**Contents**

*Using This Book* ................................................................. vii
*Accessibility* ........................................................................ ix

**Chapter 1 • Introduction to the Common Task Model** ................................................. 1
- About the SAS Studio Tasks ...................................................... 1
- Using Sample Task Templates .................................................. 3
- Create a New Task ................................................................. 5
- Create a Task with Default Option Settings ................................. 7
- Validation Steps for the Task .................................................... 8
- Testing a Task ........................................................................ 8
- Sharing Tasks ........................................................................ 8

**Chapter 2 • Working with the Registration Element** .................................................... 11
- About the Registration Element ............................................... 11
- Example: The Registration Element from the Sample Task Template .................. 12

**Chapter 3 • Working with the Metadata Element** ...................................................... 13
- About the Metadata Element .................................................... 13
- Working with the DataSources Element ....................................... 14
- Working with the Options Element ............................................ 17

**Chapter 4 • Working with the UI Element** ................................................................. 53
- About the UI Element ............................................................. 53
- Example: UI Element for the Sample Task Template .................... 55

**Chapter 5 • Working with the Dependencies Element** .............................................. 59
- About the Dependencies Element ............................................. 59
- Notes on Dependencies .......................................................... 62
- Example 1: Selecting a Check Box Enables Text Boxes .................. 64
- Example 2: Selecting a Check Box Enables a Combination Box ....... 65
Example 3: Selecting a Radio Button Enables a Number Stepper Control ........................................ 67
Example 4: The Selected Value for a Combination Box Enables the Text Box ........................................ 68
Example 5: Selecting a Check Box Enables Multiple Types of Options ........................................ 70
Example 6: Compound Condition Using AND and OR Logic .................................................. 71

Chapter 6 • Working with the Requirements Element ................................................................. 75
About the Requirements Element .......................................................................................... 75
Example: Using a Requirements Element for Roles .......................................................... 76

Chapter 7 • Understanding the Code Template ........................................................................... 77
About the Code Template ......................................................................................................... 78
Using Predefined Velocity Variables ..................................................................................... 78
Predefined SAS Macros ........................................................................................................... 79
How the DataSource Element Appears in the Velocity Code .............................................. 80
How the Roles Elements Appear in the Velocity Code ......................................................... 80
How the Options Elements Appear in the Velocity Code ...................................................... 82

Chapter 8 • Example: Task Definition for List Data Task ............................................................... 93
Open the List Data Task ........................................................................................................... 93
View the XML Code for the List Data Task ............................................................................ 94
Understanding the XML Code for the List Data Task .......................................................... 95

Appendix 1 • Common Utilities for CTM Writers ......................................................................... 103
About the Predefined $CTMUtil Variable .............................................................................. 103
quotestring Method ................................................................................................................ 103
toSASName Method ................................................................................................................ 104

Recommended Reading ........................................................................................................ 105
Audience

*SAS Studio: Developer's Guide to Writing Custom Tasks* is intended for developers who need to create 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.
Using This Book
For information about the accessibility of this product, see Accessibility Features of SAS Studio 3.3 at support.sas.com.
Using This Book
Introduction to the Common Task Model

About the SAS Studio Tasks ........................................................................................................... 1

Using Sample Task Templates ........................................................................................................... 3
  What Is the Difference between the Sample Task Template and the Advanced Task Template? ............. 3
  View the Sample Task Template .................................................................................................... 3
  View the Advanced Task Template ................................................................................................ 4

Create a New Task ............................................................................................................................. 5

Create a Task with Default Option Settings ....................................................................................... 7

Validation Steps for the Task ............................................................................................................... 8

Testing a Task ..................................................................................................................................... 8

Sharing Tasks ..................................................................................................................................... 8
  About CTM and CTK Files ................................................................................................................. 8
  Accessing a Task Created by Another User ..................................................................................... 9
  Sharing a Task That You Created .................................................................................................... 9

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 the template for the task. In the CTM file, you define how the task appears to the SAS Studio user and specify the code that is needed 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 can specify whether an input data source is required to run the task, any role assignments, and the options in the task.

- The Roles element specifies the types of variables that 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 Options element specifies 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.

Using Sample Task Templates

What Is the Difference between the Sample Task Template and the Advanced Task Template?
The sample task template shows the controls that are available to you when writing a task. The advanced task template shows some of the more complex functionality in the common task model. For example, the advanced template includes dependencies, the model effects builder, data linking, and return values.

View the Sample Task Template
To view the sample task template:

1. In the navigation pane, open the Tasks section.
2. Click , and select Sample Task Template.
The sample task template that is shipped with SAS Studio appears.

View the Advanced Task Template

To view the advanced task template:

1. In the navigation pane, open the Tasks section.

2. Click , and select Advanced Task Template.
The advanced task template that is shipped with SAS Studio appears.

Create a New Task

A blank template is available to help you create a new task.
To create a new task:

1. In the navigation pane, open the **Tasks** section.

2. Click ![new-task-button](image), and select **New Task Template**.
   
   The new task template appears in SAS Studio.

3. Use the blank template to create your task. For help with the Velocity Template Language, see *Apache Velocity User’s Guide*.

4. To save the task, click ![save-task-button](image).
5 Enter a unique name for the task. The task is saved with the CTM file extension in your file system.

Create a Task with Default Option Settings

When you develop a task, you might want to include a default input data source or default option settings for the users at your site. In SAS Studio, you can save a task as a CTK file. When users at your site run this CTK file, they see your default settings.

Note: Before you can save a task, you must specify an input data set and all the options that are required to run the task.

To save a task:

1 Click 📛. The Save As window appears.

2 Select the location where you want to save the task file. You can save this file in the Folders section or in your My Tasks folder. Specify a name for this file. For the file type, select CTK Files (*.CTK). Click Save.
Note: In the Tasks section, you are still working with this task. If you save the task again, the CTK file in the Folders section is updated.

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.) A new tab that contains the user interface for the task appears in your work area. To view the SAS code for this task, click Code. The CTM code is still available from the original tab within the task.

Sharing Tasks

About CTM and CTK Files

After creating a task, you might want to share it with other users at your site. Tasks can be saved as CTM files or CTK files. A CTM file contains the XML and Velocity code for the task. To create a CTK file, a user opens the CTM file, sets several roles or options in the task user interface, and then saves the task. For more information about how to create a CTK file, see “Create a Task with Default Option Settings” on page 7.

