



THE
POWER
TO KNOW.

SAS® Studio 3.1:

Developer's Guide

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2014. *SAS® Studio 3.1: Developer's Guide*. Cary, NC: SAS Institute Inc.

SAS® Studio 3.1: Developer's Guide

Copyright © 2014, SAS Institute Inc., Cary, NC, USA

All rights reserved. Produced in the United States of America.

For a hard-copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a) and DFAR 227.7202-4 and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

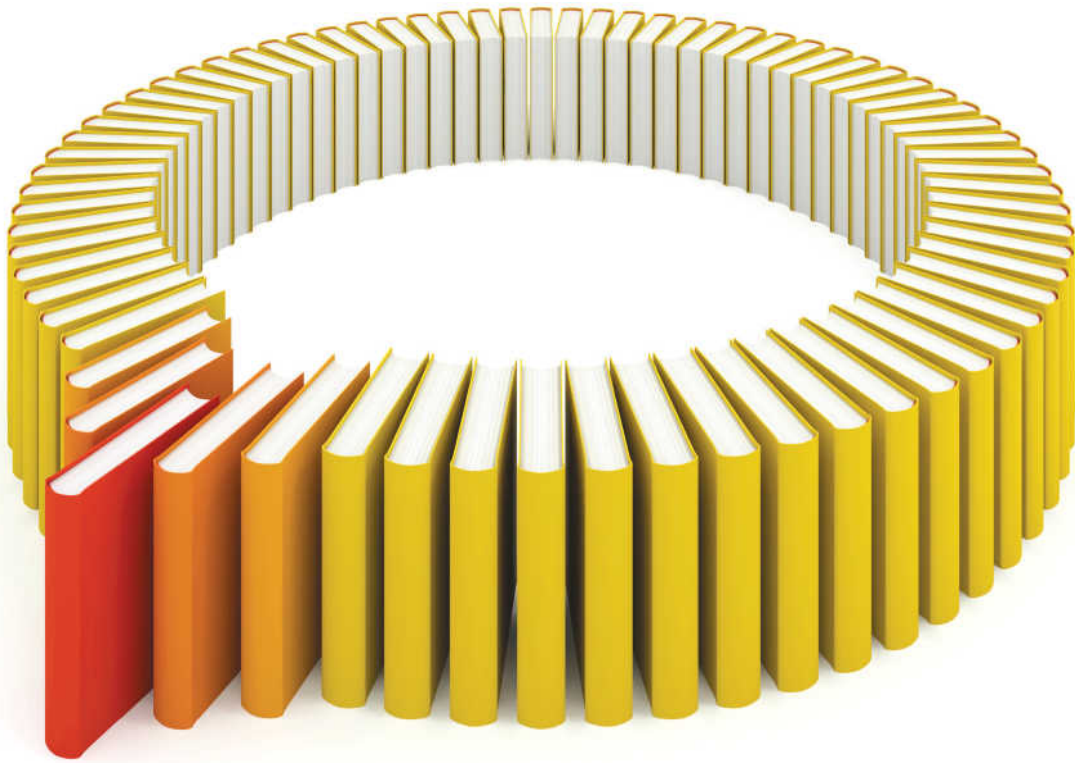
SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513-2414.

March 2014

SAS provides a complete selection of books and electronic products to help customers use SAS® software to its fullest potential. For more information about our offerings, visit **support.sas.com/bookstore** or call 1-800-727-3228.

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.



Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

Contents

<i>Using This Book</i>	<i>vii</i>
<i>Accessibility</i>	<i>ix</i>
<i>Recommended Reading</i>	<i>xi</i>
 Chapter 1 • Introduction to the Common Task Model	1
About the SAS Studio Tasks	1
Create a New Task	3
Validation Steps for the Task	5
Testing a Task	5
Sharing Tasks	6
 Chapter 2 • Working with the Registration Element	7
About the Registration Element	7
Example: The Registration Element from the Sample Task Definition	8
 Chapter 3 • Working with the Metadata Element	9
About the Metadata Element	9
Working with the DataSources Element	9
Working with the Options Element	13
 Chapter 4 • Working with the UI Element	27
About the UI Element	27
Example: UI Element for the Sample Task Definition	29
 Chapter 5 • Working with the Dependencies Element	33
About the Dependencies Element	33
Notes on Dependencies	35
Example 1: Selecting a Check Box Enables Text Boxes	37
Example 2: Selecting a Check Box Enables a Combination Box ...	38
Example 3: Selecting a Radio Button Enables a Number Stepper Control	40

Example 4: The Selected Value for a Combination Box Enables the Text Box	41
Example 5: Selecting a Check Box Enables Multiple Types of Options	43
Example 6: Compound Condition Using AND and OR Logic	44
Chapter 6 • Working with the Requirements Element	47
About the Requirements Element	47
Example: Using a Requirements Element for Roles	48
Chapter 7 • Understanding the Code Template	49
About the Code Template	49
Predefined Variables	50
Predefined SAS Macros	50
Data Source	51
Roles	52
Options	52
Chapter 8 • Example: Task Definition for List Data Task	59
Open the List Data Task	59
View the XML Code for the List Data Task	60
Understanding the XML Code for the List Data Task	61

Using This Book

Audience

The *SAS Studio: Developer's Guide* is intended for programmers who need to develop custom tasks for their site. This document describes the common task model for SAS Studio and explains the syntax used in this task model.

Prerequisites

For task development, it is recommended that you use the latest version of Google Chrome because of its debugging tools.

Accessibility

For information about the accessibility of this product, see [Accessibility Features of SAS Studio 3.1](#) at support.sas.com.

Recommended Reading

- *SAS Studio: Administrator's Guide*
- *Getting Started with Programming in SAS Studio*
- *SAS Studio: User's Guide*

For a complete list of SAS books, go to support.sas.com/bookstore. If you have questions about which titles you need, please contact a SAS Book Sales Representative:

SAS Books

SAS Campus Drive

Cary, NC 27513-2414

Phone: 1-800-727-3228

Fax: 1-919-677-8166

E-mail: sasbook@sas.com

Web address: support.sas.com/bookstore

Introduction to the Common Task Model

<i>About the SAS Studio Tasks</i>	1
<i>Create a New Task</i>	3
<i>Validation Steps for the Task</i>	5
<i>Testing a Task</i>	5
<i>Sharing Tasks</i>	6
About CTM and CTK Files	6
Accessing a Task Created by Another User	6
Sharing a Task That You Created	6

About the SAS Studio Tasks

SAS Studio is shipped with several predefined tasks, which are point-and-click user interfaces that guide the user through an analytical process. For example, tasks enable users to create a bar chart, run a correlation analysis, or rank data. When a user selects a task option, SAS code is generated and run on the SAS server. Any output (such as graphical results or data) is displayed in SAS Studio.

Because of the flexibility of the task framework, you can create tasks for your site. In SAS Studio, all tasks use the same common task model and the Velocity Template Language. No Java programming or ActionScript programming is required to build a task.

The common task model (CTM) defines how the task appears to the SAS Studio user and how to run the task. A task is defined by its input data and the options that are available to the user. (Some tasks might not require an input data source.) In addition, the task has metadata so that it is recognized by SAS Studio.

In SAS Studio, a task is defined by the `Task` element, which has these children:

Registration

The `Registration` element identifies the type of task. In this element, you define the task name, icon, and unique identifier.

Metadata

The `Metadata` element specifies whether an input data source is required to run the task, any role assignments, and the options in the task.

- The UI engine reads the `Roles` element to determine what types of variables are required by the task. Here is the information that you would specify in this element:
 - type of variable that the user can assign to this role (for example, numeric or character)
 - the minimum or maximum number of variables that you can assign to a role
 - the label or description of the role that appears in the user interface
- The UI engine reads the `Options` element to determine how to display the options in the user interface.

UI

The `UI` element describes how to present the user interface to the user. A top-down layout is supported.

Dependencies

The `Dependencies` element describes any dependencies that options might have on one another. For example, selecting a check box could enable a text box.

Requirements


The `Requirements` element specifies what conditions must be met in order for code to be generated.

Code Template

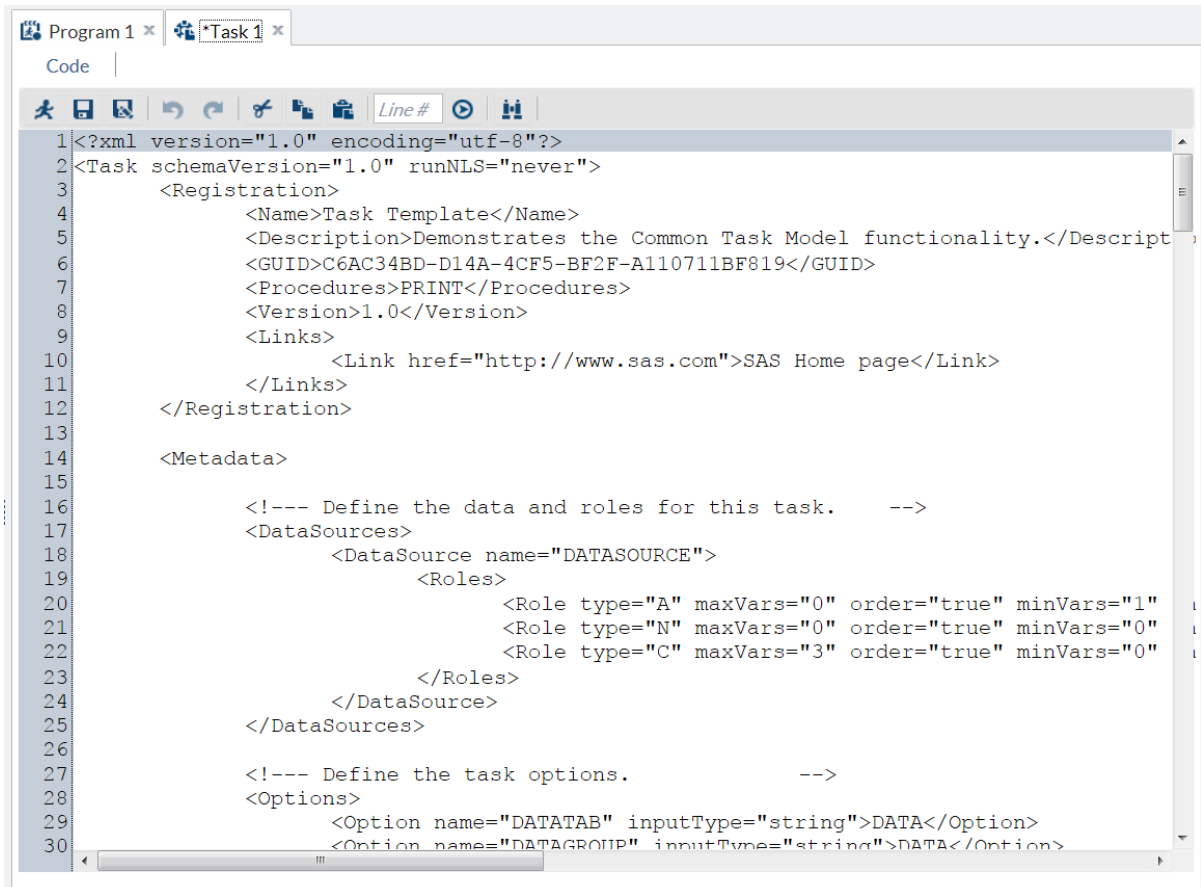
The `Code Template` element determines the output of the task. For most tasks, the output is SAS code. However, you can create any type of text output.

