metadata within the row `fieldName`. In this case it is called `tenureYears`.
- ▶ The only link between `modelerData` and `modelerDataModel` is the order of the columns. The first column in the data is given the metadata of the first column of `modelerDataModel`. If the metadata (`modelerDataModel`) does not match the `modelerData`, an error is produced. Figure 3-3 shows a schematic of how this works.

You can think of it this way: There are two tables in R, which leads to one table in SPSS Modeler. The first table only takes the values (x's). The names are taken from the second table. Because of this situation, make sure that the first column of `xs` correspond to the first column in the second table.

	modelerData			modelerDataModel			
	$RName_1$	...	$RName_n$		$X_1$	...	$X_n$
R	$x_{1,1}$	...	$x_{1,n}$	fieldName	$Name_1$	...	$Name_n$
	$x_{2,1}$	...	$x_{2,n}$	fieldLabel	.	...	.
	⋮	⋮	⋮	fieldStorage	XXX	...	XXX
	$x_{m-1,1}$	...	$x_{m-1,n}$	fieldMeasure	.	...	.
	$x_{m,1}$	...	$x_{m,n}$	fieldFormat	.	...	.
				fieldRole	.	...	.
SPSS					$Name_1$	...	$Name_n$
					$x_{1,1}$	...	$x_{1,n}$
					$x_{2,1}$	...	$x_{2,n}$
					⋮	⋮	⋮
					$x_{m,1}$	...	$x_{m,n}$
Note that only the names of the <code>modelerDataModel</code> are used							

Figure 3-3 Schematic of `modelerDataModel`

This concept might seem unusual to standard R users. To people who know SPSS Modeler, you can summarize it as `modelerDataModel` taking over the role of the type node.

### 3.2.3 modelerModel

modelerModel is the R object that is stored within the R nugget. This object is populated within the R model node, after which you could use modelerModel within the R nugget for scoring. This approach works the same way as IBM Modeler works, in that you ask a model node to calculate a formula. After the calculation, that formula is stored within the nugget, together with the way it should be used to calculate a scoring.

You only use this object within the R model node and nugget. Within the R model node, there are two syntax windows:

- ▶ R model building calculates whatever you want to store within modelerModel that could be used within your nugget calculations to score your data. This thing can be any object within R. As any SPSS Model builder node, this is a terminal code, meaning no data goes back to SPSS Modeler. Optionally, some outputs and the modelerModel are stored within the nugget.
- ▶ R model scoring is the syntax to define how you use the object modelerModel. The modelerModel object contains what you assigned to it in the R model building syntax. After you have done this, you can use modelerModel to derive new data. Apart from the use of modelerModel, this is very similar to the R transform node.

Example 3-3 shows creating a basic linear model for the variable tenure. This model's formula should be saved in the modelerModel, after which it can be used in the scoring.

#### *Example 3-3 Create and save a modelerModel*

---

```
#Create the model and save it in modelerModel
modelerModel <- lm(tenure ~ age + region + ed + income, data= modelerData)
#Add some summary of the model in the nugget
summary(modelerModel)
#together with a histogram
hist(modelerModel$residuals, main = "residual histogram")
#and the residual vs actuals scatterplot
plot(modelerData$tenure, modelerModel$fitted.values, xlab = "actual", ylab =
"predicted" )
#All of these output will be stored in the modeler nugget tabs
```

---

```
#Use the model to make a prediction, and add it to the existing data.
pred <- predict(modelerModel, modelerData)
modelerData <- cbind(modelerData,pred)
#Take care of the metadata!
newVar <-c(fieldName="$L-tenure", fieldLabel="", fieldStorage="real",
fieldMeasure="",
fieldFormat="", fieldRole="")
modelerDataModel <- cbind(modelerDataModel,newVar)
```

---

It is important to note that modelerModel can be filled with any type of object, but is very often a model class. Example 3-3 shows using the modelerModel for predictions. The object stored was clearly a (statistical) model.

Example 3-4 on page 15 shows saving two numbers (M and SD) within the modelerModel object. Imagine you want to calculate the z-values of a certain variable. To create the z-values, you need the mean and the standard deviation of the column. Store both of these values within modelerModel, after which you can use them in the scoring syntax.

**Note:** There is a very good reason this example is not combined in an R Transform node. For further details, see 3.4, “Read data options” on page 16.

Example 3-4 shows you do not need to store a statistical model within your `modelerModel` object. The results can be any R object.

*Example 3-4 Calculate mean and standard deviation and calculate z scores*

---

```
#calculate mean and standard deviation
M <- mean(modelerData$tenure)
SD <- sd(modelerData$tenure)
#and save it in a list called modelerModel.
modelerModel <- list(avg = M, sDev = SD)

#calculate z scores using the elements in modelerModel
zTenure <- (modelerData$tenure - modelerModel$avg)/modelerModel$sDev
modelerData <- cbind(modelerData,zTenure)
#define new metadata column and add it.
newVar <- c(fieldName="zTenure", fieldLabel="",fieldStorage="real",
fieldMeasure="",
fieldFormat="", fieldRole="")
modelerDataModel <- cbind(modelerDataModel,newVar)
```

---

### 3.3 Some general remarks

This section covers general remarks and examples to help you speed up by using this interaction between R and SPSS Modeler:

- ▶ You are not required to build `modelerData` from the existing data within that frame. `modelerData` is filled with the dataset you have in Modeler. However, nothing stops you from throwing that data away in R and defining some new data coming from another data source in R. As an example, imagine a link from The Weather Company web site. This website gives the weather history in Brussels, Belgium in the month of November 2015. Example 3-5 shows R code to retrieve the weather data (into `modelerData`) and redefining the data into a `modelerDataModel` object.

*Example 3-5 Access weather data and redefine as modelerDataModel*

---

```
#Define the link
linkPath <- "http://www.wunderground.com/history/airport/EBBR/2015/11/01/
MonthlyHistory.html?req_city=Brussels&format=1"
#Read the data as csv
modelerData <- read.csv(linkPath)
modelerData[,1] <- as.Date(modelerData[,1])
#Redefining modelerDataModel, all are real numbers, except the first column is
the date.
modelerDataModel <- as.data.frame(t(data.frame(fieldName =
colnames(modelerData),
fieldLabel = "", fieldStorage = c("date",rep("real",ncol(modelerData)-1)),
fieldMeasure = "", fieldFormat = "", fieldRole = "")))
```

---

As you can see, this code does not use the old definition of the defined R objects, but completely redefines them. Placing this in a R transform node gives this new dataset to the modeler. You can use this approach to create an R input node.