You can share CTM and CTK files by attaching these files to an e-mail or saving these files in a network location.
Accessing a Task Created by Another User

To access a task that is created by another user in 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 UPLOAD. 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, open the Folders section. Click 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.
Working with the Registration Element

About the Registration Element

The Registration element represents a collection of metadata for the task. This element is required in order to know the type of task.

Here are the child elements for the Registration element:

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the task. This name is used throughout the application to represent the task.</td>
</tr>
<tr>
<td>Description</td>
<td>A description of the task. This text could appear in the task properties or in tooltips for the task.</td>
</tr>
<tr>
<td>GUID</td>
<td>A unique identifier for the task.</td>
</tr>
<tr>
<td>Procedures</td>
<td>A list of SAS procedures that are used by this task.</td>
</tr>
<tr>
<td>Version</td>
<td>A simple integer value that represents the version of the task.</td>
</tr>
<tr>
<td>Element Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Links</td>
<td>A list of hyperlinks to help or resources related to this task. <strong>Note:</strong> If you do not have any resources to link to, this element is optional.</td>
</tr>
</tbody>
</table>

**Example: The Registration Element from the Sample Task Template**

Here is the Registration element from the sample task template:

```xml
<Registration>
  <Name>Task Template</Name>
  <Description>Demonstrates the Common Task Model functionality.</Description>
  <GUID>C6AC34BD-D14A-4CF5-BF2F-A110711BF819</GUID>
  <Procedures>PRINT</Procedures>
  <Version>3.3</Version>
  <Links>
    <Link href="http://www.sas.com">SAS Home page</Link>
  </Links>
</Registration>
```
Working with the Metadata Element

About the Metadata Element ........................................... 13

Working with the DataSources Element .............................. 14
  About the DataSources Element .................................... 14
  Working with the Roles Element .................................... 14

Working with the Options Element .................................. 17
  About the Options Element .......................................... 17
  Supported Input Types ................................................... 19
  Specifying a Return Value Using the returnValue Attribute .... 45
  Populating the Values for a Select Control from a Source Control ......................................................... 46

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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>specifies the name assigned to this role.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| **type**  | specifies the type of column that can be assigned to this role. Here are the valid values:  
| A         | All column types are allowed. In the user interface, all columns are identified by the 📊 icon.  
| N         | Only numeric columns can be assigned to this role. In the user interface, numeric columns are identified by the 📊 icon.  
| C         | Only character columns can be assigned to this role. In the user interface, character columns are identified by the 📊 icon.  |

| **minVars** | specifies the minimum number of columns that must be assigned to this role. If minVars="0", the role is optional. If minVars="1", a column is required to run this task, and a red asterisk appears next to the label in the user interface. |

| **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 Template**

Here is an example of the DataSources and Roles elements from the sample task template:

```xml
<DataSources>
  <DataSource name="DATASOURCE">
```


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

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

A red asterisk appears for the **Required variable** 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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>specifies the name assigned to this option.</td>
</tr>
<tr>
<td>defaultValue</td>
<td>specifies the initial value for the option.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>inputType</td>
<td>specifies the input control for this option. Here are the valid values:</td>
</tr>
<tr>
<td></td>
<td>- checkbox</td>
</tr>
<tr>
<td></td>
<td>- color</td>
</tr>
<tr>
<td></td>
<td>- combobox</td>
</tr>
<tr>
<td></td>
<td>- datepicker</td>
</tr>
<tr>
<td></td>
<td>- distinct</td>
</tr>
<tr>
<td></td>
<td>- dualselector</td>
</tr>
<tr>
<td></td>
<td>- inputtext</td>
</tr>
<tr>
<td></td>
<td>- modelbuilder</td>
</tr>
<tr>
<td></td>
<td>- multientry</td>
</tr>
<tr>
<td></td>
<td>- numstepper</td>
</tr>
<tr>
<td></td>
<td>- numbertext</td>
</tr>
<tr>
<td></td>
<td>- outputdata</td>
</tr>
<tr>
<td></td>
<td>- radio</td>
</tr>
<tr>
<td></td>
<td>- select</td>
</tr>
<tr>
<td></td>
<td>- slider</td>
</tr>
<tr>
<td></td>
<td>- string</td>
</tr>
<tr>
<td></td>
<td>- textbox</td>
</tr>
<tr>
<td></td>
<td>- validationtext</td>
</tr>
<tr>
<td></td>
<td>For more information, see “Supported Input Types” on page 19.</td>
</tr>
<tr>
<td>indent</td>
<td>specifies the indentation for this option in the task interface. Here are</td>
</tr>
<tr>
<td></td>
<td>the valid values:</td>
</tr>
<tr>
<td></td>
<td>- 1  – minimal indentation (about 17px)</td>
</tr>
<tr>
<td></td>
<td>- 2  – average indentation (about 34px)</td>
</tr>
<tr>
<td></td>
<td>- 3  – maximum indentation (about 51px)</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>returnValue</td>
<td>applies to strings that are used by input types (such as combobox and select) where the user has a selection of choices. If the returnValue attribute is specified in other contexts, this attribute is ignored. For more information, see “Specifying a Return Value Using the returnValue Attribute” on page 45.</td>
</tr>
</tbody>
</table>

**Supported Input Types**

**checkbox**

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

Here is the example code in the sample task template:

```xml
<Option name="GROUPCHECK" inputType="string">CHECK BOX</Option>
<Option name="labelCheck" inputType="string">
  An example of a check box. Check boxes are either on or off.
</Option>
<Option name="chkEXAMPLE" defaultValue="0" inputType="checkbox">
  Check box
</Option>
```

Here is an example of a check box control in the user interface:

▲ CHECK BOX

An example of a check box. Check boxes are either on or off.

☐ Check box
color

This input type has one attribute.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
</table>
| required  | specifies whether a value is required. Valid values are true and false. The default value is false.  

**Note:** If the `required` attribute is set to `true` and no default value is specified, the user must select a color to run the task.

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

```xml
<Option name="GROUPCOLOR" inputType="string">COLOR SELECTOR</Option>  
<Option name="labelCOLOR" inputType="string">An example of a color selector.</Option>  
<Option name="colorEXAMPLE" defaultValue="red" inputType="color">  
    Choose a color
</Option>
```

Here is an example of a color control in the user interface:

![COLOR SELECTOR](image)

An example of a color selector.