Create a New Task

To create a new task:

- 1 In the navigation pane, open the **Tasks** section.
- 2 Click .


The sample task definition that is shipped with SAS Studio appears.

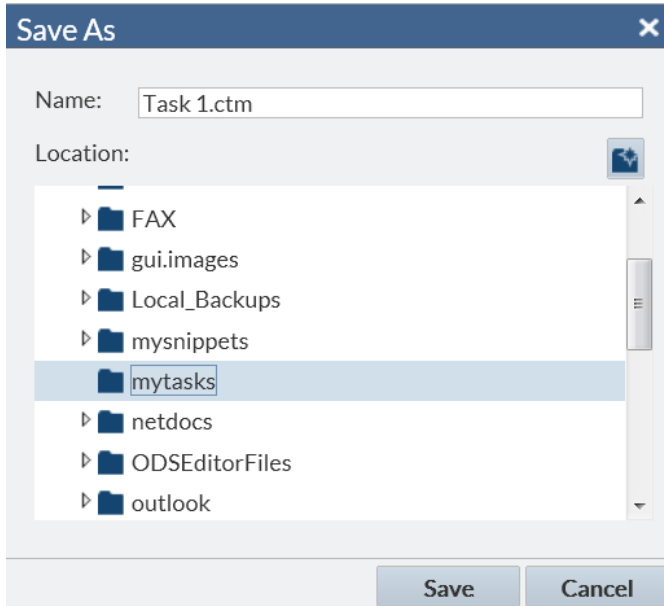


```

1<?xml version="1.0" encoding="utf-8"?>
2<Task schemaVersion="1.0" runNLS="never">
3  <Registration>
4    <Name>Task Template</Name>
5    <Description>Demonstrates the Common Task Model functionality.</Description>
6    <GUID>C6AC34BD-D14A-4CF5-BF2F-A110711BF819</GUID>
7    <Procedures>PRINT</Procedures>
8    <Version>1.0</Version>
9    <Links>
10      <Link href="http://www.sas.com">SAS Home page</Link>
11    </Links>
12  </Registration>
13
14  <Metadata>
15
16    <!-- Define the data and roles for this task. -->
17    <DataSources>
18      <DataSource name="DATASOURCE">
19        <Roles>
20          <Role type="A" maxVars="0" order="true" minVars="1" />
21          <Role type="N" maxVars="0" order="true" minVars="0" />
22          <Role type="C" maxVars="3" order="true" minVars="0" />
23        </Roles>
24      </DataSource>
25    </DataSources>
26
27    <!-- Define the task options. -->
28    <Options>
29      <Option name="DATATAB" inputType="string">DATA</Option>
30      <Option name="DATAGROUP" inputType="string">DATA</Option>

```


- 3 Edit the sample task definition to create your task. For help with the Velocity Template Language, see *Apache Velocity User's Guide*.
- 4 To save the task, click .
- 5 Enter a unique name for the task. The task is saved with the CTM file extension in your file system.



Validation Steps for the Task

When you run a task, SAS Studio validates the code by determining whether the XML is well formed, whether the Velocity template has any syntax errors, and whether there are any logical XML errors.

Testing a Task

To test your task, click . (Alternatively, you can press F3.) The user interface for the task appears to the right of your work area.

If you are using Google Chrome, press F12 to view any errors.


Sharing Tasks

About CTM and CTK Files

After creating a task, you might want to share it with other users at your site. Tasks are saved as CTM files. You might also want to share CTK files, which are CTM files with some of the roles and options preselected. For more information about how to create a CTK file, see the “Save a Task and Its Option Settings” topic in *SAS Studio: User's Guide*.


Accessing a Task Created by Another User

To access this task from SAS Studio:

- 1 Save the CTM or CTK file to your local computer. (This file could have been sent to you by e-mail.)
- 2 In SAS Studio, open the **Folders** section and click . The Upload Files window appears.
- 3 Specify where you want to upload the files and click **Choose Files** to select a file.
- 4 Click **Upload**.

Sharing a Task That You Created

If you save the CTM or CTK file to a shared network location, other users can create a folder shortcut to access the task from SAS Studio. The advantage to this approach is that you have only one copy of the CTM file.

To create a new folder shortcut, click  in the Folders section and select **Folder Shortcut**. Enter the shortcut name and full path and click **Save**. The new shortcut is added to the list of folder shortcuts.

2

Working with the Registration Element

About the Registration Element 7

Example: The Registration Element from the Sample Task Definition 8

About the Registration Element

The `Registration` element represents a collection of metadata for the task. The hosting application requires this element in order to know the type of task.

Here are the child elements for the `Registration` element:

Element Name	Description
Name	The name of the task. This name is used throughout the application to represent the task.
Description	A description of the task. This text could appear in the task properties or in tooltips for the task.
GUID	A unique identifier for the task.
Procedures	A list of SAS procedures that are used by this task.
Version	A simple integer value that represents the version of the task.
Links	A list of hyperlinks to help or resources related to this task.

Example: The Registration Element from the Sample Task Definition

To create a new task, you can use the sample task definition that is shipped with SAS Studio.

Here is the `Registration` element from the sample task definition:

```
<Registration>
  <Name>Task Template</Name>
  <Description>Demonstrates the Common Task Model functionality.</Description>
  <GUID>C6AC34BD-D14A-4CF5-BF2F-A110711BF819</GUID>
  <Procedures>PRINT</Procedures>
  <Version>1.0</Version>
  <Links>
    <Link href="http://www.sas.com">SAS Home page</Link>
  </Links>
</Registration>
```

3

Working with the Metadata Element

<i>About the Metadata Element</i>	9
<i>Working with the DataSources Element</i>	9
About the DataSources Element	9
Working with the Roles Element	10
<i>Working with the Options Element</i>	13
About the Options Element	13
Supported Input Types	14

About the Metadata Element

The `Metadata` element comprises two parts: the `DataSources` element and the `Options` element.

Working with the DataSources Element

About the DataSources Element

The `DataSources` and `DataSource` elements create a simple grouping of the data that is required for the task. If these elements are not specified, then no input data is needed to run the task.




The `DataSource` element is the only child of the `DataSources` element, and the `DataSources` element can have only one `DataSource` child. The `DataSource` element specifies the information about the data set for the task. The only child for the `DataSource` element is the `Roles` element.

Working with the Roles Element

About the Roles Element

The `Roles` element identifies the variables that must be assigned in order to run the task. This element is a way to group the individual role assignments that are needed for a task.

The `Role` tag, which is the only child of the `Roles` element, describes one type of role assignment for the task.

Attribute	Description
name	Specifies the name assigned to this role.
type	<p>Specifies the type of column that can be assigned to this role.</p> <p>Here are the valid values:</p> <p>A All column types are allowed. In the user interface, all columns are identified by the  icon.</p> <p>N Only numeric columns can be assigned to this role. In the user interface, numeric columns are identified by the  icon.</p> <p>C Only character columns can be assigned to this role. In the user interface, character columns are identified by the  icon.</p>
minVars	Specifies the minimum number of columns that must be assigned to this role. If <code>minVars=0</code> , the role is optional. If <code>minVars=1</code> , a column is required to run this task, and a red asterisk appears next to the label in the user interface.

Attribute	Description
maxVars	Specifies the maximum number of columns that can be assigned to this role. If maxVars=0, users can assign an unlimited number of columns to this role.
exclude	Specifies the list of roles that are mutually exclusive to this role. If a column is assigned to a role in this list, the column does not appear in the list of available columns for this role.
order	Specifies that the user can order the columns that are assigned to this role. Valid values are true and false. If order=true, the user can use the up and down arrows in the user interface to modify the order.

Example: DataSources and Roles Elements from the Sample Task Definition

Here is an example of the DataSources and Roles elements:

```
<DataSources>
  <DataSource name="PRIMARYDATA">
    <Roles>
      <Role type="A" maxVars="0" order="true" minVars="1"
        name="VAR"> Required variable label</Role>
      <Role type="N" maxVars="0" order="true" minVars="0"
        name="OPTNVAR" exclude="VAR">Numeric variable label</Role>
      <Role type="C" maxVars="3" order="true"
        minVars="0" name="OPTCVAR">Character variable label</Role>
    </Roles>
  </DataSource>
</DataSources>
```

When you run this code, you get the Data and Roles sections in this example:

▲ DATA

SASHELP.CLASS ▼



▲ ROLES

*Required variable
label:



 Column

Numeric variable
label:



 Column

Character variable
label: (3 items)



 Column

A red asterisk appears for the **Required variable label** role because you must assign a column to this role. In the code, this requirement is indicated by `minVars=1`.

Working with the Options Element

About the Options Element

The `Options` element identifies the options that are required in order to run the task. The `Option` tag, which is the only child of the `Options` element, describes the assigned option.

Attribute	Description
<code>name</code>	Specifies the name assigned to this option.
<code>defaultValue</code>	Specifies the initial value for the option.
<code>inputType</code>	<p>Specifies the input control for this option. Here are the valid values:</p> <ul style="list-style-type: none">■ <code>checkbox</code>■ <code>color</code>■ <code>combobox</code>■ <code>inputtext</code>■ <code>modelbuilder</code>■ <code>numstepper</code>■ <code>radio</code>■ <code>slider</code>■ <code>string</code>■ <code>validationtext</code> <p>For more information, see “Supported Input Types” on page 14.</p>

Attribute	Description
indent	Specifies the indentation for this option in the task interface. Here are the valid values: <ul style="list-style-type: none">■ 1 – minimal indentation (about 17px)■ 2 – average indentation (about 34px)■ 3 – maximum indentation (about 51px)

Supported Input Types

checkbox

This input type does not have additional attributes. The valid values for `checkbox` are 0 (unchecked) and 1 (checked).