- ▶ Within an R model node, there are places for two scripts. The building script is the script populated within the R nugget. It is not run when you run the model node. As a result, these two scripts are independent. The only thing the scripts share is the value of the object `modelerModel`, which is saved within the nugget when running the building syntax and picked up within the R scoring syntax. Eventual R-libraries that are required should be loaded in both scripts. Example 3-6 shows a model for a random forest and using the `randomForest` library to make a prediction, adding it to the existing data.

*Example 3-6 Random forest model and making predictions*

---

```
#Load the library
library("randomForest")
#Create the model and save it in modelerModel
modelerModel <- randomForest(tenure ~ age + region + ed + income, data=
modelerData, ntree=50)
```

---

```
#Load the library
library("randomForest")
#Use the model to make a prediction, and add it to the existing data.
pred<- predict(modelerModel, modelerData)
modelerData <- cbind(modelerData,pred)
#Take care of the metadata!
newVar <-
c(fieldName="$RF-tenure",fieldLabel="",fieldStorage="real",fieldMeasure="",
fieldFormat="",fieldRole="")
modelerDataModel <- cbind(modelerDataModel,newVar)
```

---

- ▶ A package is a collection of R objects defined for a certain purpose. These packages often are specific statistical functionalities, such as `randomForest` in Example 3-6. A basic R installation comes with the standard packages, however, there are many more packages available that are made available by the R community, on CRAN<sup>1</sup>.

Packages need to be installed and made locally available in libraries. After the package is installed on the system as a library, you can load this library in any R session by the code `library(<name>)`. To install a package, you have several options. The easiest is to write a code such as `install.packages("randomForest")` within R. You select a CRAN mirror where this library is downloaded from, and the download is automatic. Normally you do this activity once. Although possible, it is not recommended to run this package installation command from within SPSS. The reason is that these libraries are saved in a temporary folder and afterwards are deleted. If you still want to do this through SPSS, you have to hard code the installation path.

## 3.4 Read data options

This section discusses settings within the node under Read data options. The basics of the R integration with SPSS Modeler can be done without the knowledge of the settings, as more advanced R knowledge is helpful. The user guide still has a good explanation of these items. However, there is one more thing that might be important. For SPSS Modeler version 17 and

<sup>1</sup> CRAN, <https://cran.r-project.org/>

lower, the R integration of non-terminal nodes (that is transform and nuggets), are by default done in batches of 1000. This approach allows these R nodes to work on Hadoop and other clustered environments. As a result, it is very important to realize that any R code that would span multiple lines of data would lead to false results. For a workaround for this, see 5.1.1, “ibmspsscf70 library” on page 30.

In Example 3-4, you could calculate the mean and the standard deviation of the variable in a non-terminal node. This action would start with running this code for the first 1000 lines of data. That leads to a specific mean, deviation, and corresponding z-scores. However, the next 1000 lines, a new mean and deviation would be calculated and the z scores are based on these new values. To solve this situation, the means and standard deviations are calculated in the R model (that is a terminal node) over all the data and used in the R nugget to calculate the z scores.

**Note:** This approach might lead to a very slow integration between SPSS and R in the case of streaming R nodes in a local, non-clustered environment. However, as of IBM SPSS Modeler version 17.1, there is a default option not to use this approach of batch processing or to increase the batch size. For the lower versions, there is a workaround possible if you still want to increase this batch size or turn it off.







# Custom Dialog Builder

This chapter discusses the Custom Dialog Builder.

This chapter contains the following sections:

- ▶ About the Custom Dialog Builder
- ▶ Tools
- ▶ Custom dialogs
- ▶ Simple example

## 4.1 About the Custom Dialog Builder

The Custom Dialog Builder allow you to create and manage R nodes with prefilled R code to use inside IBM SPSS Modeler streams. In this way, users can create their own nodes. You can start the Custom Dialog Builder in the **Tools** menu under **Custom Dialog Builder**.

When opening Custom Dialog Builder, you see two windows. One window is the custom dialog (Figure 4-1) itself, the other is the toolset to populate the dialog.

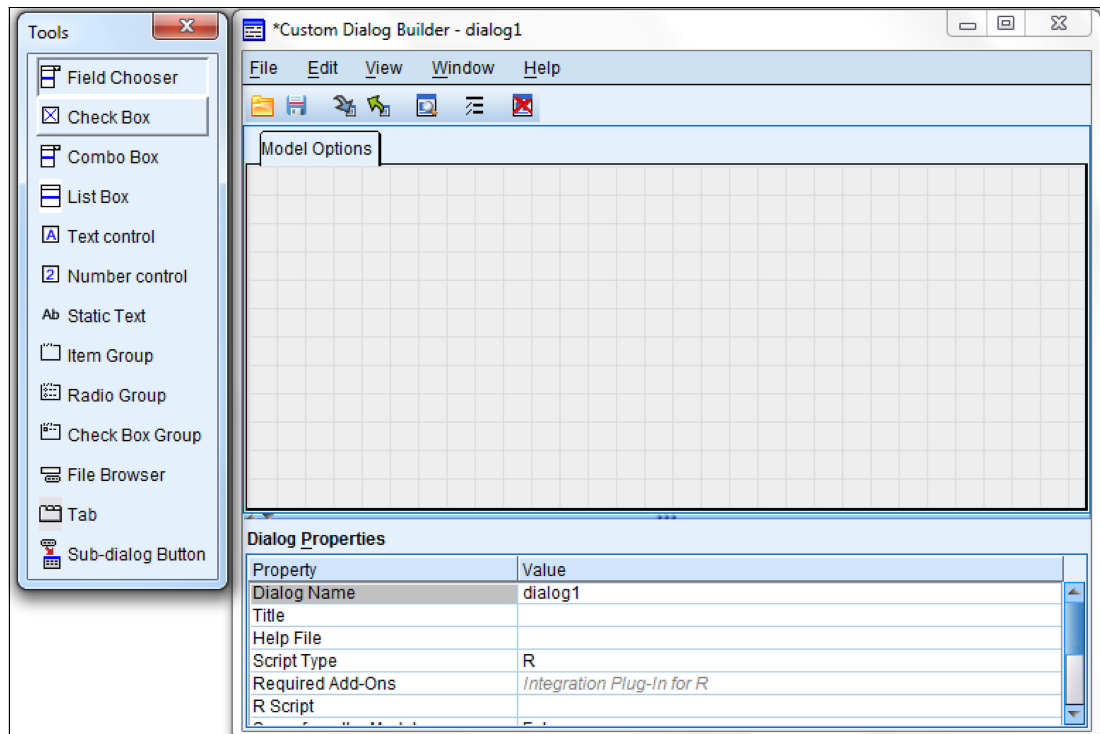


Figure 4-1 Custom Dialog Builder

## 4.2 Tools

The tools window is a list of items you can place within your dialog. This window includes the field chooser, check and combo boxes, text and number controls, tabs, and more. You can select any item and drag it onto the dialog. You can select any of these items and drag them onto the dialog.