Choose a color

combobox

This input type has these attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
</table>
| required  | specifies whether a value is required. Valid values are true and false. The default value is false.  

**Note:** If the `required` attribute is set to `true` and no default value is specified, the combobox control displays the text specified in the `selectMessage` attribute.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>selectMessage</td>
<td>specifies the message to display when a value is required for the combobox control and no default value has been set. The default message is Select a value.</td>
</tr>
<tr>
<td>width</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

```xml
<Option name="GROUPCOMBO" inputType="string">COMBOBOX</Option>
<Option name="labelCOMBO" inputType="string">An example of a combobox.</Option>
<Option name="comboEXAMPLE" defaultValue="value2" inputType="combobox" width="100%">Combobox:</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>
```

Here is an example of a combobox control in the user interface:
datepicker

This input type has these attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>specifies the format of the date value. You can use any valid SAS date format. If no format attribute is provided, it defaults to mmddyyys8. (12/24/93).</td>
</tr>
<tr>
<td>required</td>
<td>specifies whether a date is required. By default, no date is required.</td>
</tr>
<tr>
<td>width</td>
<td>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.</td>
</tr>
</tbody>
</table>

If you specify the `defaultValue` attribute for this input type, the value must be in ISO8601 format (yyyy-mm-dd).

The code in the sample task template creates datepicker control with the label Choose a date:.

```html
<Option name="GROUPDATE" inputType="string">DATE PICKER</Option>
<Option name="labelDATE" inputType="string">An example of a date picker.</Option>
<Option name="dateEXAMPLE" inputType="datepicker"
    format="monyy7.">Choose a date:</Option>
```
Here is an example of a datepicker control in the user interface:

![DATE PICKER](image)

**distinct**
This input type has these attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>required</td>
<td>specifies whether a value is required. The default value is false. <strong>Note:</strong> If the <code>required</code> attribute is set to true and no default value is specified, the combobox control displays the text specified in the <code>selectMessage</code> attribute.</td>
</tr>
<tr>
<td>selectMessage</td>
<td>specifies the message to display when a value is required for the combobox control and no default value has been set. The default message is Select a value.</td>
</tr>
<tr>
<td>source</td>
<td>specifies the role to use to get the distinct values. The <code>maxVars</code> control for the role must be set to 1. In other words, users can assign only one variable to this role.</td>
</tr>
</tbody>
</table>
### Attribute | Description
--- | ---
**max** | specifies the maximum number of distinct values to obtain and display in the UI. By default, the maximum value is 100. Larger maximum values might cause a long delay in populating the UI control.  
**Note:** Missing values are ignored, so missing values do not appear in the list of distinct values.

**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.

In this example, you want the user of this task to see the first 15 distinct values for the response variable.

In the code, you first specify the `Datasources` element because an input data set is required to run this task. Then in the `Roles` element, you specify that only one response variable is required to run this task. The `name` attribute for this role is **VAR**.

Now, you want to create an option that lists the first 15 distinct values in the **VAR** variable. The code for the distinct input type includes these attributes.

- The `source` attribute specifies that the values that appear in the **Age of interest** option come from the **VAR** role (in this example, the Age variable).
- The `max` attribute specifies that a maximum of 15 values should be available for the **Age of interest** option.

```xml
<DataSources>
  <DataSource name="DATASOURCE">
    <Roles>
      <Role type="A" maxVars="1" order="true" minVars="1" name="VAR">Response variable</Role>
    </Roles>
  </DataSource>
</DataSources>
.Options>
  <Option name="values" inputType="distinct" source="VAR" max="15">Age of interest:</Option>
</Options>
```
Here is an example of the distinct control in the user interface:

![Age of interest:](image)

**dualselector**

This input type has these attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>height</strong></td>
<td>specifies the height of the control. This value can be in em or px. If a height is not specified, SAS Studio sizes the control based on a reasonable default.</td>
</tr>
<tr>
<td><strong>required</strong></td>
<td>specifies whether any input text is required. Valid values are true and false. The default value is false.</td>
</tr>
<tr>
<td><strong>width</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

You can specify default values for the dualselector control by using the **defaultValnue** attribute. Any default values that you specify are selected at run time. If you need to
specify multiple default values, use a comma-separated list of values for the default value attribute.

This example shows how the dualselector control works.

```<Options>
  <Option name="ANOTHERLIST" inputType="dualselector"
    defaultValue="anothertest2, anothertest3">Test choices:</Option>
  <Option inputType="string" name="anothertest1">Another 1</Option>
  <Option inputType="string" name="anothertest2">Another 2</Option>
  <Option inputType="string" name="anothertest3">Another 3</Option>
  <Option inputType="string" name="anothertest4">Another 4</Option>
  <Option inputType="string" name="anothertest5">Another 5</Option>
  <Option inputType="string" name="anothertest6">Another 6</Option>
</Options>

<UI>
  <OptionChoice option="ANOTHERLIST">
    <OptionItem option="anothertest1"/>
    <OptionItem option="anothertest2"/>
    <OptionItem option="anothertest3"/>
    <OptionItem option="anothertest4"/>
    <OptionItem option="anothertest5"/>
    <OptionItem option="anothertest6"/>
  </OptionChoice>
</UI>
```

When you run this code, the Test choices option appears in the user interface. In this example, the default value attribute specifies to use the values for anothertest2 and anothertest3 as the default values for this option. As a result, Another 2 and Another 3 are automatically selected for the Test choices option.
To change the selected values, click **Edit**. A new dialog box appears. From this dialog box, the user can see a list of all the available variables and then select which variables to use for the **Test choices** option.

When the user clicks **OK**, any variables in the **Selected** pane now appear in the list of values for the **Test choices** option. To specify the order of the values in the **Test choices** option, use the up and down arrows for the **Selected** pane.

**inputtext**

This input type has these attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>required</td>
<td>specifies whether any input text is required. Valid values are true and false. The default is false.</td>
</tr>
<tr>
<td>missingMessage</td>
<td>specifies the tooltip text that appears when the text box is empty but input text is required. No message is displayed by default.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>promptMessage</td>
<td>specifies the tooltip text that appears when the text box is empty and the user has selected the text box.</td>
</tr>
<tr>
<td>width</td>
<td>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.</td>
</tr>
</tbody>
</table>

The code in the sample task template creates a text box called **Input text**. The default value is “Text goes here.” If the user removes this text, the message “Enter some text” appears because a value is required.

```xml
<Option name="textEXAMPLE" defaultValue="Text goes here" inputType="inputtext"
   indent="1"
   required="true"
   promptMessage="Enter some text."
   missingMessage="Missing text.">Input text:</Option>
```

Here is an example of an inputtext control in the user interface:

An example of an input text. This text field is required.