The code in the sample task definition creates two check boxes.

```
<Option name="GROUP1" inputType="string">CHECK BOXES</Option>
<Option name="chkCheck defaultValue=1" inputType="checkbox">
  Enable another field</Option>
<Option name="chkEXAMPLE" defaultValue="0" inputType="checkbox">
  Check box</Option>
```

When you run the code, here is the user interface that is created:

▲ CHECK BOXES

☒ Enable another field

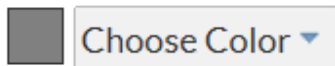
☐ Check box

color

This input type does not have additional attributes. Here is an example from the sample task definition:

```
<Option name="colorEXAMPLE defaultValue="gray"
  inputType="color">Choose Color</Option>
```

When you run the code, here is the user interface that is created:



combobox

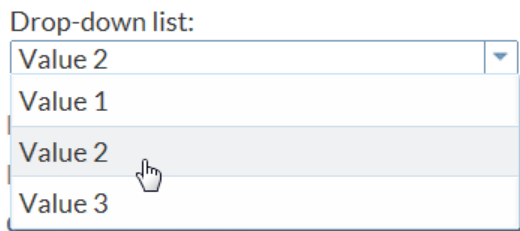
This input type has one attribute.

Attribute	Description
width	Specifies the width of the control. This value can be in percent (%), em, or px. By default, SAS Studio sizes the control based on the available width and content.

The code in the sample task definition creates a combination box called **Drop-down list**. This list contains three options: **Value 1**, **Value 2**, and **Value 3**.

```
<Option name="comboEXAMPLE" defaultValue="value2" inputType="combobox">
  Drop-down list:</Option>
<Option name="value1" inputType="string">Value 1</Option>
<Option name="value2" inputType="string">Value 2</Option>
<Option name="value3" inputType="string">Value 3</Option>
```

When you run the code, here is the resulting user interface:



inputtext

This input type has these attributes:

Attribute	Description
required	Specifies whether any input text is required. Valid values are true and false. The default is false.

Attribute	Description
missingMessage	Specifies the tooltip text that appears when the text box is empty but input text is required. No message is displayed by default.
promptMessage	Specifies the tooltip text that appears when the text box is empty and the user has selected the text box.
width	Specifies the width of the control. This value can be in percent (%), em, or px. By default, SAS Studio sizes the control based on the available width and content.

The code in the sample task definition creates a text box called **Indented field with default value**. The default value for this option is output.xls. If the user removes this text, the message “Enter a filename” appears because a valid filename is required.

```
<Option name="txtEXAMPLE" defaultValue="output.xls" inputType="inputtext"
  width="100%"
  indent="1"
  required="true"
  promptMessage="Enter a filename"
  missingMessage="Missing filename">Indented field with default value:</Option>
```

When you run the code, here is the user interface that is created:

▲ TEXT INPUT FIELD

☒ Enable text input field

*Indented field with default value:

output.xls

modelbuilder

A *model* is an equation that consists of a dependent or response variable and a list of effects. The user creates the list of effects from variables and combinations of variables.

Here are examples of effects:

main effect

For variables Gender and Height, the main effects are Gender and Height.

interaction effect

For variables Gender and Height, the interaction is Gender * Height. You can have two-way, three-way, ...*n*-way interactions.

The order of the variables in the interaction is not important. For example, Gender * Height is the same as Height * Gender.

polynomial effect

You can create polynomial effects with continuous variables. For the continuous variable X, the quadratic polynomial effect is X*X. You can have second-order, third-order, ...*n*th-order polynomial effects.

The `modelbuilder` input type has these attributes:

Attribute	Description
<code>required</code>	Specifies whether any input text is required. Valid values are <code>true</code> and <code>false</code> . The default is <code>false</code> .
<code>roleContinuous</code>	Specifies the role that contains the continuous variables. The default value is <code>null</code> .
<code>roleClassification</code>	Specifies the role that contains the classification variables. The default value is <code>null</code> .
<code>excludeTools</code>	Specifies the effect and model buttons to exclude from the user interface. Valid values are ADD, CROSS, NEST, TWOFACT, THREEFACT, FULLFACT, NFACTORIAL, POLYEFFECT, POLYMODEL and NFACTPOLY. Separate multiple values with spaces or commas.
<code>width</code>	Specifies the width of the control. The width value can be specified in percent, em, or px. By default, the control is automatically sized based on the available width and content.

Note: At least one of the role attributes (`roleContinuous` or `roleClassification`) is required. If both attributes are set to `null`, no variables are available to create the model.

Here is some example code for the model builder from the Generalized Linear Model task:

```
<Option excludeTools="THREEFACT,NFACTPOLY" inputType="modelbuilder"
  name="modelbuilder" roleClassification="classVariables"
  roleContinuous="continuousVariables"
  width="100%">Model</Option>
```

Model

Variables

Model effects

EFFECTS

Add
+

Cross
X

N: 2

Polynomial, Degree=N

MODELS

Two-way
Factorial

Full
Factorial

N: 2

N-way
Factorial

Polynomial, Order=N

After selecting an input data source and identifying the columns that contain the continuous or classification variables, you can start building your model. This example uses the Sashelp.Cars data set as the input data source. MSRP, EngineSize, Invoice, and Cylinders are the continuous variables.

Model




Variables

MSRP

EngineSize

Invoice

Cylinders

Model effects   

MSRP*MSRP

EngineSize

EngineSize*EngineSize

EFFECTS

Add

Cross

N: 2

Polynomial, Degree=N

MODELS

Two-way Factorial

Full Factorial

N: 2

N-way Factorial

Polynomial, Order=N

numstepper

This input type has these attributes:

Attribute	Description
decimalPlaces	Specifies the number of decimal places to display. Valid values include a single value or a range. To create a field that allows 0 to 3 decimal places, specify decimalPlaces="0,3".

Attribute	Description
increment	Specifies the number of values that the option increases or decreases when a user clicks the up or down arrow. The default value is 1.
maxValue	Specifies the maximum value that is allowed. If the user tries to exceed this value, a message appears. The default value is 9000000000000.
minValue	Specifies the minimum value that is allowed. If the user specifies a value that is below the minimum value, a message appears.
required	Specifies whether a value is required. Valid values are true and false. The default value is false.
width	Specifies the width of the control. This value can be in percent (%), em, or px. By default, SAS Studio sizes the control based on the available width and content.

The first example in the sample task definition creates an option with an assigned default value of 5.

```
<Option name="labelNumStepperEXAMPLE1" inputType="string">
  Numeric stepper with default values:</Option>
<Option name="basicStepperEXAMPLE" defaultValue="5" inputType="numstepper"
  indent="1">Label:</Option>
```

When you run this code, here is the resulting user interface:

Numeric stepper with default values.

Label:

The second example in the sample task definition creates an option with a specified minimum value, maximum value, and increment.

```
<Option name="labelNumStepperEXAMPLE2" inputType="string">
  Numeric stepper with a minimum value of -10, a maximum value of 120, and
  an increment of 2.</Option>
<Option name="advancedStepperEXAMPLE" defaultValue="80" inputType="numstepper"
  increment="2"
```



```
minValue="-10"  
maxValue="120"  
decimalPlaces="0,2"  
size="8em"  
indent="1">Label:</Option>
```

When you run the code, here is the resulting user interface:

Numeric stepper with a minimum value of -10, a maximum value of 120, and an increment of 2.

Label:

radio

This input type has one attribute:

Attribute	Description
variable	Specifies a variable that contains the name of the currently selected radio button.

The second example in the sample task definition creates an option called **Radio button group label** with the **Radio button 1** button selected by default.

```
<Option name="labelEXAMPLE2" inputType="string">Radio button group label:</Option>  
<Option name="radioButton1" variable="radioEXAMPLE" defaultValue="radioButton1"  
  inputType="radio">Radio button 1</Option>  
<Option name="radioButton2" variable="radioEXAMPLE"  
  inputType="radio">Radio button 2</Option>  
<Option name="radioButton3" variable="radioEXAMPLE"  
  inputType="radio">Radio button 3</Option>
```

When you run the code, here is the resulting user interface:

Radio button group label:

☐ Radio button 1

☐ Radio button 2

☐ Radio button 3

slider

This input type has these attributes:

Attribute	Description
discreteValues	Specifies the number of discrete values in the slider. For example, if discreteValues=3, the slider has three values: a minimum value, a maximum value, and a value in the middle.
maxValue	Specifies the maximum value for this option.
minValue	Specifies the minimum value for this option.
showButtons	Specifies whether to show the increase and decrease buttons for the slide. Valid values are true and false. The default value is true.

The first example in the sample task definition creates a slider option with buttons.

```
<Option name="labelSliderEXAMPLE1" inputType="string">
  Slider with buttons.</Option>
<Option name="labelSliderEXAMPLE1" defaultValue="80.00"
  inputType="slider" discreteValues="14" minValue="-10"
  maxValue="120">Label</Option>
```

When you run the code, here is the resulting user interface:

Slider with buttons.



The second example in the sample task definition creates a slider option without buttons.

```
<Option name="labelSliderEXAMPLE2"
  inputType="string">Slider without buttons.</Option>
<Option name="labelSliderEXAMPLE2" defaultValue="80.00"
  inputType="slider" discreteValues="14" minValue="-10"
  maxValue="120" showButtons="false">Label</Option>
```

When you run the code, here is the resulting user interface:

Slider without buttons.



string

This input type has no attributes. The code for the sample task definition contains several examples of the string input type. In the code for the slider option, the explanatory text (**Slider with buttons**) is created by the string input type.

```
<Option name="labelSliderEXAMPLE1" inputType="string">
  Slider with buttons.</Option>
<Option name="labelSliderEXAMPLE1" defaultValue="80.00"
  inputType="slider" discreteValues="14" minValue="-10"
  maxValue="120">Label</Option>
```

When you run the code, here is the resulting user interface:

Slider with buttons.



validationtext

This input type has these attributes:

Attribute	Description
required	Specifies whether any input text is required. Valid values are true and false. The default is false.
invalidMessage	Specifies the tooltip text to display when the content in the text box is invalid. By default, no message is displayed.
missingMessage	Specifies the tooltip text that appears when the text box is empty but text is required. By default, no message is displayed.
promptMessage	Specifies the tooltip text that appears when the text box is empty and the text box is selected. By default, no message is displayed.
regExp	Specifies the regular expression pattern to use for validation. This syntax comes directly from JavaScript Regular Expressions.
width	Specifies the width of the control. This value can be in percent (%), em, or px. By default, SAS Studio sizes the control based on the available width and content.