After you have an item in the dialog, you can select it so you see the item properties. These properties are specific to this item and might change dependent on the type of item. The most important are the identifier (the way it is referenced within the script) and the Title (the one that is visual in the dialog).

## 4.3 Custom dialogs

The big gray window is the dialog itself. For the moment it is empty waiting to be populated with items from the **Tools** list. Click this gray dialog to show the dialog properties. Similar to main items, the dialog includes the name and title of the dialog, the script, and the type and position of the created node.

With regards to the script to be written, the global rule is that you reference the items within the dialog using their identifier between double percentages: `%%<identifier>%%`.

After you finish creating the custom node, you can install it by clicking the green arrow in the toolbar. You can also save intermediate versions to disk.

## 4.4 Simple example

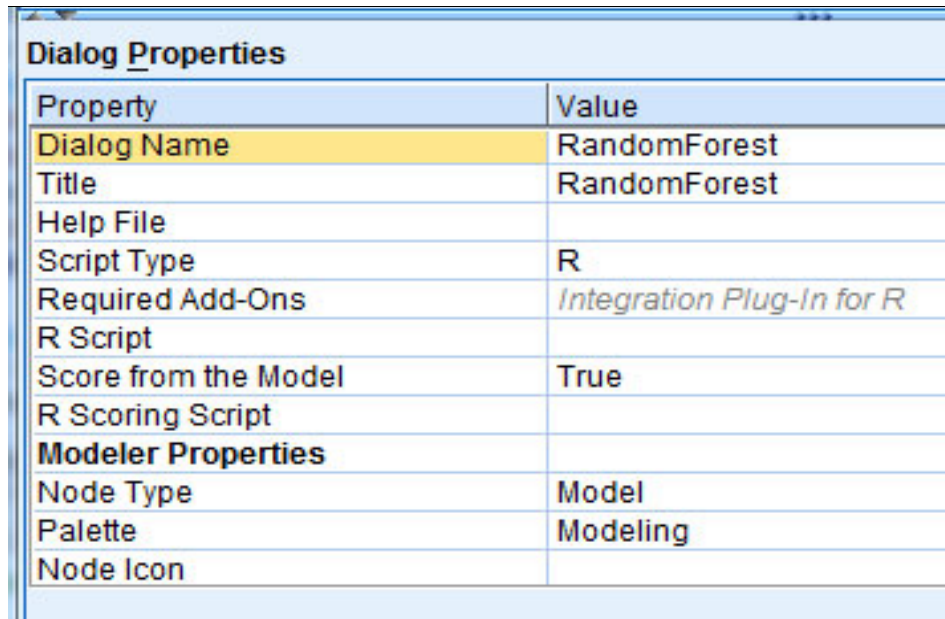
Create a custom dialog for the `randomForest` model (Example 3-6 on page 16) created in 3.3, “Some general remarks” on page 15.

The most important thing is ensuring flexibility in the code for the user. In the case of this model, there might be three things that you want flexible:

- ▶ Input variables
- ▶ Target
- ▶ Number of trees in the forest

Perform the following steps to create a custom dialog:

1. Fill in the custom dialog properties as shown in Figure 4-2.



Property	Value
Dialog Name	RandomForest
Title	RandomForest
Help File	
Script Type	R
Required Add-Ons	<i>Integration Plug-In for R</i>
R Script	
Score from the Model	True
R Scoring Script	
<b>Modeler Properties</b>	
Node Type	Model
Palette	Modeling
Node Icon	

Figure 4-2 Dialog properties for RandomForest

2. In this fixed example, the target is tenure, but a user might choose another field. As a result, place a field chooser on the dialog. Change the properties as shown in Figure 4-3. The variable filter properties allows you to select only categorical variables.

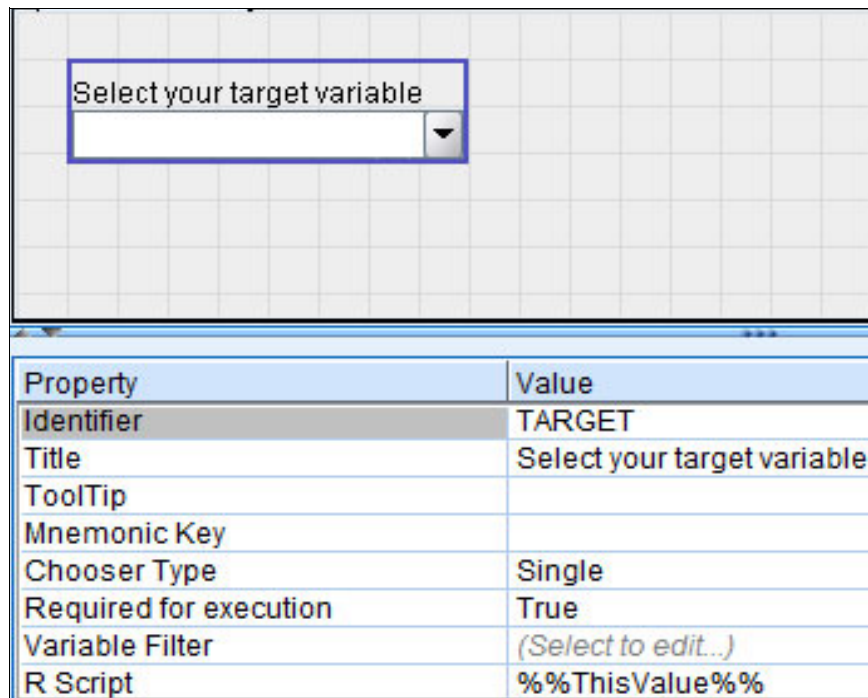


Figure 4-3 TARGET variable

3. In the example, the inputs include age and income. In some cases, the user could choose any other field. To support this, you need to place a field chooser on the dialog. The difference is that you can select several variables as there might be different inputs. To make it easier, you can separate these values by a "+" (plus) sign. Change the properties as shown in Figure 4-4.