*Input text:*

![Input text](image)

**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.

**nested effect**
For variables Gender and Height, an example of a nested effect is Gender(Height).

**polynomial effect**
You can create polynomial effects with continuous variables. For the continuous variable X, the quadratic polynomial effect is $X^2$. You can have second-order, third-order, ... $n$th-order polynomial effects.

The **modelbuilder** input type has these attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>required</td>
<td>specifies whether any input text is required. Valid values are true and false. The default is false.</td>
</tr>
<tr>
<td>roleContinuous</td>
<td>specifies the role that contains the continuous variables. The default value is null.</td>
</tr>
<tr>
<td>roleClassification</td>
<td>specifies the role that contains the classification variables. The default value is null.</td>
</tr>
<tr>
<td>excludeTools</td>
<td>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.</td>
</tr>
<tr>
<td>width</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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 **modelbuilder** input type from the Generalized Linear Model task:

```xml
<Option excludeTools="THREEFACT,NFACTPOLY" inputType="modelbuilder"/>
```
Here is an example of a modelbuilder control in the user interface:

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, Horsepower, and MPG_City are the continuous variables.
**multientry**

This input type has these attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>required</td>
<td>specifies whether a value is required. Valid values are <code>true</code> and <code>false</code>. The default value is <code>false</code>.</td>
</tr>
<tr>
<td>width</td>
<td>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.</td>
</tr>
<tr>
<td>reorderable</td>
<td>specifies whether the user can reorder the values in the list. Valid values are <code>true</code> and <code>false</code>. The default value is <code>false</code>.</td>
</tr>
</tbody>
</table>
The code in the sample task template creates the **Multiple entry** option.

```xml
<Options>
  <Option name="labelMULTIENTRY" inputType="string">An example of a multiple entry. This control allows the user to add their own values to create a list.</Option>
  <Option name="multientryEXAMPLE" inputType="multientry">Multiple entry:</Option>
</Options>

<UI>
  ...
  <OptionItem option="labelMULTIENTRY" />
  <OptionChoice option="multientryEXAMPLE">
    <OptionItem option="value1" />
    <OptionItem option="value2" />
    <OptionItem option="value3" />
  </OptionChoice>
  ...
</UI>
```

In this example, the **Multiple entry** option has 3 values: Value 1, Value 2, and Value 3. To add additional values to the list, enter the name of the new value in the text box and click +.

An example of a multiple entry. This control allows the user to add their own values to create a list.

Multiple entry:

```
Value 1
Value 2
Value 3
```

To enable users to reorder the values in this list, set the `reorderable` attribute to `true`, as shown in this example.

```xml
<Options>
  <Option name="labelMULTIENTRY" inputType="string">An example of a multiple entry. This control allows the user to add their own values to create a list.</Option>
  <Option name="multientryEXAMPLE" inputType="multientry" reorderable="true">Multiple entry:</Option>
</Options>
```
Now, the multientry control includes up and down arrows.

Multiple entry:

Value 1
Value 2
Value 3

**numbertext**

This input type has these attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>decimalPlaces</td>
<td>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=&quot;0,3&quot;. The maximum number of decimal places is 15.</td>
</tr>
<tr>
<td>invalidMessage</td>
<td>specifies the tooltip text that appears when the content is invalid.</td>
</tr>
<tr>
<td>maxValue</td>
<td>specifies the maximum value that is allowed. If the user tries to exceed this value, a message appears. The default value is 9000000000000.</td>
</tr>
</tbody>
</table>
### Attribute Description

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>minValue</td>
<td>specifies the minimum value that is allowed. If the user specifies a value that is below the minimum value, a message appears.</td>
</tr>
<tr>
<td>missingMessage</td>
<td>specifies the tooltip text that appears when the text box is empty, but a value is required.</td>
</tr>
<tr>
<td>promptMessage</td>
<td>specifies the tooltip text that appears when the text box is empty, and the field has focus.</td>
</tr>
<tr>
<td>rangeMessage</td>
<td>specifies the tooltip text that appears when the value in the text box is outside the specified range.</td>
</tr>
<tr>
<td>required</td>
<td>specifies whether a value is required. Valid values are true and false. The default value is false.</td>
</tr>
<tr>
<td>width</td>
<td>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.</td>
</tr>
</tbody>
</table>

This example code creates a field called **Number to order**.

```xml
<Option name="labelNUMBERTEXT" inputType="string">An example of a number text. The minimum value is set to 0 and the maximum value is set to 100.
<inputType="numbertext" minValue="0" maxValue="100"
promptMessage="Enter a number between 0 and 100."
invalidMessage="This number is out of range. Enter a number between 0 and 100.">
Number text:</Option>
```

Here is an example of the numbertext control in the user interface:
According to the code, the minimum value for this field is 0, and the maximum value is 100. Because 110 exceeds the maximum value, the default out of range message appears.

**numstepper**

This input type has these attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>decimalPlaces</td>
<td>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=&quot;0,3&quot;.</td>
</tr>
<tr>
<td>increment</td>
<td>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.</td>
</tr>
<tr>
<td>maxValue</td>
<td>specifies the maximum value that is allowed. If the user tries to exceed this value, a message appears. The default value is 9000000000000.</td>
</tr>
<tr>
<td>minValue</td>
<td>specifies the minimum value that is allowed. If the user specifies a value that is below the minimum value, a message appears.</td>
</tr>
<tr>
<td>required</td>
<td>specifies whether a value is required. Valid values are true and false. The default value is false.</td>
</tr>
<tr>
<td>width</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

```xml
<Option name="labelNumStepperEXAMPLE1" inputType="string">
    An example of a basic numeric stepper.</Option>
<Option name="basicStepperEXAMPLE" defaultValue="5" inputType="numstepper" indent="1">
    Basic numeric stepper:</Option>
```
Here is an example of a numstepper control in the user interface:

```
An example of a basic numeric stepper.

Basic numeric stepper:
```

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

```
<Option name="labelNumStepperEXAMPLE2" inputType="string">
  An example of a 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"
  width="8em"
  indent="1">
  Advanced numeric stepper:
</Option>
```

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

```
An example of a numeric stepper with a minimum value of -10, a maximum value of 120, and an increment of 2.

Advanced numeric stepper:
```

### outputdata

The `outputdata` input type creates a text box where the user can specify the name of the output data set that is created by a task.