The code for the sample task definition creates a text box called **Label**.

```
<Option name="validationTextExample" defaultValue="99999"
  inputType="validationtext"
  required="true"
  width="100%"
  promptMsg="A message that tells the user what type of value to enter"
  invalidMsg="Invalid value message"
  missingMsg="This value is required."
  regExp="\d{5}">Label:
</Option>
```

When you run the code, here is the resulting user interface:

*Label:

99999

If you remove the default value from this box, the `This value is required` message appears.



The image shows a form element with a label `*Label:` and an empty input field. A blue error message box on the left contains the text `This value is required.` The input field has a red border and a red exclamation mark icon on the right side.

When the user begins entering a value, this message appears: A message that tells the user what type of value to enter.



The image shows a form element with a label `*Label:` and an input field containing the character `9`. A blue message box on the left contains the text `A message that tells the user what type of value to enter`. The input field has a blue border.

If the specified value is invalid, the `Invalid value message` appears.



The image shows a form element with a label `*Label:` and an input field containing the text `99A`. A blue error message box on the left contains the text `Invalid value message`. The input field has a red border and a red exclamation mark icon on the right side.

4

Working with the UI Element

About the UI Element

27

Example: UI Element for the Sample Task Definition

29

About the UI Element

This element is read by the UI engine to determine the layout of the user interface. Only linear layouts are supported. The `UI` tag is for grouping purposes only. There are no attributes associated with this tag.

The `UI` element has these children:

Child	Description
Container	<p>A tab that contains any options for the task. For example, you might want to display the option for selecting the input data and assigning columns to roles on the same page. The UI engine displays these options sequentially.</p> <p>A label is created for the tab. The <code>Container</code> tag takes only one attribute. The string for this option is the value of the <code>string</code> input type in the <code>Metadata</code> element.</p>

Child	Description
Group	<p>A title for a group of options. The UI engine displays these options sequentially.</p> <p>This tag takes these attributes:</p> <ul style="list-style-type: none"> ■ The <code>option</code> attribute is an option name in the metadata. This string is the same as the string value for the metadata option. ■ The <code>open</code> attribute specifies whether a group is expanded or collapsed. By default, <code>open=false</code>, and the group is collapsed in the user interface. To display the contents of a group by default, specify <code>open=true</code>.
DataItem	A reference to an input data source. This tag has only one attribute. The string for this option is the value of the <code>string</code> input type in the <code>Metadata</code> element.
RoleItem	A reference to a role. This tag has only one attribute. The string for this option is the value of the <code>string</code> input type in the <code>Metadata</code> element.
OptionItem	A reference to an option that has a single state. This type of option is either on or off, or has a single value (such as a series of radio buttons). This tag takes the <code>option</code> attribute only. The <code>option</code> attribute refers to the metadata name attribute for the option. The string for this option is taken from the metadata string value.
OptionChoice	A reference to an option that has a choice of values. This type of option could be displayed as a combination box. The <code>OptionChoice</code> element uses the <code>OptionItem</code> or <code>OptionValue</code> element to represent the choice of values.
OptionValue	A value choice. This tag is valid only as a child of the <code>OptionChoice</code> element.

Example: UI Element for the Sample Task Definition

The code for the sample task definition creates a group for each input type. Here is the code for the first three groups:

```
<UI>
  <Container option="DATATAB">
    <Group option="DATAGROUP" open="true"
      <DataItem data="DATASOURCE" />
    </Group>
    <Group option="ROLESGROUP" open="true">
      <RoleItem role="VAR"/>
      <RoleItem role="OPTNVAR"/>
      <RoleItem role="OPTCVAR"/>
    </Group>
  </Container>

  <Container option="OPTIONSTAB">
    <Group option="GROUP0" open="true">
      <OptionItem option="labelEXAMPLE"/>
    </Group>

    <Group option="GROUP1">
      <OptionItem option="chkCheck"/>
      <OptionItem option="chkEXAMPLE"/>
    </Group>

    <Group option="GROUP2">
      <OptionItem option="labelEXAMPLE2"/>
      <OptionItem option="radioButton1"/>
      <OptionItem option="radioButton2"/>
      <OptionItem option="radioButton3"/>
    </Group>

    ...
  </Container>
</UI>
```

When you run this code, the **Data** and **Options** tabs appear in the interface

The **Data** tab displays a selector for the input data source and three roles.

DATA

OPTIONS

INFORMATION

DATA

SASHELP.CLASS

ROLES

*Required variable label:

Column

Numeric variable label:

123

Column

Character variable label:

(3 items)

A

Column

The **Options** tab contains several groups. The example code creates the Groups, Check Boxes, and Radio Buttons groups. The first group is expanded by default

because the `open` attribute is set to `true`. (The sample task definition includes code to create the remaining groups on the **Options** tab.)

DATA | **OPTIONS** | INFORMATION |

▲ GROUPS

Groups are used to organize options.

- ▶ CHECK BOXES
- ▶ RADIO BUTTONS
- ▶ TEXT INPUT FIELD
- ▶ DROP-DOWN LIST
- ▶ NUMERIC STEPPER
- ▶ DATA VALIDATION
- ▶ COLOR SELECTOR
- ▶ SLIDER

5

Working with the Dependencies Element

<i>About the Dependencies Element</i>	33
<i>Notes on Dependencies</i>	35
<i>Example 1: Selecting a Check Box Enables Text Boxes</i>	37
<i>Example 2: Selecting a Check Box Enables a Combination Box</i>	38
<i>Example 3: Selecting a Radio Button Enables a Number Stepper Control</i>	40
<i>Example 4: The Selected Value for a Combination Box Enables the Text Box</i>	41
<i>Example 5: Selecting a Check Box Enables Multiple Types of Options</i>	43
<i>Example 6: Compound Condition Using AND and OR Logic</i>	44

About the Dependencies Element

The `Dependencies` element specifies how certain options rely on one another in order for the task to work properly. For example, a check box can enable or disable a text box depending on whether the check box is selected. The `Dependencies` element is a

grouping mechanism for the individual `Dependency` tags. There are no attributes associated with this element.

The `Dependencies` element can have multiple `Dependency` tags. Each `Dependency` tag has a `condition` attribute that is resolved to determine the state of the targets.

The `Target` element has three required attributes.

Attribute	Description
<code>option</code>	References the option that will receive the action.
<code>conditionResult</code>	<p>Specifies when to execute the action. The valid values for this attribute are <code>true</code> and <code>false</code>.</p> <ul style="list-style-type: none">■ If the condition is <code>true</code> and <code>conditionResult=true</code>, the action is executed.■ If the condition is <code>false</code> and <code>conditionResult=false</code>, the action is executed.■ If the value of the condition and <code>conditionResult</code> do not match (for example, one is <code>true</code> and one is <code>false</code>), the action is ignored.

Attribute	Description
<code>action</code>	<p>Specifies the action to execute. Here are the valid values:</p> <ul style="list-style-type: none"> ■ <code>show</code> ■ <code>hide</code> ■ <code>enable</code> ■ <code>disable</code> ■ <code>set</code> <p>If the value of the <code>action</code> attribute is <code>set</code>, you must also specify these two attributes:</p> <ul style="list-style-type: none"> □ The <code>property</code> attribute refers to the attribute of an element that was created from the metadata. The <code>option</code> element in the metadata has an <code>inputType</code> attribute that specifies what UI element is created. <p>With a few exceptions, any attribute that is listed for an input type is a valid value for the <code>property</code> attribute of a Target element. The invalid values are <code>required</code>, <code>width</code>, <code>indent</code>, and <code>variable</code>.</p> <ul style="list-style-type: none"> □ The <code>value</code> attribute is the value to use for the target of the <code>property</code> attribute.

Notes on Dependencies

- If `action=hide`, the associated option is not defined. If `action=show`, the associated option is defined and appears in the SAS code that is generated by the Velocity script.
- Not all dependencies are evaluated each time the Velocity script runs and produces the SAS code. When the task is first opened, all dependencies are run to establish initial values. After that, only dependencies that are linked to the current interaction in the user interface are evaluated. The value of the `condition` attribute determines whether a dependency is evaluated.
- Dependencies are evaluated in top-down order.

- An option is order independent if the option name appears only in the condition attribute of the Target element.
- An option is order dependent if the option name appears in both the condition and option attributes of the Target element.

This example shows a correct and incorrect ordering of dependencies:

```
<UI>
  <Container option="options">
    <Group option="basic options">
      <Option name="COMBOBOX"/>
      <Option name="ITEM1"/>
      <Option name="ITEM2"/>
      <Option name="ITEM3"/>
      <OptionItem option="CHECKBOX"/>
      <OptionItem option="INPUTTEXT"/>
    </Group>
  </Container>
</UI>

<Dependencies>
1 <!-- Correct ordering of the dependencies -->
  <Dependency condition="$COMBOBOX=='ITEM1'">
    <Target conditionResult="true" option="CHECKBOX" action="set"
      property="value" value="1"/>
  </Dependency>
  <Dependency condition="$CHECKBOX=='1'">
    <Target conditionResult="true" option="INPUTTEXT" action="enable"/>
    <Target conditionResult="false" option="INPUTTEXT" action="disable"/>
  </Dependency>

2 <!-- Incorrect ordering to the dependencies -->
  <Dependency condition="$CHECKBOX=='1'">
    <Target conditionResult="true" option="INPUTTEXT" action="enable"/>
    <Target conditionResult="false" option="INPUTTEXT" action="disable"/>
  </Dependency>
  <Dependency condition="$COMBOBOX=='ITEM1'">
    <Target conditionResult="true" option="CHECKBOX" action="set"
      property="value" value="1"/>
  </Dependency>
</Dependencies>
```

- 1 This first dependency is order independent. COMBOBOX is a name that is used in the condition, but the value of COMBOBOX is not a target in any of the other dependencies.

- 2 The second dependency is order dependent. CHECKBOX is used in the condition, and the value of CHECKBOX is also a target for `option="CHECKBOX"` in the preceding Dependency element. In this case, the state for INPUTTEXT is not evaluated properly because `condition="$CHECKBOX=='1'"` is evaluated before `condition="$COMBOBOX=='ITEM1'"`.

Example 1: Selecting a Check Box Enables Text Boxes

In this example from the Characterize Data task, the selection of the **SAS data sets** check box determines whether the **Frequency data** and **Univariate data** text boxes are available. (In the **Tasks** section, the Characterize Data task is in the **Data** group.)