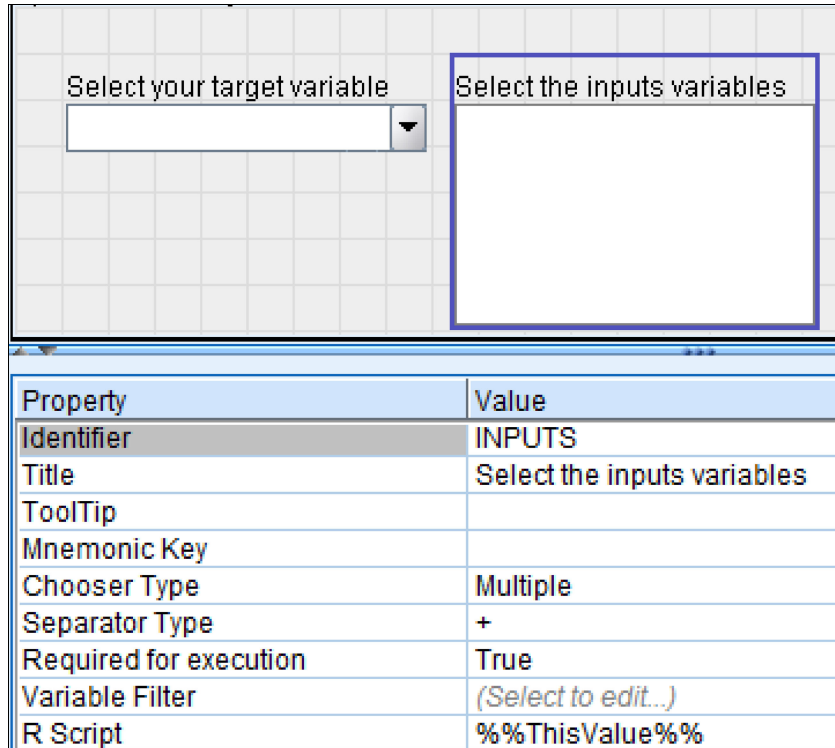


Figure 4-4 Change INPUTS property

4. Add the number of trees in the forest. In the original script the value was 50, which is the default. However, you can choose any integer value between 1 and 1000. Add a number control on the dialog and change the properties (Figure 4-5).

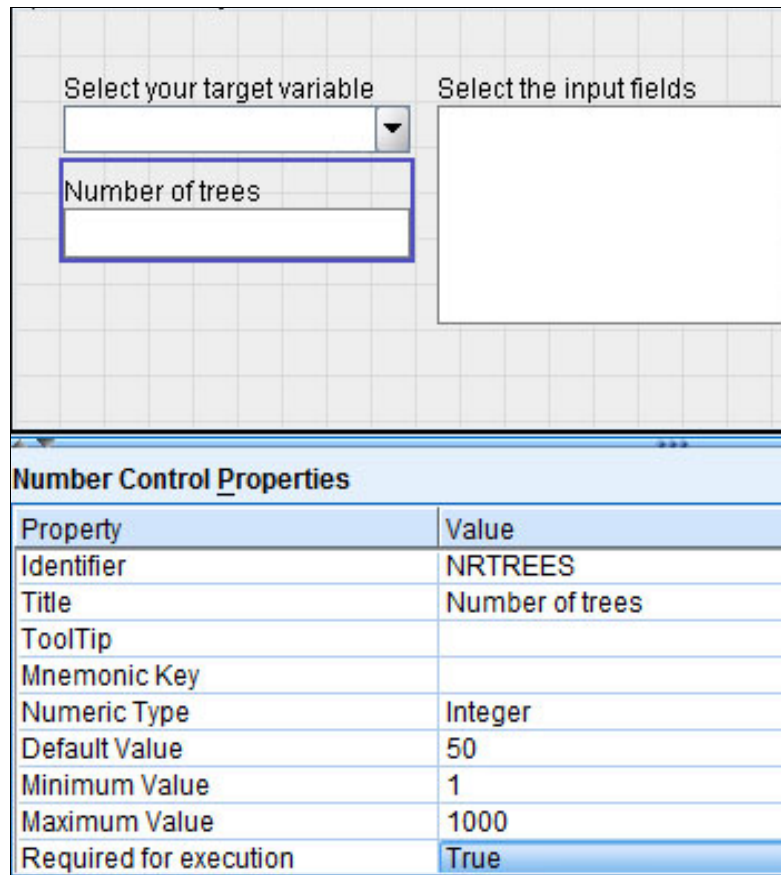


Figure 4-5 Set number of trees

5. The dialog is ready. Next add the script to the dialog. Select the **Edit** option and choose **Script Template**. This action brings you to an empty window for the script. In this case there is a tab for building code and one for the scoring script. If the scoring script is greyed out, you set the dialog property “Score from the Model” to True. See Figure 4-6.