This input type has these attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>required</td>
<td>specifies whether a name is required. The default value for this attribute is <code>false</code>, which means that no name is required.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>width</td>
<td>specifies the width of the control. The width can be specified in (percent) %, em, or px. By default, SAS Studio determines the size of the control based on the available width and content.</td>
</tr>
</tbody>
</table>

Here are the two types of valid values for this control:

- a single-level name in the format `data-set-name`
- a two-level name in the format `library-name.data-set-name`

These names must follow SAS naming conventions. For more information, see “Names in the SAS Language” in *SAS Language Reference: Concepts*.

**Note:** If you specify a single-level member name, the library is determined by the application where you are running the task (such as SAS Studio, SAS Enterprise Guide, or the SAS Add-In for Microsoft Office) or by the SAS Server. To increase the flexibility in initializing the task, use a single-level data set name for the `defaultValue` attribute.

If you use the `defaultValue` attribute, SAS Studio checks to see whether this name is unique when you open the task. If the name is unique, the outputdata control in the task uses the default name specified. If the name is not unique, a suffix (starting with 0001) is added to the default name.

In this code example, the `defaultValue` attribute is `Outputds`. If no existing data sets use this name, `Outputds` appears as the name in the outputdata control. If an `Outputds` data set already exists, SAS Studio uses the suffix to create a unique name, such as `Outputds0001`. Using this technique prevents SAS Studio from overwriting an existing data set.

```xml
<Option defaultValue="Outputds" indent="1" inputType="outputdata" name="outputDSName" required="true">Data set name:</Option>
```

Here is an example of the outputdata control from the Summary Statistics task:
radio

This input type has one attribute:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable</td>
<td>specifies a variable that contains the name of the currently selected radio button.</td>
</tr>
</tbody>
</table>

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

```xml
<Options>
  <Option name="labelRADIO" inputType="string">An example of radio buttons. One radio button can be selected at a time.</Option>
  <Option name="radioButton1" variable="radioEXAMPLE" defaultValue="1" 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>

  ...
</Options>
```
Here is how this radio control appears in the user interface:

![An example of radio buttons. One radio button can be selected at a time.](image)

- Radio button 1
- Radio button 2
- Radio button 3

**select**

This input type has these attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiple</td>
<td>specifies whether users can select one or multiple items from the list. Valid values are true and false. The default value is true.</td>
</tr>
<tr>
<td>required</td>
<td>specifies whether the user must select a value from the list. Valid values are true and false. The default value is false.</td>
</tr>
<tr>
<td>sourceLink</td>
<td>specifies that the data for this control should come from another option. For more information about this attribute, see “Populating the Values for a Select Control from a Source Control” on page 46.</td>
</tr>
<tr>
<td>width</td>
<td>specifies the width of the control in percent (%), em, or px.</td>
</tr>
<tr>
<td>height</td>
<td>specifies the height of the control in em or px.</td>
</tr>
</tbody>
</table>

The sample task template creates an option called **Select**.

```xml
<Option name="labelSELECT" inputType="string">An example of a select. This example is set up for multiple selection.</Option>
<Option name="selectEXAMPLE" inputType="select" multiple="true">Select:</Option>
```

```xml
<UI>
...
<OptionItem option="labelSELECT"/>
```
This example creates a selection list called **Subjects of interest** and has three choices: Biology, Chemistry, and Physics. The `defaultValue` attribute specifies the item or items that should be selected by default. Multiple items are in a comma-separated list. In this example, item1 (Biology) and item2 (Chemistry) are selected by default.

Here is an example of the select control in the user interface:
**slider**

This input type has these attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>discreteValues</td>
<td>specifies the number of discrete values in the slider. For example, if discreteValues=&quot;3&quot;, the slider has three values: a minimum value, a maximum value, and a value in the middle.</td>
</tr>
<tr>
<td>maxValue</td>
<td>specifies the maximum value for this option.</td>
</tr>
<tr>
<td>minValue</td>
<td>specifies the minimum value for this option.</td>
</tr>
<tr>
<td>showButtons</td>
<td>specifies whether to show the increase and decrease buttons for the slide. Valid values are true and false. The default value is true.</td>
</tr>
</tbody>
</table>

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

```xml
<Option name="labelSliderEXAMPLE1" inputType="string">
  An example of a slide with buttons.</Option>
<Option name="labelSliderEXAMPLE1" defaultValue="80.00" inputType="slider" discreteValues="14" minValue="-10" maxValue="120">Slider with buttons</Option>
```

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

![Slider with buttons](image)

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

```xml
<Option name="labelSliderEXAMPLE2" inputType="string">An example of a slider without buttons.</Option>
<Option name="labelSliderEXAMPLE2" defaultValue="80.00" inputType="slider" discreteValues="14" minValue="-10" maxValue="120" showButtons="false">Slider without buttons</Option>
```
When you run the code, here is the resulting user interface:

```
An example of a slider without buttons.
Slider without buttons
```

### string

The **string** input type can be used to display informational text to the user, to define strings for the **OptionChoice** tags, and to define string values that are used by the Velocity code.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>returnValue</td>
<td>is the string that is returned in the control's Velocity variable (instead of the control's name). This attribute applies only when the string is used in an <strong>OptionChoice</strong> tag.</td>
</tr>
</tbody>
</table>

The code for the sample task template contains several examples of the string input type. In the code for the slider option, the explanatory text (**An example of a slider with buttons.**) is created by the string input type.

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

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

```
An example of a slider with buttons.
Slider with buttons
```
**textbox**

The `textbox` input type enables the user to enter multiple lines of text. This input type has these attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>required</code></td>
<td>specifies whether any input text is required. Valid values are <code>true</code> and <code>false</code>. The default is <code>false</code>.</td>
</tr>
<tr>
<td><code>width</code></td>
<td>specifies the width of the control. This value can be in percent (%), <code>em</code>, or <code>px</code>. By default, SAS Studio sizes the control based on the available width and content.</td>
</tr>
<tr>
<td><code>height</code></td>
<td>specifies the height of the control. This value can be in <code>em</code> or <code>px</code>. By default, SAS Studio sizes the control based on the available height and content.</td>
</tr>
</tbody>
</table>

If you specify the `defaultValue` attribute with this input type, you can specify the initial string to display in the text box. In this example, the text ‘Enter text here’ appears in the text box by default. Note the use of single quotation marks around the text. This example shows how you would include single quotation marks in your default text. These quotation marks are not required.

```
<Option name="textSimple" required="true" inputType="textbox"
       defaultValue="'Enter text here'">Text Box</Option>
```

Here is an example of a textbox control in the user interface. Note this example uses the default text. When the user types in the textbox control, this text disappears.