In this example, SASDATASETS is the name of the check box. FRQDATA and UNIDATA are the names of the input text fields.

```
<Option name=SASDATASETS" defaultValue="1" inputType="checkbox">
  SAS data sets</Option>
<Option name="FRQDATA" indent="1" defaultValue="WORK.CharacterizeDataFRQ"
  inputType="inputtext" width="100%">Frequency data:</Option>
<Option name="UNIDATA" indent="1" defaultValue="WORK.CharacterizeDataUNI"
  inputType="inputtext" width="100%">Univariate data:</Option>

<Dependency condition="$SASDATASETS=='1'"/>
  <Target conditionResult="true" option="FRQDATA" action="enable"/>
  <Target conditionResult="false" option="FRQDATA" action="disable"/>
  <Target conditionResult="true" option="UNIDATA" action="enable"/>
  <Target conditionResult="true" option="UNIDATA" action="enable"/>
</Dependency>
```

By default, the **SAS data sets** check box is selected, so the **Frequency data** and **Univariate data** text boxes are enabled. Here are the results that appear on the **Options** tab:

☒ SAS data sets

*Frequency data:

*Univariate data:

If you clear the **SAS data sets** check box, the **Frequency data** and **Univariate data** text boxes are not available. Here are the results that appear on the **Options** tab:

☐ SAS data sets

*Frequency data:

*Univariate data:

Example 2: Selecting a Check Box Enables a Combination Box

In this example from the Summary Statistics task, the **Quantile method** drop-down list is enabled only if the user selects the check box for at least one of the quantile options. (In the **Tasks** section, the Summary Statistics task is in the **Statistics** group.)

In this example, P1, P5, P10, Q1, MEDIAN, Q3, P90, P95, and P99 are the names of the check boxes. QUANTILE is the name of the combination box.

```
<Option name="P1" defaultValue="0" inputType="checkbox">1st</Option>
<Option name="P5" defaultValue="0" inputType="checkbox">5th</Option>
<Option name="P10" defaultValue="0" inputType="checkbox">10th</Option>
<Option name="Q1" defaultValue="0" inputType="checkbox">Lower quartile</Option>
<Option name="MEDIAN" defaultValue="0" inputType="checkbox">Median</Option>
```

```

<Option name="Q3" defaultValue="0" inputType="checkbox">Upper quartile</Option>
<Option name="P90" defaultValue="0" inputType="checkbox">90th</Option>
<Option name="P95" defaultValue="0" inputType="checkbox">95th</Option>
<Option name="P99" defaultValue="0" inputType="checkbox">99th</Option>
<Option name="QUANTILE" defaultValue="QMOS" inputType="checkbox"
  width="100%">Quantile method:</Option>
  <Option name="QMOS" inputType="string">Order statistics</Option>
  <Option name="QMPPA" inputType="string">Piecewise-parabolic
    algorithm</Option>

<Dependency condition="$P1=='1' || $P5=='1' || $P10=='1' || $Q1=='1' ||
  $MEDIAN=='1' || $Q3=='1' || $P90=='1' || $P95=='1' || $P99=='1'"/>
  <Target conditionResult="true" option="QUANTILE"
    action="enable"/>
  <Target conditionResult="false" option="QUANTILE"
    action="disable"/>
</Dependency>

```

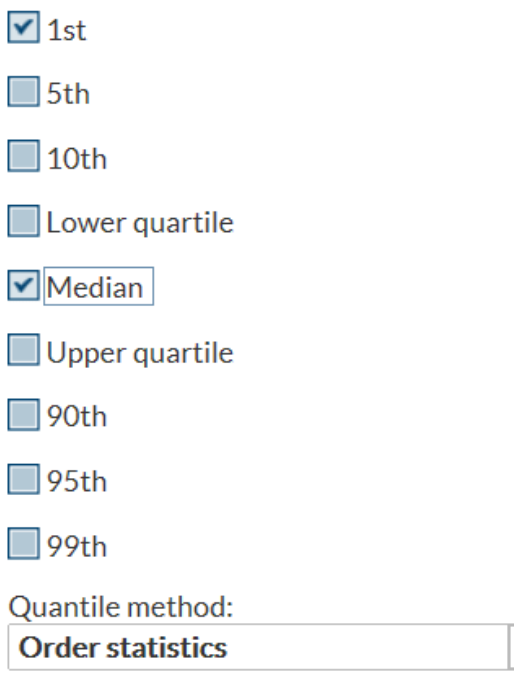
By default, no check boxes are selected, so the **Quantile method** drop-down list is not available. Here is the result that appears on the **Options** tab:

- ☐ 1st
- ☐ 5th
- ☐ 10th
- ☐ Lower quartile
- ☐ Median
- ☐ Upper quartile
- ☐ 90th
- ☐ 95th
- ☐ 99th

Quantile method:

Order statistics

If you selected one or more of these check boxes, the **Quantile method** drop-down list is available. Here is the result that appears on the **Options** tab:



A screenshot of a web form's 'Options' tab. It contains a list of nine checkboxes with labels: '1st', '5th', '10th', 'Lower quartile', 'Median', 'Upper quartile', '90th', '95th', and '99th'. The '1st' and 'Median' checkboxes are checked. Below this list is a label 'Quantile method:' followed by a text input field containing the text 'Order statistics'.

Example 3: Selecting a Radio Button Enables a Number Stepper Control

In this example from the Rank Data task, the selection of the **Group = n (NTILES)** option determines whether the **Number of groups** option is available. (In the **Tasks** section, the Rank Data task is in the **Data** group.)

In this example, RMGN is the name of the radio button, and NUMGRPS is the name of the numstepper variable.

```
<Option name="RMGN" inputType="radio" variable="RMGRP">Group = n (NTILES)</Option>
<Option name="NUMGRPS" defaultValue="1" minValue="0" inputType="numstepper"
  indent="1">Number of groups:</Option>

<Dependency condition="$RMGRP.equalsIgnoreCase('RMGN')"/>
```

```

<Target conditionResult="true" option="NUMGRPS" action="enable"/>
<Target conditionResult="false" option="NUMGRPS" action="disable"/>
</Dependency>

```

By default, the **Group = n (NTILES)** option is not selected, so the **Number of groups** option is not available. Here is the result that appears on the **Options** tab:

☐ Group = n (NTILES)

Number of groups:

When you select the **Group = n (NTILES)** option, the **Number of groups** option is available. Here is the result that appears on the **Options** tab:

☒ Group = n (NTILES)

Number of groups:

Example 4: The Selected Value for a Combination Box Enables the Text Box

In this example from the Sort Data task, the **Value** text box is available only if you select **B (bytes)**, **KB (kilobytes)**, **MB (megabytes)**, or **GB (gigabytes)** from the **Memory for sorting** drop-down list. (In the **Tasks** section, the Sort Data task is in the **Data** group.)

In this example, MEMSORT is the name of the drop-down list that is created by the combobox variable. The MEMSORT option has these values: MSSD (Server default), MSMA (Maximum allowed), MSB (B (bytes)), MSKB (KB (kilobytes)), MSMB (MB (megabytes)), and MSGB (GB (gigabytes)). MEMAMT is the name of the validation text box.

```

<Option name="MEMSORT" defaultValue="MSSD" inputType="combobox">

```

```

Memory for sorting:</Option>
  <Option name="MSSD" inputType="string">Server default</Option>
  <Option name="MSMA" inputType="string">Maximum available</Option>
  <Option name="MSB" inputType="string">B (bytes)</Option>
  <Option name="MSKB" inputType="string">KB (kilobytes)</Option>
  <Option name="MSMB" inputType="string">MB (megabytes)</Option>
  <Option name="MSGB" inputType="string">GB (gigabytes)</Option>
<Option name="MEMAMT" defaultValue="0" inputType="validationtext"
  regExp="[0-9]*$" invalidMessage="Enter an integer value
  greater than or equal to 0.">Value:</Option>

<Dependency condition="$MEMSORT.equalsIgnoreCase('MSB') ||
  $MEMSORT.equalsIgnoreCase('MSKB') ||
  $MEMSORT.equalsIgnoreCase('MSMB') ||
  $MEMSORT.equalsIgnoreCase('MSGB') ||" />
  <Target conditionResult="true" option="MEMAMT" action="enable"/>
  <Target conditionResult="false" option="MEMAMT" action="disable"/>
</Dependency>

```

By default, the **Server default** option is selected, so the **Value** option is not available. Here is the result that appears on the **Options** tab:

Memory for sorting:

*Value:

If you select the **B (bytes)** option, the **Value** option is available. Here is the result that appears on the **Options** tab:

Memory for sorting:

*Value:

Example 5: Selecting a Check Box Enables Multiple Types of Options

In this example from the Table Attributes task, selecting the **Enhanced report** check box enables the **Sort variables by** drop-down list and the **Ascending** and **Descending** radio buttons. (In the **Tasks** section, the Table Attributes task is in the **Data** group.)

In this example, ENHANCEDRPT is the name of the check box. SORTROWS, ASCENDING, and DESCENDING are the names of the **Sort variables by**, **Ascending**, and **Descending** options, respectively.

```
<Option name="ENHANCEDRPT" defaultValue="1" inputType="checkbox">
  Enhanced report</Option>
<Option name="SORTROWS" defaultValue="SRVN" inputType="combobox"
  width="100%">In the data variable table, sort rows by:</Option>
  <Option name="SRVN" inputType="string">Variable name</Option>
  <Option name="SRVO" inputType="string">Variable order in table</Option>
  <Option name="SRVT" inputType="string">Variable type</Option>
  <Option name="SRVF" inputType="string">Variable format</Option>
  <Option name="SRVL" inputType="string">Variable label</Option>
<Option name="ORDERSEQ" inputType="string">Order sequence:</Option>
<Option name="ASCENDING" variable="ORDERRADIO" inputType="radio"
  indent="1" defaultValue="1">Ascending</Option>
<Option name="DESCENDING" variable="ORDERRADIO"
  inputType="radio" indent="1">Descending</Option>

<Dependency condition="$ENHANCEDRPT=='1'"/>
  <Target conditionResult="true" option="SORTROWS" action="enable"/>
  <Target conditionResult="true" option="ASCENDING" action="enable"/>
  <Target conditionResult="true" option="DESCENDING" action="enable"/>
  <Target conditionResult="false" option="SORTROWS" action="disable"/>
  <Target conditionResult="false" option="ASCENDING" action="disable"/>
  <Target conditionResult="false" option="DESCENDING" action="disable"/>
</Dependency>
```

By default, the **Enhanced report** check box is selected, so all options are available. Here is the result that appears on the **Options** tab:

☒ Enhanced report

Sort variables by:
 ▼

Order sequence:

☒ Ascending

☐ Descending

If you clear the **Enhanced report** check box, the **Sort variables by**, **Ascending**, and **Descending** options are not available. Here is the result that appears on the **Options** tab:

☐ Enhanced report

Sort variables by:
 ▼

Order sequence:

☐ Ascending

☐ Descending

Example 6: Compound Condition Using AND and OR Logic

In this example from the One-Way Frequencies task, the **Maximum time (seconds)** option is available only if you select the **Exact test** check box (for Binomial proportion or Chi-square goodness of fit) and the **Limit computation time** check box. (In the **Tasks** section, the One-Way Frequencies task is in the **Statistics** group.)