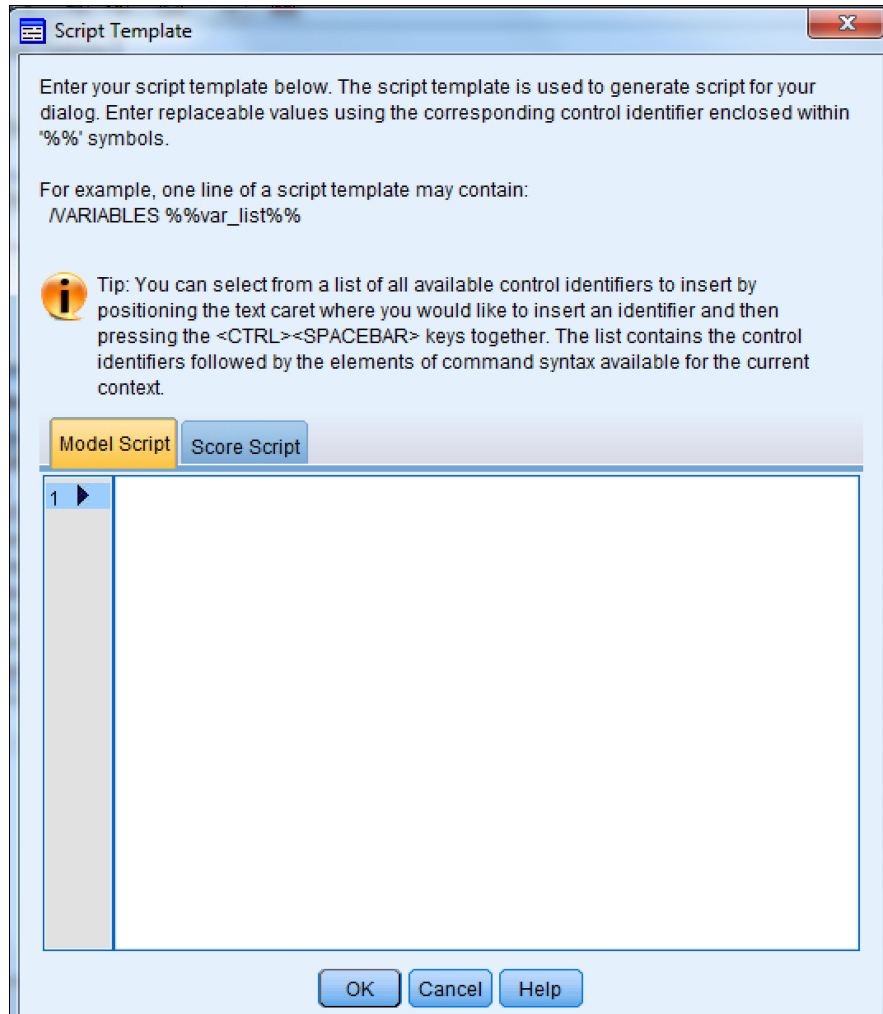


Figure 4-6 Script template

- Start with the scoring script because it is easier. The only thing that needs to be adapted for custom input is the variable name sent back to SPSS Modeler. Copy the scoring code and change the name tenure to %%TARGET%% (this is the name of the identifier of the target). See Figure 4-7.

```

Model Script | Score Script
1 #Load the library
2 library("randomForest")
3
4 #Use the model to make a prediction, and add it to the existing data.
5 pred<- predict(modelerModel, modelerData)
6 modelerData <- cbind(modelerData,pred)
7
8 #Take care of the metadata!
9 newVar <- c(fieldName="SRF-%%TARGET%%",fieldLabel="",fieldStorage="real",fi
10 modelerDataModel <- cbind(modelerDataModel,newVar)
11
  
```

Figure 4-7 Score script using %%TARGET%%

- Fill in the code for the building script (Figure 4-8) and change as shown for the target (Figure 4-3 on page 22) and set the values and input variables, together with the number of trees. Press **OK** to close the script window.

```

Model Script | Score Script
1 #Load the library
2 library("randomForest")
3
4 #Create the model and save it in modelerModel
5 modelerModel <- randomForest(%%TARGET%%~%%INPUTS%%, |data= modeler
6
7 #print the summary of the modelerModel
8 summary(modelerModel)
9
  
```

Figure 4-8 Build script



8. At the Custom Dialog Builder, save the dialog in the appropriate location (Figure 4-9). To deploy the dialog, click the green deploy arrow in the toolbar. Close the dialog builder.

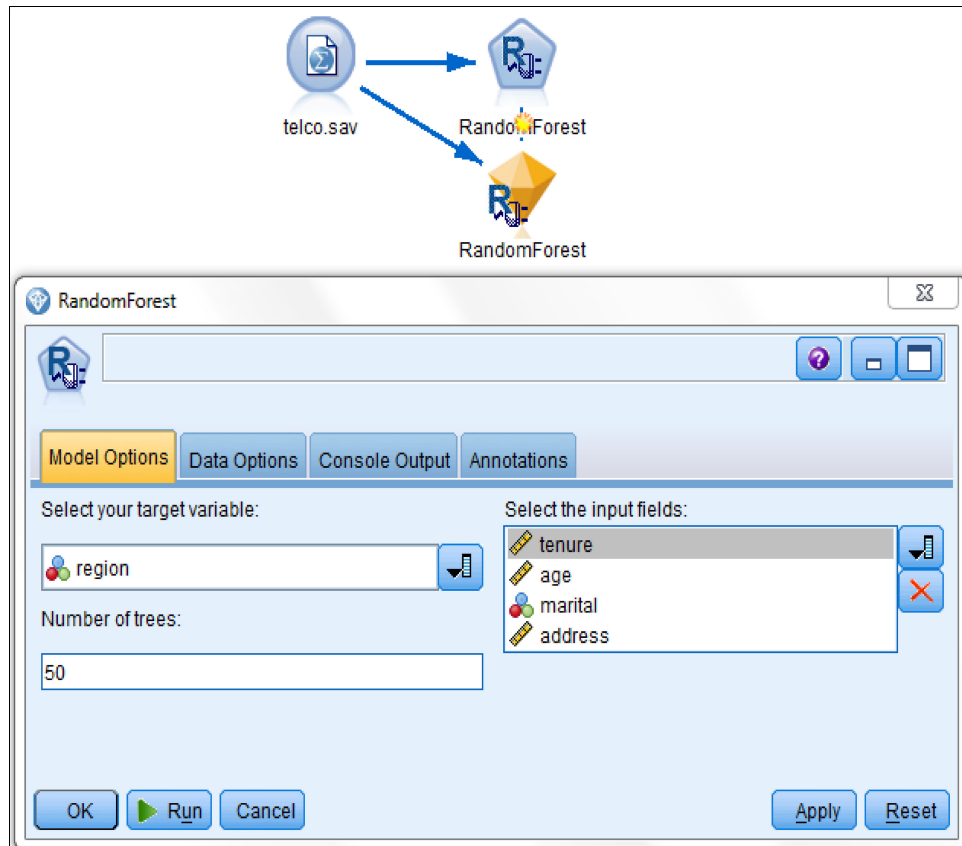


Figure 4-9 Save Dialog

You see the new node in the model palette. You can use this node within your stream.

Place the resulting `cfe` file in the correct location as explained in 5.2.1, “How to save and share a custom dialog” on page 33) and a stream where it is deployed.





## Tips and tricks

This chapter provides a detailed view into what actually happens with the code.

This chapter contains the following sections:

- ▶ R code
- ▶ Custom Dialog Builder tips
- ▶ What about SQL Pushback? Hadoop Pushback?
- ▶ What about real-time scoring? and IBM SPSS Modeler Solution Publisher?
- ▶ More about the metadata in modeler and the consequences on R integration

## 5.1 R code

This section provides example R code.

### 5.1.1 ibmspssc70 library

Lets take a more detailed look at what actually happens with the R code. The IBM delivered R package `ibmspssc70` is installed in the library folder of your R installation. This library contains several functions that handle the data traffic between IBM SPSS Modeler and R.

Running any R node in SPSS does not only run the code you write. It also runs some extra code 'behind the scenes.' You can see this code in the Console output window of the R node. Looking for example at this tab for an R nugget, you see that your code is similar to Example 5-1.