**Comments:**

'Enter text here.'
**validationtext**

This input type has these attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>required</td>
<td>specifies whether any input text is required. Valid values are true and false. The default value is false.</td>
</tr>
<tr>
<td>invalidMessage</td>
<td>specifies the tooltip text to display when the content in the text box is invalid. By default, no message is displayed.</td>
</tr>
<tr>
<td>missingMessage</td>
<td>specifies the tooltip text that appears when the text box is empty but text is required. By default, no message is displayed.</td>
</tr>
<tr>
<td>promptMessage</td>
<td>specifies the tooltip text that appears when the text box is empty and the text box is selected. By default, no message is displayed.</td>
</tr>
<tr>
<td>regExp</td>
<td>specifies the regular expression pattern to use for validation. This syntax comes directly from JavaScript Regular Expressions.</td>
</tr>
<tr>
<td>width</td>
<td>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.</td>
</tr>
</tbody>
</table>

The code for the sample task template creates a text box called **Validation text**:

```xml
<Option name="labelVALIDATIONTEXT" inputType="string">An example of a validation text. A regular expression of 5 characters has been applied.</Option>
<Option name="validationTextExample" defaultValue="99999" inputType="validationtext" promptMsg="Enter a string 5 characters long." invalidMsg="More than 5 characters have been entered." regExp="\d\{5\}">Validation text:
</Option>
```

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

*An example of a validation text. A regular expression of 5 characters has been applied.*

**Validation text:**

99999
If you remove the default value from this box, the message appears: Enter a string 5 characters long.

When the user begins entering a value, this message appears: Enter a string 5 characters long.

If the specified value is contains more than 5 characters, the message for an invalid value appears.

Specifying a Return Value Using the returnValue Attribute

For input types (such as combobox and select) that enable users to select from a list of choices, the default behavior is to return the name of the selected item in the list. However, because the name attribute must be unique for every option, this default behavior could be limiting in some scenarios.

When you specify the returnValue attribute on an Option element, the string that is specified for the returnValue attribute is returned instead of the name.

The following example is available from the advanced task template. In this example, the $vegetables Velocity variable has the value of 1, 2, or 3, depending on what option item the user selected in the user interface. If you do not specify the returnValue attribute, the Velocity variable returns carrots, peas, or corn.

```xml
<Options>
   <Option name="RETURNVALUETAB" inputType="string">RETURN VALUE</Option>
   <Option name="labelReturnValue" inputType="string">This tab shows an example of the option's returnValue attribute. This attribute can be used
```
in the OptionChoice controls to customize Velocity return values.

```xml
<Option name="vegetables" inputType="select" multiple="true">Select the vegetables</Option>
<Option name="carrots" returnValue="1" inputType="string">Carrots</Option>
<Option name="peas" returnValue="2" inputType="string">Peas</Option>
<Option name="corn" returnValue="3" inputType="string">Corn</Option>
</Options>

If you run the advanced task template, here is the resulting Return Value tab.

---

**Populating the Values for a Select Control from a Source Control**

**About Data Linking**

Data linking is a way to populate a control based on the contents of another control. Data linking is currently supported when a select control links to data from a role or from
the model effects builder. If the select control links to anywhere else, any children in the OptionChoice tag are ignored.

The select control is the recipient of the data. The control that the select input type links to is called the source. To link a select input type to its source, you define the sourceLink attribute and use the name of the source control.

The Velocity code that is returned for the select control uses the same Velocity structure that you would expect from the source control.

This example is from the advanced task template.

```xml
<Option name="DATALINKINGTAB" inputType="string">DATA LINKING</Option>
<Option name="DATALINKINGTEXT" inputType="string">This tab shows examples of data linking. Data linking allows controls to be populated based on data from another control</Option>
<Option name="ROLELINKING" inputType="string">LINKING TO ROLES</Option>
<Option name="selectRoles" inputType="select" multiple="true" sourceLink="dataVariables">This select is populated from the Variables selected from the Data tab.</Option>
<Option name="MEBLINKING" inputType="string">LINKING TO MODEL EFFECTS BUILDER</Option>
<Option name="selectMEB" inputType="select" multiple="true" sourceLink="modelBuilder">This select is populated from the output of the Model Effects Builder.</Option>

...<UI>
  <OptionItem option="DATALINKINGTAB"/>
  <OptionItem option="DATALINKINGTEXT"/>
  <Group option="ROLELINKING" open="true">
    <OptionChoice option="selectRoles"/>
  </Group>
  <Group option="MEBLINKING" open="true">
    <OptionChoice option="selectMEB"/>
  </Group>
</Container>
...
</UI>
```
If you run the code for the advanced task template, here is the resulting Data Linking tab.

**Linking to a Role**

If a select control is linked to a role, the values in the select control are the current list of roles in the roles option. In this example, the name of the role variable is NUMVAR (specified in the name attribute). In the select control, the sourceLink attribute links to NUMVAR.

```xml
<DataSources>
  <DataSource name="PRIMARYDATA">
    <Roles>
      <!-- Role list content here -->
    </Roles>
  </DataSource>
</DataSources>
```
The Velocity variable that is created for the select control is $roleList. The contents of the $roleList variable mimic the output of a typical role control. For more information, see “How the Roles Elements Appear in the Velocity Code” on page 80.

**Linking to Effects from the Model Builder**

If a select control is linked to a `modelbuilder` input type, the values in the select control are the list of effects in the model effects builder.

An additional attribute called `sourceType` can be used to set a filter on the data that is sent to the select control. Currently, the only defined filter is ‘filterClassification’. When this filter is specified, only classification effects appear in the select control.

In this example, the modelbuilder control is named MEB. In the select control, the `sourceLink` attribute links to MEB, and the `sourceType` attribute specifies the ‘filterClassification’ filter. As a result, only classification effects appear in the source control.