In this example, SECONDS is the name of the number stepper control. chkEXACTP is the name of the **Exact test** check box for binomial proportions. chkCHIEXACTP is the name of the **Exact test** check box for chi-square goodness of fit. chkLIMIT is the name of the **Limit computation time** check box.

```
<Option name="chkEXACTP" indent="1" defaultValue="0"
  inputType="checkbox">Exact test</Option>
<Option name="chkCHIEXACTP" indent="1" defaultValue="0"
  inputType="checkbox">Exact test</Option>
<Option name="chkLIMIT" indent="1" defaultValue="1"
  inputType="checkbox">Limit computation time</Option>
<Option name="SECONDS" indent="2" defaultValue="900"
  inputType="checkbox">Maximum time (seconds):</Option>

<Dependency condition="(((($chkEXACTP=='1') || ($chkCHIEXACTP=='1'))
  &and; ($chkLIMIT=='1')))" />
  <Target conditionResult="true" option="SECONDS" action="enable"/>
  <Target conditionResult="false" option="SECONDS" action="disable"/>
</Dependency>
```

By default, neither of the **Exact test** check boxes is selected, so the **Maximum time (seconds)** option is not available. Here is the result that appears on the **Options** tab:

Binomial proportion

☒ Asymptotic test

*Test proportion:

undefined

☐ Exact test

Chi-square goodness-of-fit

☒ Asymptotic test

☐ Exact test

☐ Use Monte Carlo estimation

Confidence level:

Note: For some large problems, computation of exact tests might require a large amount of time and memory. Consider using asymptotic tests for such problems. Alternatively, when asymptotic methods might not be sufficient for such large problems, consider using Monte Carlo estimation of exact p-values.

Exact Computation Methods

☒ Limit computation time

*Maximum time (seconds):

When you select either of the **Exact test** check boxes and the **Limit computation time** check box, the **Maximum time (seconds)** option is available. Here is the result that appears on the **Options** tab:

▲ Binomial proportion

☒ Asymptotic test

*Test proportion:

undefined

☒ Exact test

▲ Chi-square goodness-of-fit

☒ Asymptotic test

☐ Exact test

☐ Use Monte Carlo estimation

Confidence level:

Note: For some large problems, computation of exact tests might require a large amount of time and memory. Consider using asymptotic tests for such problems. Alternatively, when asymptotic methods might not be sufficient for such large problems, consider using Monte Carlo estimation of exact p-values.

▲ Exact Computation Methods

☒ Limit computation time

*Maximum time (seconds):

6

Working with the Requirements Element

<i>About the Requirements Element</i>	47
<i>Example: Using a Requirements Element for Roles</i>	48

About the Requirements Element

The `Requirements` element specifies a list of conditions that must be met in order for the task to run. If the condition is true, SAS code can be generated. If the condition is false, no code is generated. When defining a requirement, you can specify the message to display when the requirement is not met.

The `Requirements` element can have multiple `Requirement` tags. Each `Requirement` tag has a `condition` attribute, which is a conditional expression that is used to evaluate whether the requirement is met. The conditional expression that is used is identical to the conditional expression in Apache Velocity.

Each `Requirement` tag also has a `Message` element, which has no attributes. The value of this element is the message that is displayed if the condition is not satisfied.

Because dependencies can affect the state of the user interface as well as the state of the Velocity variables, the `Requirements` element is evaluated after the `Dependencies` element. As a result, any changes due to dependencies are made before determining whether the requirements are satisfied.

Example: Using a Requirements Element for Roles

In this example, the code refers to three roles: AVAR, BYVAR, and FVAR. The user must assign a variable to at least one of these roles in order for the task to run. If no variables are assigned to any of these roles, the SAS code cannot be generated, and the task will not run.

```
<Metadata>
  <Roles>
    <Role maxVars="0" minVars="1" name="AVAR" nlsKey="AVARKey"
      order="true" type="A">Analysis variables<Role>
    <Role maxVars="0" minVars="1" name="BYVAR" nlsKey="BYVARKey"
      order="true" type="A">Group analysis by<Role>
    <Role maxVars="0" minVars="1" name="FVAR" nlsKey="FVARKey"
      order="true" type="N">Frequency count<Role>
  </Roles>
  ...
</Metadata>

<Requirements>
  <Requirement condition="$AVAR.size() > 0 || $BYVAR.size() > 0
    || $FVAR.size() > 0">
    <Message>At least one variable must be assigned to the Analysis
      variables role, the Group analysis by role, or the Frequency
      count role.</Message>
  </Requirement>
</Requirements>
```

7

Understanding the Code Template

- About the Code Template* 49
- Predefined Variables* 50
- Predefined SAS Macros* 50
- Data Source* 51
- Roles* 52
- Options* 52
 - checkbox 52
 - color 53
 - combobox 53
 - inputtext 54
 - numstepper 54
 - radio 55
 - slider 56
 - validationtext 56
 - string 57
 - modelbuilder 57

About the Code Template

The code template creates the string output of the task. For most tasks, this output is SAS code. The only contents in the Code Template element is a CDATA block of the

Apache Velocity scripting language. The string output is produced using this scripting language.

Predefined Variables

To access option variables, a Velocity variable is defined for each option. The names of these variables correlate to the option name attribute. For example, to access a check box with a `name` attribute of `cbx1`, a Velocity variable of `$cbx1` is defined.

Here are the predefined Velocity variables:

Variable	Description
<code>\$SASLIBRARY</code>	The name of the library that contains the input data source. The <code>\$SASLIBRARY</code> variable is defined only if you have a <code>DataSource</code> element in the task.
<code>\$SASTABLE</code>	The table name of the input data source. The <code>\$SASTABLE</code> variable is defined only if you have a <code>DataSource</code> element in the task.

Predefined SAS Macros

If you need to generate SAS code, SAS Studio has these predefined macros:

SAS Macro	Description
<code>%web_drop_table(table-name1, table-name2, ... table-name_n)</code>	Drops the specified tables
<code>%web_open_table(table-name)</code>	Opens the specified table
<code>%web_open_file(filename, type)</code>	Opens the specified file with the specified MIME type

SAS Macro	Description
<code>%web_open_url(url)</code>	Opens the specified URL

Data Source

For the input data source, a Velocity variable is used to access information about the data. This variable is the same as the name attribute for the `DataSource` tag.

In this example, the input data source is given the name of `DATASOURCE`. As a result, the Velocity variable, `$DATASOURCE`, is created. When this script is run, the `$DATASOURCE` variable is checked to see whether an input data source has been assigned.

```
<DataSource name="DATASOURCE">
</DataSource>
```

```
...
```

```
<CodeTemplate>
  <![CDATA[
%put DATASET=$DATASOURCE;
%put SASLIBRARY=$SASLIBRARY;
%put SASTABLE=$SASTABLE;
```

```
...
```

```
%web_open_table($DATASOURCE);
```

```
]]>
</CodeTemplate>
```

Roles

For each role, a Velocity variable is used to access the role information. This variable is the same as the role's name attribute.

In this example, the **Analysis Group** role is given the name of BY. As a result, the Velocity variable, \$BY, is created. When this script is run, the \$BY variable is checked to see whether any columns are assigned. If the user has assigned any columns to the **Analysis Group** role, the generated SAS code sorts on these columns.

```
<Roles>
  <Role type="A" maxVars="0" order="true" minVars="0"
    name="VAR">Columns</Role>
  <Role type="A" maxVars="0" order="true" minVars="0"
    name="BY">Analysis group</Role>
  <Role type="N" maxVars="0" order="true" minVars="0"
    name="SUM">Total of</Role>
  <Role type="A" maxVars="0" order="true" minVars="0"
    name="ID">Identifying label</Role>
</Roles>
<CodeTemplate>
  <![CDATA[
    #if( $BY )/* Sort $DATASET for BY group processing. */

    PROC SORT DATA=$DATASET OUT=WORK.SORTTEMP;
      BY #foreach($item in $BY ) $item#end;
    #end
    RUN;]]>
</CodeTemplate>
```

Options

checkbox

The `checkbox` variable holds the state information for the check box option. If the check box is selected, the variable is set to 1. If the check box is not selected, the variable is set to 0.

In this example, the code outputs the character **N** if the **Print row numbers** check box is selected.

```
<Options>
  <Option name="PRINTNUMROWS" defaultValue="1"
    inputType="checkbox">Print row numbers</Option>
</Options>
<Code Template>
  <![CDATA[
#if ($PRINTNUMROWS == '1')
    N
#end]]>
</CodeTemplate>
```

color

The `color` variable holds the specified color.

In this example, the code template is printed as `colorEXAMPLE=specified-color`.

```
<Options>
  <Option name="colorEXAMPLE" defaultValue="white"
    inputType="color">Select a color</Option>
</Options>
<CodeTemplate>
  <![CDATA[
%put colorEXAMPLE=$colorEXAMPLE;
#end]]>
</CodeTemplate>
```

combobox

The `combobox` variable holds the name of the selected option. If no option is selected, the variable is null.

This example outputs the string `HEADING=option-name`, where *option-name* is the value selected from the **Direction of heading** drop-down list. If the user selects **Horizontal** from the **Direction of heading** drop-down list, the output is `HEADING="horizontal"`.

```
<Options>
  <Option name="HEADING" defaultValue="default"
```

```

        inputType="combobox">Direction of heading:</Option>
    <Option name="default" inputType="string">Default</Option>
    <Option name="horizontal" inputType="string">Horizontal</Option>
    <Option name="vertical" inputType="string">Vertical</Option>
</Options>
<UI>
    <Container option="OPTIONSTAB">
        <OptionChoice option="HEADING">
            <OptionItem option="default"/>
            <OptionItem option="horizontal"/>
            <OptionItem option="vertical"/>
        </OptionChoice>
    </Container>
</UI>
<CodeTemplate>
    <![CDATA[
    #if ($HEADING && (&HEADING != "default"))
        HEADING=$HEADING
    #end
    ]]>
</CodeTemplate>

```

inputtext

The `inputtext` variable holds the string that was specified in the text box.