*Example 5-1 R nugget with IBM R package code*

---

```
modelerModel <- ibmspsscoutput.GetModel()
while(ibmspsscdata.HasMoreData()){
  modelerDataModel <- ibmspsscdata.GetDataModel()
  modelerData <- ibmspsscdata.GetData(rowCount=1000, missing=NA, rDate="None",
  logicalFields=FALSE)
  @Your R code
  ibmspsscdataModel.SetDataModel(modelerDataModel)
  ibmspsscdata.SetData(modelerData)
}
```

---

All the functions starting with `ibmspssc` are functions within the R package `ibmspssc70` library. It is this part of the code that is responsible for the transfer of the data and the metadata to R. Also, you see a while loop, indicating that the data goes to R in batches of 1000. The other options are the values of the data read options within the node.

The last lines of the code (Example 5-1) prepares the data to be send back to SPSS Modeler. Notice the closing brace, which is the end of the while loop.

Because this library is loaded with every interaction between SPSS and R, you are free to use these functions within your code as well. If you would like to avoid these runs in batches of 1K and you do not have SPSS Modeler version 17.1 available, you can start your R code with another loop to first continue filling the `modelerData` and then start your actual code. Example 5-2 shows what your R node should start with.

*Example 5-2 Starter code*

---

```
while(ibmspsscdata.HasMoreData())
{
  modelerData
  <-rbind(modelerData,ibmspsscdata.GetData(rowCount=100000,missing=NA,rDate="None",
  logicalFields=FALSE))
}
@Some more code using modelerData which is now the complete dataset
```

---

## 5.1.2 Some useful parts of R code

The following sections provide useful R code snippets.

### Make sure a package is installed

Whenever you have created a R node using packages, be sure that anyone using this node has this package installed on their machine, while trying to minimize user interference. Example 5-3 shows verifying the packages are present.

*Example 5-3 Verify package is present*

---

```
packages <- function(x) {
  x <- as.character(match.call()[[2]])
  if (!require(x, character.only=TRUE)) {
    install.packages(pkgs=x, repos="http://cran.r-project.org")
    require(x, character.only=TRUE)
  }
}
packages(rpart)
```

---

The code verifies that the library is installed on the system, if not it will silently install it from the CRAN mirror (you can change this to another mirror or a local repository if needed). The installation only happens the first time the node is used.

### Create the metadata corresponding to the R data frame

Sometimes your data is transformed compared to the original data, such that it is difficult to build your metadata starting from the original data. In this situation, you might want to change the metadata to link to the data in R independent of the original. This approach is particularly useful if you want to use R as a sort node. Example 5-4 shows a function that asks for a data frame and creates `modelerData` and `modelerDataModel` accordingly.

*Example 5-4 Create modelerData and modelerDataModel*

---

```
sendToModeler <- function (dataFrame) {
  if(is.null(dim(dataFrame))){
    stop("Invalid data received: not a data.frame")}
  if (dim(dataFrame)[1]<=0) {
    print("Warning : modelerData has no line, all fieldStorage fields set to strings")
    getStorage <- function(x){return("string")}
  } else {
    getStorage <- function(x) {
      x <- unlist(x)
      res <- NULL
      #if x is a factor, typeof will return 'integer' so we handle this case first
      if(is.factor(x)) {
        res <- "string"
      } else {
        res <- switch(typeof(x), integer="integer", double = "real", "string")
      }
    }
    return (res)
  }
}
col = vector("list", dim(dataFrame)[2])
for (i in 1:dim(dataFrame)[2]) {
  col[[i]] <- c(fieldName= names(dataFrame[i]) ,fieldLabel = "", fieldStorage=
```

```

getStorage(dataFrame[i]), fieldMeasure = "", fieldFormat = "", fieldRole = "")
}
mdm<-do.call(cbind,col)
modelerDataModel<-data.frame(mdm)
modelerData <- dataFrame
}
sendToModeler(iris)

```

---

If you use this code, you should make sure you only use this on data frames that are not very dependent on the content of the original modelerData. If results are not as expected, you might find an answer to this situation in 5.5, “More about the metadata in modeler and the consequences on R integration” on page 37.

### Looping through several variables

Looping through several variables is common functionality that is relatively easy in R. However, with the Custom Dialog Builder, it might be more difficult because the string `%%INPUTS%%` is exactly replaced by the string `age + income + gender` or something similar (you can change the “+” sign to commas or spaces depending on the separator chosen).

Now the problem is that to loop in R, you need to transform this string into the R vector `c(“age”, “income”, “gender”)`, as shown in Example 5-5.

#### *Example 5-5 Function to remove trailing spaces*

---

```

#Create a function to remove trailing spaces
trim <- function(x) gsub("^\\s+|\\s+$", "", x)
#Create the vector of using the strsplit functions
InputsAsVector <- trim(strsplit("%%INPUTS%%", "+")[[1]])
for (input in inputAsVector){
@Some more code to run for every
eld de
ned
}

```

---

One important remark is that this method never works if you have variable names containing trailing spaces or + symbols. This is because this code recognizes every + as the symbol to separate the variables, and removes the trailing spaces. It is difficult in this way to distinguish between a + coming from a variable name or a + being a separator.

### Use predefined roles

Predefined roles is an option SPSS Modeler users are used to that you might want to extend to R usage. The idea is to distinguish between inputs and targets (and others) merely in the Type-node. After this type-node is defined, all the modeling nodes by default use these settings and variables.

You might want to use some R code to distinguish between inputs and targets. The code example in Example 5-6 shows looking for the ag targets and for all the input variables.

#### *Example 5-6 Look for ag targets and input variables*

---

```

TARGET <- modelerDataModel[1,(modelerDataModel[6,] == "target" &
modelerDataModel[4,] == "
flag")]
INPUTS <- as.vector(t(modelerDataModel[1,(modelerDataModel[6,] == "input"))
@Some more code

```

---

## Removing columns

Sometimes you just want to remove a column in both `modelerData` and `modelerDataModel`. You should make sure to delete the appropriate column, as the link between data and metadata is merely the order. You can use Example 5-7 R code to remove the column `tenure`.

*Example 5-7 Code to remove the column `tenure`*

---

```
#define the remove function
removeColumn <- function(name){
modelerDataModel[,modelerDataModel[1,]==name]<<-NULL
modelerData[,colnames(modelerData)== name] <<- NULL
}
#apply the function to the tenure variable
removeColumn("tenure")
@Some more code
```

---

## 5.2 Custom Dialog Builder tips

This section discusses tips on using Customer Dialog Builder.

### 5.2.1 How to save and share a custom dialog

The specifications of a custom dialog can be saved to an external file, with the extension `.cfd` or `.mpe` for SPSS Modeler Version 18. This file can be saved and reopened through the general save buttons on the custom dialog tool bar.