```xml
<Options>
  <Option name="meb" inputType="modelbuilder" roleContinuous="CONTVARS" roleClassification="CLASSVARS"/>
  <Option name="mebList" inputType="select" sourceLink="MEB" sourceType="filterClassification"/>
</Options>
```

The Velocity variable that is created for the select control is $mebList. The contents of the $mebList variable mimic the output of the model effects builder. For more information, see “modelbuilder” on page 86.

Another example is in the Linear Regression task. In this task, the effects listed in the model builder are the options for the **Select the effects to test** option on the **Options** tab.

The **Variables** pane in the model builder lists the variables that the user assigned to either the **Classification variables** role or the **Continuous variables** role. The user
can create main, crossed, nested, and polynomial effects. These effects appear in the **Model effects** pane.

On the **Options** tab, all classification effects are available from the **Select effects to test** option.

Here are the relevant portions of code from the Linear Regression task:

```xml
<Option inputType="string" name="modelGroup">MODEL EFFECTS</Option>
<Option inputType="string" name="modelTab">MODEL</Option>

1<Option inputType="modelbuilder" name="modelBuilder"
   excludeTools="POLYEFFECT,TWOFACT,THREEFACT,NFACTPOLY"
   roleClassification="classVariable"
   roleContinuous="continuousVariables"
   width="100%">Model</Option>

...  
<Option inputType="string" name="multCompareGroup">Multiple Comparisons</Option>
```
Select effects to test</Option>

1. Creates the model builder on the Models tab. Classification variables and continuous variables can be used to create the model effects.

2. Creates the Select effects to test option. The sourceLink attribute specifies that the initial list of values for this option is the list of model effects in the model builder. The sourceType attribute filters the list generated by the sourceLink attribute. The filterClassification filter specifies that only effects that include the classification variable should be available in the Select effects to test option.

In the Perform multiple comparisons option, the initial list of model effects includes region, line, product, region(line), line(product), and cost. However, cost is a continuous variable. When this list is filtered, only the model effects that involve classification variables (region, line, and product) are listed as values for the Select effects to test option.
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:

<table>
<thead>
<tr>
<th>Child</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container</td>
<td>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. 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.</td>
</tr>
<tr>
<td>Child</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Group</td>
<td>A title for a group of options. The UI engine displays these options sequentially. This tag takes these attributes:</td>
</tr>
<tr>
<td></td>
<td>- 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.</td>
</tr>
<tr>
<td></td>
<td>- 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>.</td>
</tr>
<tr>
<td>DataItem</td>
<td>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.</td>
</tr>
<tr>
<td>RoleItem</td>
<td>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.</td>
</tr>
<tr>
<td>OptionItem</td>
<td>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.</td>
</tr>
<tr>
<td>OptionChoice</td>
<td>A reference to an option that has a choice of values. The <code>OptionChoice</code> element uses the <code>OptionItem</code> or <code>OptionValue</code> element to represent the choice of values.</td>
</tr>
<tr>
<td></td>
<td>These input types can use the <code>OptionChoice</code> element in the user interface:</td>
</tr>
<tr>
<td></td>
<td>- <code>combobox</code></td>
</tr>
<tr>
<td></td>
<td>- <code>distinct</code></td>
</tr>
<tr>
<td></td>
<td>- <code>dualselector</code></td>
</tr>
<tr>
<td></td>
<td>- <code>multiedit</code></td>
</tr>
<tr>
<td></td>
<td>- <code>select</code></td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Child</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OptionValue</td>
<td>A value choice. This tag is valid only as a child of the OptionChoice element.</td>
</tr>
</tbody>
</table>

### Example: UI Element for the Sample Task Template

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

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

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

    <Group option="GROUPCHECK">
      <OptionItem option="label Check"/>
      <OptionItem option="chk EXAMPLE"/>
    </Group>

    <Group option="GROUPCOLOR">
      <OptionItem option="label COLOR"/>
      <OptionItem option="color EXAMPLE"/>
    </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.

The **Options** tab contains several groups. The previous code creates the Groups, Check Boxes, and Color Selector groups. The first group is expanded by default.
because the open attribute is set to true. (The sample task template includes code to create the remaining groups on the Options tab.)
About the Dependencies Element

The Dependencies element specifies how certain options or roles 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. A dependency can have multiple Target elements.

The Target element has three required attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>references the option that receives the action. Valid values are OptionItem, Role, OptionChoice, or Group element.</td>
</tr>
<tr>
<td>conditionResult</td>
<td>specifies when to execute the action. The valid values for this attribute are true and false.</td>
</tr>
<tr>
<td></td>
<td>- If the condition is true and conditionResult=true, the action is executed.</td>
</tr>
<tr>
<td></td>
<td>- If the condition is false and conditionResult=false, the action is executed.</td>
</tr>
<tr>
<td></td>
<td>- If the value of the condition and conditionResult do not match (for example, one is true and one is false), the action is ignored.</td>
</tr>
</tbody>
</table>
### Attribute | Description
--- | ---
action | specifies the action to execute. Here are the valid values:
- show
- hide
- enable
- disable
- set

If the value of the `action` attribute is `set`, you must also specify these two attributes:

- The `property` attribute refers to the attribute of an element that was created from the metadata. The `option` element in the metadata has an `inputType` attribute that specifies what UI element is created.

  **Note:**
  Here are a few exceptions:
  - In the UI element, any `RoleItem` cannot be the target of a dependency where `action=set`.
  - The `required`, `width`, `indent`, and `variable` (for the radio input type) attributes are invalid values for the `property` attribute of a `Target` element.

- The `value` attribute is the value to use for the target of the `property` attribute.

  If the `value` attribute targets an item with the `select` input type, the `value` attribute can accept a single value or a comma-separated list of values.

  **Note:** If the dependency has a comma-separated list of values and the `select` element that the dependency targets is set to `multiple="false"`, only the first value in the comma-separated list is evaluated. The rest of the values in the list are ignored.
To understand how dependencies work, run the advanced task template. Examples of dependencies are available from the Dependencies tab.

- If action=hide for a Target element, the element is hidden. If action=show, the element is enabled and contributes to 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. All UI elements have a name in the Options element (in the metadata section of the common task model). When a
user selects a UI element, the name is checked against each dependency. Only conditions that contain the name of the UI element are evaluated, and all valid actions are performed.

- Dependencies can have cascading effects.
  - Dependencies that are order dependent cannot be written in a circular manner.
  - 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 the condition and option attributes of the Target element.

This example shows a correct and incorrect ordering of dependencies:

```xml
<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>
  <!-- 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>

  <!-- 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"
```
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.

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.