This example outputs the string `OBS=` and the text specified in the **Column** text box. If the user enters **Student Number** into the **Column** text box, the output is `OBS="Student Number"`.

```

<Options>
    <Option name="OBSHEADING" indent="1" defaultValue="Row number"
        inputType="inputtext">Column label:</Option>
</Options>
<CodeTemplate>
    <![CDATA[
    OBS="$OBSHEADING"#end]]>
</CodeTemplate>

```

numstepper

The `numstepper` variable holds the string specified in the number control box.

This example outputs the string `GROUPS=` and the value in the **Number of groups** box. If the user enters 2 into the **Number of groups** text box, the string output is `GROUPS=" 2"`.

```
<Options>
  <Option name="NUMGRPS" defaultValue="1" minValue="0"
    inputType="numstepper" indent="1">Number of groups:</Option>
</Options>
<CodeTemplate>
  <![CDATA[
GROUPS="$NUMGRPS"#end]]>
</CodeTemplate>
```

radio

The radio button options are grouped together with the same variable attribute. It is this attribute that defines the Velocity scripting variable. The Velocity scripting variable holds the name of the selected radio button. If no radio button is selected, the variable is null.

In this example, there are four radio buttons.

- If the first radio button is selected, there is no output.
- If the second radio button is selected, the string output is `GROUPS=" 100"`.
- If the third radio button is selected, the string output is `GROUPS=" 10"`.
- If the fourth radio button is selected, the string output is `GROUPS=" 4"`.

```
<Options>
  <Option name="RMSL" inputType="radio" variable="RMGRP"
    defaultValue="1">Smallest to largest</Option>
  <Option name="RMPR" inputType="radio"
    variable="RMGRP">Percentile ranks</Option>
  <Option name="RMDC" inputType="radio" variable="RMGRP">Deciles</Option>
  <Option name="RMQR" inputType="radio" variable="RMGRP">Quartiles</Option>
</Options>
<CodeTemplate>
  <![CDATA[
#if ($RMGRP.equalsIgnoreCase("RMPR")) GROUP=100 #end
#if ($RMGRP.equalsIgnoreCase("RMDC")) GROUP=10 #end
#if ($RMGRP.equalsIgnoreCase("RMQR")) GROUP=4 #end
]]>
</CodeTemplate>
```

slider

The `slider` variable holds the numeric string that is specified on the slider control.

This example outputs the string `datalabelattrs=(size="n")`, where *n* is the value of the **Label Font Size** option. If the value of the **Label Font Size** option is 10, the output is `datalabelattrs=(size=10)`.

```
<Options>
  <Option name="labelSIZE" defaultValue="7" inputType="slider"
    discreteValues="16" minValue="5" maxValue="20">Label Font Size</Option>
</Options>
<CodeTemplate>
  <![CDATA[
    datalabelattrs=(size=$labelSIZE)]>
</CodeTemplate>
```

validationtext

The `validationtext` variable holds the string that was specified in the text box.

The following example outputs the string `OBS=` and the text specified in the **Column** text box. If the user enters **Student Number** into the **Column** text box, the output is `OBS="Student Number"`.

```
<Options>
The inputtext variable holds the string entered into the text field.

The following example outputs the string "rho0=" and the text in
the Null hypothesis correlation option. If the user specifies 0,
the string rho0=0 is generated.
<source lang="xml">
<Options>
  <Option name="nullRho" indent="1" inputType="validationtext"
    defaultValue="0" required="true"
    promptMessage="Enter a number greater than -1 and less than 1
      for the null hypothesis correlation"
    invalidMessage="Enter a number greater than -1 and less than 1
      for the null hypothesis correlation"
    missingMessage="Enter a number grearter than -1 and less than 1
      for the null hypothesis correlation"
    regExp="[-+]?(0\.\d*)|(\.\d+)|0">Null hypothesis correlation:</Option>
</Options>
```

```
<CodeTemplate>
  <![CDATA[
rho0=$nullRho]]>
</CodeTemplate>
```

string

This input type cannot be accessed within the Velocity script.

modelbuilder

The Model Effects Builder is a custom component. This example code shows how the Model Effects Builder might be used in the user interface for a task. The Velocity code shows how to process the effects that are generated by the `modelbuilder` component.

```
<Metadata>
  <DataSources>
    <DataSource name="dataset">
      <Roles>
        <Role type="N" maxVars="0" minVar="1" order="true"
          name="CONTVARS">Continuous variables</Role>
        <Role type="A" maxVars="0" minVar="0" order="true"
          name="CLASSVARS">Classification variables</Role>
      </Roles>
    </DataSource>
  </DataSources>
  <Options>
    <Option inputType="string" name="modelGroup">MODEL</Option>
    <Option inputType="string" name="modelTab">MODEL</Option>
    <Option excludeTools="THREEFACT, NFACTPOLY" inputType="modelbuilder"
      name="modelBuilder roleClassification="classVariables"
      roleContinuous="continuousVariables" width="100%">Model</Option>
    <Option inputType="string" name="responseGroup">Response</Option>
  </Options>
</Metadata>
<UI>
  <Container>
    <Container option="modelTab">
      <Group open="true" option="modelGroup">
        <OptionItem option="modelBuilder"/>
      </Group>
    </Container>
  </UI>
```

```

<CodeTemplate>
<![CDATA[

#macro ( ModelEffects )
#if ( $modelBuilder )
#foreach ( $item in $modelBuilder )
## if first element is 'm', then this is a main effect
#if ( $item.get(0) == 'm' )
#foreach( $subitem in $item.get(1) )$subitem #end
## if first element is 'i', then this is an interaction effect
#elseif ( $item.get(0) == 'i' )
#foreach( $subitem in $item.get(1) )$subitem#if($velocityCount
    < $item.get(1).size())*#else #end#end
## if first element is 'n', then this is a nested effect
#elseif ( $item.get(0) == 'n' )
#foreach( $subitem1 in $item.get(1) )$subitem1#if($velocityCount
    < $item.get(1).size())*#end#end(#foreach($subitem2 in
    $item.get(2))$subitem(2)#if($velocityCount <
    $item.get(2).size())*#end#end)
#end
#end
#end
#end

]]>
</CodeTemplate>

```

8

Example: Task Definition for List Data Task

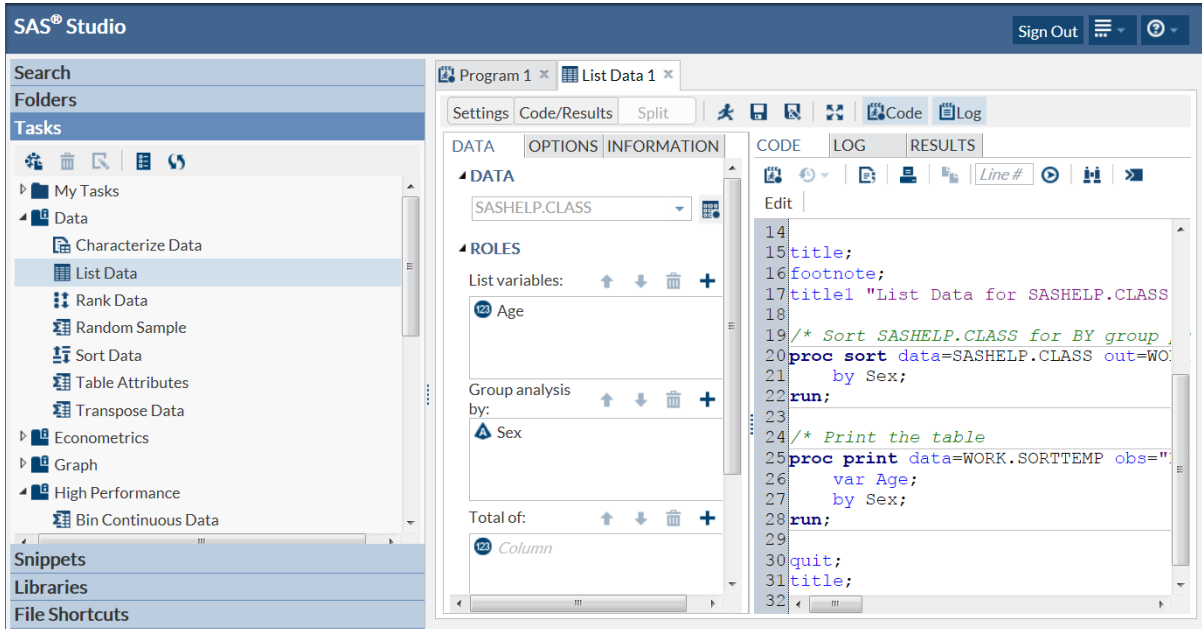
<i>Open the List Data Task</i>	59
<i>View the XML Code for the List Data Task</i>	60
<i>Understanding the XML Code for the List Data Task</i>	61

Open the List Data Task

To view the user interface for a predefined task:

- 1 In the navigation pane, open the **Tasks** section.
- 2 Expand the **Data** folder.
- 3 Right-click **List Data** and select **Open**. Alternatively, you can double-click the name of the task to open it.


The task opens to the right of the work area.