After you deploy the dialog to your palette, this dialog is saved as a local file, under a slightly different extension `.cfe`. You can find this `cfe` file in the path `c:\ProgramData\IBM\SPSS\Modeler\XX\CDB` (replace `XX` with the version of your IBM SPSS Modeler installation).

To share this node with others, this node needs to be copied within the same folder on the other SPSS Modeler instance.

### 5.2.2 Link to dialog and script

If you have an identifier called `TARGET` and you fill in a variable `churn` in these dialog, all the references of `%%TARGET%%` are replaced in your code by `churn`. If you have multiple variables selected (say `age` and `income` in the identifier `INPUTS`) and you select “+” as the separator, then within the code `%%INPUTS%%` is replaced by the verbatim `age + income`.

Although this is valid in default cases, it is not entirely true. There is still another level. It is dependent on the tool property in called “R script”.

Within this line, the value `%%ThisValue%%` is replaced by whatever you fill in the dialog. And it is the value of this R script property that verbatim replaces your identifier in the main R code. Because often the value of R script is just `%%ThisValue%%`, there is no need for any change.

This R-script property starts to be useful when you work with radio buttons, in which case you might like to run different codes for each button. In general there are two ways to do this activity:

- ▶ Create your R script using if statements, as shown in Example 5-8.

*Example 5-8 R script with if statements*

```
If '%%choice%' == 'A' then ....
else if '%%choice%' == 'B' then ....
```

- ▶ Write the full code that has to be run when the bullet radio button is selected.

Imagine the following scenario. The user selects some variables and some computations are done on each of them. Depending on the outcome of the computations and defined cutoffs, you can choose between any of these actions:

- ▶ Remove the columns
- ▶ Keep the columns as data but automatically set the role to “None”
- ▶ Do nothing and just let the data flow back to the modeler without changes

There are different levels that you can work with. You can first add a radio button group in the dialog and call it WHATTODO. See Figure 5-1.

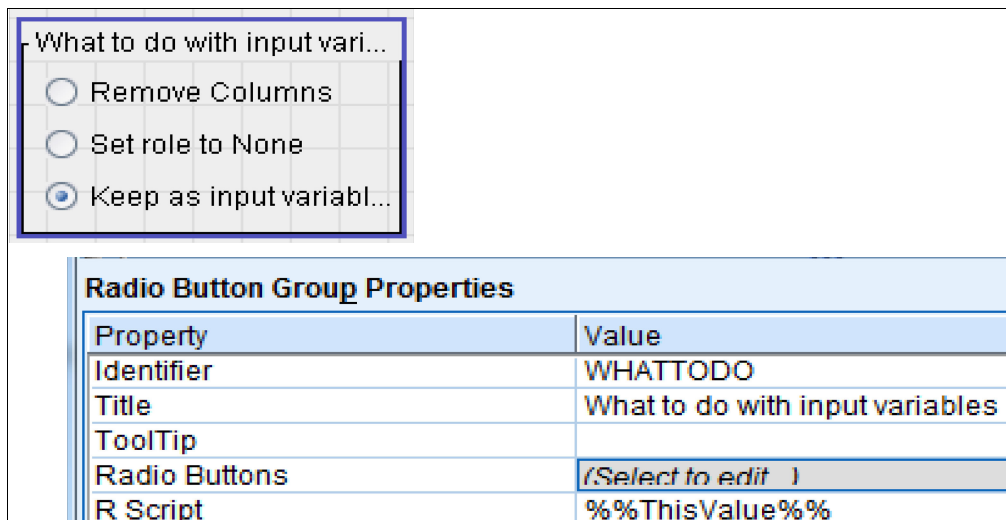


Figure 5-1 WHATTODO radio button

The value of the R script variable is not changed. Click through to the radio button itself, you want the results to look similar to Figure 5-2.

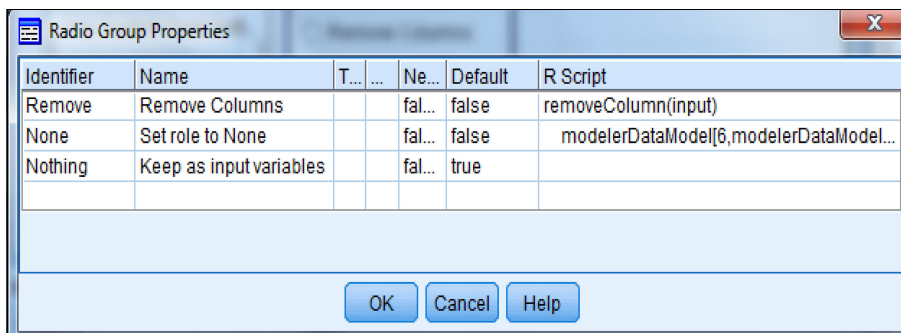


Figure 5-2 Radio Group Properties



Figure 5-2 on page 34 shows the three options, each with its corresponding R code:

- ▶ `removeColumn (input)` referring to a function written to remove columns
- ▶ `modelerDataModel[6,modelerDataModel[1,]==input] <- "none"`, to change the role of that field to "None"
- ▶ Basically an empty string

With these settings, your code `%%WHATTODO%%` is replaced by the corresponding R code depending on which radio button the user selects. The R script behind the dialog is shown in Example 5-9.

*Example 5-9 R script for %%WHATTODO%%*

---

```
trim <- function (x) gsub("^\\s+|\\s+$", "", x)
inputAsVector <- trim(strsplit("%%INPUTS%%",",")[[1]])
for (input in inputAsVector){
%%WHATTODO%%
}
@Some more code
```

---

This approach simplifies the possibility of running different R codes depending on the end users choice.

## 5.3 What about SQL Pushback? Hadoop Pushback?

SQL pushback supports R nodes for IBM PureData® for Analytics, SAP Hana, and Oracle by utilizing their R support. Databases need to have the appropriate vendor provided R extensions installed. Depending upon the vendor, a subset of libraries or scrips are supported.

Let's discuss R usage on PureData for Analytics which has R nuggets available for SQL pushback. The reason for R nuggets is that data is divided onto the several Synergistic Processor Units (SPUs) of PureData for Analytics. R code is independently being run on the different SPUs and never on the entire table.

Example 5-10 shows the code that cannot be pushed back to the PureData for Analytics environment.

*Example 5-10 Code not pushed to PureData for Analytics*

---

```
@This code cannot be pushed back to PureData for Analytics, as it is a model
building node
#Create the model and save it in modelerModel
modelerModel <- lm(tenure ~ ., data= modelerData)
```

---

```
@This code can be pushed back to PDA, as it is a model scoring node
#Make sure they are considered as factors, as PDA will by default only have
numerics
modelerData$marital <- as.factor(modelerData$marital)
modelerData$ed <- as.factor(modelerData$ed)
modelerData$region <- as.factor(modelerData$region)
modelerData$churn <- as.factor(modelerData$churn)
#Use the model to make a prediction, and add it to the existing data.
```

```
pred <- predict(modelerModel, modelerData)
modelerData <- cbind(modelerData,pred)
#Take care of the metadata!
newVar <-c(fieldName="TenureSQLScore", fieldLabel="", fieldStorage="real",
fieldMeasure="",
fieldFormat="", fieldRole="")
modelerDataModel <- cbind(modelerDataModel,newVar)
```

---

Another item to consider is that the `modelerModel` is always a local object. Within the R code that is pushed back, this object is transferred to PureData for Analytics behind the scenes. This transfer is not always a problem. However, the size of `modelerModel` can often be quite big. For example, a linear model for the telco data set is approximately 370 KB. As a comparison, the size where that model is run is approximately 50 KB. RandomForests data set is 50 KB. RandomForests or any other type of models can be huge. The transfer to PureData for Analytics is something that needs to be considered when using this approach.

Because everything runs on the different SPUs R and all the necessary libraries have to be installed on both the host and every SPU on the PureData for Analytics environment. For the R libraries, you do not do that in the same way as loading R libraries on a local system, because PureData for Analytics is not connected to the internet. How to solve this situation is more a question of the R-PureData for Analytics link, rather than SPSS. Example 5-11 shows a small R script to load a package on the PureData for Analytics from the local R instance.

*Example 5-11 Local R script*

---

```
@Note this will be a local R script.
#Load PDA local R libraries
library(nzr)
library(nza)
#Connect to the appropriate DSN
nzConnectDSN('PDA-DSN')
#install the library on PDA
nzInstallPackages("http://cran.r-project.org/src/contrib/rpart_4.1-10.tar.gz")
```

---

This code installs the package `rpart` onto PureData for Analytics (assuming you are logged on with appropriate credentials). You can choose to run this code in native R or within SPSS Modeler in a R output node. Both options work. After it is installed on the PureData for Analytics, you can use the libraries as normal.

## 5.4 What about real-time scoring? and IBM SPSS Modeler Solution Publisher?

Both the real-time scoring and Solution Publisher are supported. You should make sure the R and the R extensions are installed with these considerations:

- ▶ For Solution Publisher, you should just install the R extension in the `/ext/bin` directory of your Solution publisher. Install R on the machine where your solution is published.
- ▶ For real-time scoring, you should ensure that the R extensions are installed in the `/components/modeler/ext/bin` directory of both your server and your scoring server. Functionally, only R transform and R nuggets are relevant for this part. Install R on the machine where your application server (for example IBM WebSphere® Application Server) is installed.

## 5.5 More about the metadata in modeler and the consequences on R integration

Metadata in SPSS Modeler is something particular and is very important for the way SPSS Modeler works.

It is important that SPSS Modeler knows at all times all the metadata of the data at every node within the stream. You might have already noticed that when you add a new field (with a derive node), all the type nodes downstream immediately take into account this extra field. In order to do this, behind the scenes, SPSS Modeler lets some small dummy data flow around. This data only has to verify the metadata in near real-time.

If you want to know what this dummy data looks like, you can add an R transform node just after the source node and use the following syntax (Example 5-12) to write out the data that is passed by SPSS through R and back. This code does not do anything with the data. It just writes it back into a file.

*Example 5-12 Syntax to write data passed by SPSS to disk*

---

```
path <- "C:/test.txt"
sink(path)
writeLines(as.character(Sys.time()))
writeLines("Data:")
print(modelerData)
```

---

After these lines are added to the R transform node and continue creating the stream, you see that this file is already populated without running anything. You can see the data that is passed through the node only contains five lines of data with 1-2-3-4-5 and "a-b-c-d-e" (some enclosed by quotes, for the string variables) depending on the metadata. These five lines of dummy data modeler are going around the stream every time SPSS Modeler needs/wants to check the metadata.

You can even append this script with `print(modelerModel)` to see the value of `modelerModel` is not yet assigned. `modelerModel` cannot play any role in the assigning of `modelerDataModel`.

This approach has some consequences with the R integration. It explains a lot of the "strange" behavior in your projects, where the reason is not always obvious. Let's say you have a multinomial logistic model with  $n$  different categories. You want to have a column back for all the categories. A naive approach is shown in Example 5-13.

*Example 5-13 One example being the naive approach.*

---

```
library("rpart")
modelerModel <- rpart(custcat ~ tenure+age+income, data = modelerData)
print(summary(modelerModel))
```

---

```
library("rpart")
probs <- predict(modelerModel,modelerData,type="prob")
modelerData <- cbind(modelerData,probs)
for (x in colnames(probs) ){
modelerDataModel<-cbind(modelerDataModel, c(fieldName=paste("$P-",x,sep = ""),
fieldLabel="")
```

```
,fieldStorage="real", fieldMeasure="", fieldFormat="", fieldRole=""))
}
```

---

Running this approach in R natively produces the correct `modelerData` and `modelerDataModel` objects. However, this approach does not work in SPSS Modeler because `modelerModel` is not assigned when modeler assesses the metadata. This code runs with an empty `modelerModel` and the five dummy records. As a result probs are empty so nothing within the for loop runs.

A work around is to derive the number of columns not from the `modelerModel`, but in another way. Example 5-14 shows this different approach.

*Example 5-14 Work around approach*

---

```
library("rpart")
modelerModel <- rpart(custcat ~ tenure+age+income, data = modelerData)
print (summary(modelerModel))

library(rpart)
probs <- predict(modelerModel,modelerData,type="prob")
modelerData <- cbind(modelerData,probs)
for (x in c(1,2,3,4)){
modelerDataModel<-cbind(modelerDataModel, c(fieldName=paste("$P-",x,sep = ""),
fieldLabel=""
,fieldStorage="real", fieldMeasure="", fieldFormat="", fieldRole=""))
}
```

---

However, with this approach there is path a problem because it needs to hardcode the values of the for loop. The reason behind this approach with SPSS Modeler is that this metadata should be available in near real-time. However, `modelerData` and `modelerModel` are objects that can be very big and lead to a large delay in obtaining this metadata.





REDP-5388-00

ISBN 0738455601

Printed in U.S.A.

Get connected