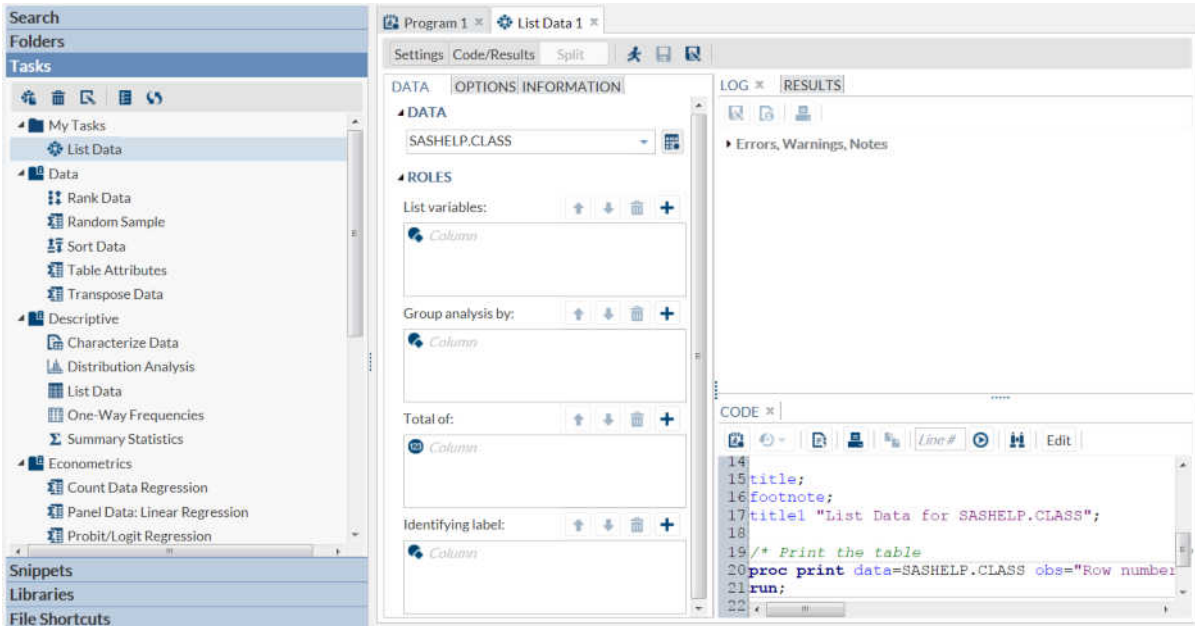


View the XML Code for the List Data Task

You cannot edit the code for a predefined task. However, you can copy the task code and edit the copy.

To view the code for a predefined task:

- 1 In the navigation pane, open the **Tasks** section.
- 2 Expand the **Data** folder.
- 3 Right-click **List Data** and select **Add to My Tasks**. A copy of the task is added to your **My Tasks** folder.
- 4 Open the **My Tasks** folder and select the copied task.
- 5 Click . The XML and Velocity code for the task appears.



Understanding the XML Code for the List Data Task

This example shows the task definition for the List Data task and labels each element in the XML code.

```
<?xml version="1.0" encoding="UTF-16"?><Task schemaVersion="1.0">
1<Registration>
  <Name>List Data</Name>
  <Description>The List Data task displays the contents of a table as a
    report.</Description>
  <GUID>3C74D5C3-B845-4926-A749-E56CBA1283E2</GUID>
  <Procedures>PRINT SORT</Procedures>
  <Version>1</Version>
  <Links>
    <Link href="http://support.sas.com/cdlsearch?nh=25&ct=80000&qt=PROC+PRINT">PROC PRINT Documentation</Link>
    <Link href="http://support.sas.com/cdlsearch?nh=25&ct=80000&qt=PROC+SORT">PROC SORT Documentation</Link>
  </Links>
</Registration>
```

```

2<Metadata>
  <DataSources>
    <DataSource name="DATASOURCE">
      <Roles>
        <Role maxVars="0" minVars="0" name="VAR" order="true"
          type="A">List variables</Role>
        <Role maxVars="0" minVars="0" name="BY" order="true"
          type="A">Group analysis by</Role>
        <Role maxVars="0" minVars="0" name="SUM" order="true"
          type="N">Total of</Role>
        <Role maxVars="0" minVars="0" name="ID" order="true"
          type="A">Identifying label</Role>
      </Roles>
    </DataSource>
  </DataSources>

  <Options>
    <Option defaultValue="all" inputType="combobox" name="ROWS2LIST"
      width="264px">Rows to list:</Option>
    <Option inputType="string" name="all">All rows</Option>
    <Option inputType="string" name="firstnrows">First n rows</Option>
    <!--<Option inputType="string" name="firstnpct">First n percent
      of rows</Option>-->
    <!--<Option inputType="string" name="everynrow">Every nth row</Option>-->
    <Option decimalPlaces="0" defaultValue="10" indent="1"
      inputType="numstepper" maxValue="999999" minValue="1"
      name="NVALUE">Amount (n):</Option>
    <!--<Option decimalPlaces="0" defaultValue="25" indent="1"
      inputType="numstepper" maxValue="100" minValue="1"
      name="NPERCENT">Percent (n):</Option>-->

    <Option inputType="string" name="DATATAB">DATA</Option>
    <Option inputType="string" name="DATAGROUP">DATA</Option>
    <Option inputType="string" name="ROLESGROUP">ROLES</Option>

    <Option inputType="string" name="OPTIONSTAB">OPTIONS</Option>
    <Option inputType="string" name="BASICOPTIONS">BASIC OPTIONS</Option>

    <Option defaultValue="1" inputType="checkbox" name="OBS">
      Display row numbers</Option>
    <Option defaultValue="Row number" indent="1" inputType="inputtext"
      name="OBSHEADING" width="250px">Column label</Option>

    <Option defaultValue="1" inputType="checkbox" name="LABEL">Use column
      labels as column headings</Options>

    <Option defaultValue="0" inputType="checkbox" name="PRINTNUMROWS">Display

```

```

    number of rows</Option>

<Option defaultValue="0" inputType="checkbox" name="ROUND">Round values
    before summing the variable</Option>

<Option defaultValue="default" inputType="combobox" name="HEADING"
    width="264px">Heading direction:</Option>
<Option inputType="string" name="default">Default</Option>
<Option inputType="string" name="horizontal">Horizontal</Option>
<Option inputType="string" name="vertical">Vertical</Option>

<Option defaultValue="default" inputType="combobox" name="WIDTH"
    width="264px">Column width</Option>
<Option inputType="string" name="full">Full</Option>
<Option inputType="string" name="minimum">Minimum</Option>
<Option inputType="string" name="uniform">Uniform</Option?
<Option inputType="string" name="uniformby">Uniform by</Option>

<Option defaultValue="0" inputType="checkbox"
    name="SPLITLABEL">Split labels</Option>
<Option defaultValue="*" indent="1" inputType="combobox"
    name="SPLITLABELVALUE"
    width="80px">Split character</Option>
</Options>
</Metadata>
3 <UI>
    <Container option="DATATAB">
        <Group open="true" option="DATAGROUP">
            <DataItem data="DATASOURCE"/>
        </Group>
        <Group open="true" option="ROLESGROUP">
            <RoleItem role="VAR"/>
            <RoleItem role="BY"/>
            <RoleItem role="SUM"/>
            <RoleItem role="ID"/>
        </Group>
    </Container>

    <Container option="OPTIONSTAB">

        <Group open="true" option="BASICOPTIONS">

            <OptionItem option="OBS"/>
            <OptionItem option="OBSHEADING"/>
            <OptionItem option="LABEL"/>
            <OptionItem option="PRINTNUMROWS"/>
            <OptionItem option="ROUND"/>

```

```

    <OptionChoice option="HEADING"/>
      <OptionItem option="default"/>
      <OptionItem option="horizontal"/>
      <OptionItem option="vertical"/>
    </OptionChoice>

    <OptionChoice option="WIDTH"/>
      <OptionItem option="default"/>
      <OptionItem option="full"/>
      <OptionItem option="minimum"/>
      <OptionItem option="uniform"/>
      <OptionItem option="uniformby"/>
    </OptionChoice>

    <OptionItem option="SPLITLABEL"/>
      <OptionChoice option="SPLITLABELVALUE"/>
        <OptionValue>*</OptionValue>
        <OptionValue>!</OptionValue>
        <OptionValue>@</OptionValue>
        <OptionValue>#</OptionValue>
        <OptionValue>$</OptionValue>
        <OptionValue>%</OptionValue>
        <OptionValue>^</OptionValue>
        <OptionValue>amp</OptionValue>
        <OptionValue>+</OptionValue>
      </OptionChoice>
    </Group>
  </Container>
</UI>
4 <Dependencies>
  <Dependency condition="OBS=='1'">
    <Target action="enable" conditionResult="true" option="OBSHEADING"/>
    <Target action="disable" conditionResult="false" option="OBSHEADING"/>
  </Dependency>
  <Dependency condition="$SPLITLABEL=='1'">
    <Target action="enable" conditionResult="true" option="SPLITLABELVALUE"/>
    <Target action="disable" conditionResult="false" option="SPLITLABELVALUE"/>
  </Dependency>
  <Dependency condition="$ROWS2LIST.equalsIgnoreCase('firstnrows')">
    <Target action="enable" conditionResult="true" option="NVALUE"/>
    <Target action="disable" conditionResult="false" option="NVALUE"/>
  </Dependency>
</Dependencies>
5 <CodeTemplate>
  <![CDATA[

```

```

#set( $TABLE = $DATASOURCE )
title;
footnote;

title1 "List Data for $DATASOURCE";

#if( $BY.size()>0 ) /*Sort $DATASOURCE for BY group processing. */

proc sort data=$DATASOURCE out=WORK.SORTTEMP;
    by #foreach($item in $BY) $item#end;
run;
#set( $TABLE = "WORK.SORTTEMP" )
#end

/* Print the table      */

proc print data=$TABLE
#if($ROWS2LIST.equalsIgnoreCase("firstnrows"))
    (obs=$NVALUE)
#end
#if ($OBS == '1')
    obs="$OBSHEADING"
#else
    noobs
#end
#if ($HEADING && ($HEADING != "default"))
    heading=$HEADING
#end
#if ($LABEL == '1')
    label
#end
#if ($PRINTNUMROWS == '1')
    n
#end
#if ($ROUND == '1')
    round
#end
#if ($DIVIDE == "0")
    rows=page
#end
#if($WIDTH && ($WIDTH != 'default'))
    width=$WIDTH
#end
#if ($SPLITLABEL == '1')
    split='$SPLITLABELVALUE'
#end

```

```

;
#if( $VAR.size()>0 )
    var #foreach ( $item in $VAR ) $item#end;
#end
#if( $BY.size()>0 )
    by #foreach ( $item in $BY ) $item#end;
#end
#if( $SUM.size()>0 )
    sum #foreach ( $item in $SUM ) $item#end;
#end
#if( $ID.size()>0 )
    id #foreach ( $item in $ID ) $item#end;
#end
run;
quit;

title;
footnote;
]]>
</CodeTemplate>
</Task>

```

Use this list to identify each section in the preceding code:

- 1 The `Registration` element. For more information, see [“Working with the Registration Element” on page 7](#).
- 2 The `Metadata` element specifies whether an input data source is required, defines the roles for columns in the input data source and any options that are required for the task to run. For more information, see [“Working with the Metadata Element” on page 9](#).
- 3 The `UI` element specifies the layout of the task in the user interface. For more information, see [“Working with the UI Element” on page 27](#).
- 4 The `Dependencies` element specifies the options that rely on one another in order for the task to work. For more information, see [“Working with the Dependencies Element” on page 33](#).
- 5 The `Code Template` element generates the SAS code for the task. For more information, see [“Understanding the Code Template” on page 49](#).