```xml
<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: WORK.CharacterizeDataFRQ
  - Univariate data: WORK.CharacterizeDataUNI

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: WORK.CharacterizeDataFRQ
  - Univariate data: WORK.CharacterizeDataUNI

---

**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>
```
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:

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

**Quantile method:**

| 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.

```xml
<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')"/>
```
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:

![Options tab with Group = n (NTILES) deselected](image)

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:

![Options tab with Group = n (NTILES) selected](image)

---

**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.

```xml
<Option name="MEMSORT" defaultValue="MSSD" inputType="combobox">"
Memory for sorting:

- Server default
- Maximum available
- B (bytes)
- KB (kilobytes)
- MB (megabytes)
- GB (gigabytes)

The value option is available only if the selected unit is B (bytes), KB (kilobytes), MB (megabytes), or GB (gigabytes).

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:

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

<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>
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.

```xml
<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>
```

```xml
<Dependency condition="(((chkEXACTP=='1') || (chkCHIEXACTP=='1')) && (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*: 0.5
  - *undefined*: 95%
- **Exact test**

### Chi-square goodness-of-fit
- **Asymptotic test**
- **Exact test**
  - *Use Monte Carlo estimation*
    - **Confidence level**: 95%

*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)*: 300

---

*Source: [Chapter 5 / Working with the Dependencies Element](https://example.com)*
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: 0.5
    - undefined 95%
  - **Exact test**
- **Chi-square goodness-of-fit**
  - **Asymptotic test**
  - **Exact test**
    - Use Monte Carlo estimation
    - Confidence level: 95%

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): 300
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. For more information, see the Apache Velocity User’s Guide.

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.

```xml
<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() &gt; 0 || $BYVAR.size() &gt; 0 || $FVAR.size() &gt; 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>
```
Understanding the Code Template

About the Code Template ................................................................. 78

Using Predefined Velocity Variables ............................................. 78
  Predefined Velocity Variables ...................................................... 78
  Floating Point Math ................................................................. 79

Predefined SAS Macros ............................................................... 79

How the DataSource Element Appears in the Velocity Code ............. 80

How the Roles Elements Appear in the Velocity Code ...................... 80

How the Options Elements Appear in the Velocity Code .................. 82
  checkbox ................................................................. 82
  color ................................................................. 82
  combobox ............................................................... 83
  datepicker ............................................................... 84
  distinct ................................................................. 84
  dualselector .............................................................. 85
  inputtext ................................................................. 85
  modelbuilder .............................................................. 86
  multientry ................................................................. 87
  numbertext ................................................................. 88
  numstepper ............................................................... 88
  outputdata ................................................................. 89
  radio ................................................................. 89
  select ................................................................. 90
  slider ................................................................. 91
About the Code Template

The code template creates the string output of the task. For most tasks, this output is SAS code. The Code Template element contains a CDATA block of the Apache Velocity scripting language. The string output is produced using this scripting language.

Using Predefined Velocity Variables

Predefined Velocity Variables

Here are the predefined Velocity variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SASLIBRARY</td>
<td>The name of the library that contains the input data source. The $SASLIBRARY variable is defined only if you have a DataSource element in the task.</td>
</tr>
<tr>
<td>$SASTABLE</td>
<td>The table name of the input data source. The $SASTABLE variable is defined only if you have a DataSource element in the task.</td>
</tr>
<tr>
<td>$sasOS</td>
<td>The operating system for the SAS server.</td>
</tr>
<tr>
<td>$sasVersion</td>
<td>The version of the SAS server.</td>
</tr>
<tr>
<td>$MathTool</td>
<td>The Java object for the Apache Velocity MathTool. For more information, see “Floating Point Math” on page 79.</td>
</tr>
</tbody>
</table>
**Variable** | **Description**
--- | ---
$CTMUtil | This tool holds a Java object that provides common utility methods for the common task models. For more information, see Appendix 1, “Common Utilities for CTM Writers,” on page 103.

**Floating Point Math**

Using the MathTool from Apache Velocity, mathematical expressions can be evaluated in the Velocity context. For example, you can convert a double value to an integer by using the `intValue()` method. For more information, see the MathTool Reference Documentation at http://velocity.apache.org.

This example shows how to use mathematical expressions in the Velocity template. $PCT contains a value between 1 and 100.

```Velocity
<Options>
  <Options name="PCT" defaultValue="10" inputType="inputtext">Value used in the equation</Option>
</Options>
<CodeTemplate>
  <![CDATA[
  #if ($PCT)
  #set ($OUTCALC = 1 - ($MathRool.toDouble($PCT)/100))
  $MathTool.roundTo(2, $OUTCALC)
  $MathTool.toDouble($PCT).intValue()
  #end
]]>
</CodeTemplate>
```

**Predefined SAS Macros**

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

<table>
<thead>
<tr>
<th>SAS Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%web_drop_table(<em>library-name</em>table-name)</td>
<td>drops the specified table. Specifying the library name is optional.</td>
</tr>
</tbody>
</table>
### SAS Macro Description

<table>
<thead>
<tr>
<th>SAS Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%web_open_table(library-name=table-name)</code></td>
<td>opens the specified table. Specifying the library name is optional.</td>
</tr>
<tr>
<td><code>%web_open_file(filename, type)</code></td>
<td>opens the specified file with the specified MIME type.</td>
</tr>
<tr>
<td><code>%web_open_url(url)</code></td>
<td>opens the specified URL.</td>
</tr>
</tbody>
</table>

### How the DataSource Element Appears in the Velocity Code

You can specify a maximum of one `DataSource` element in the common task model. (You can also have a task with no `DataSource` element.) If you define the `DataSource` element, a Velocity variable is created to access the name of the specified data source. The value of the variable is the same as the value of the `name` attribute for the `DataSource` element.

### How the Roles Elements Appear in the Velocity Code

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

You can use the Velocity variable’s GET method to obtain the attributes for each role variable. The GET method takes a string parameter that accepts one of these values:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>specifies the SAS format that is assigned to the variable.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>informat</td>
<td>specifies the SAS informat that is assigned to the variable.</td>
</tr>
<tr>
<td>length</td>
<td>specifies the length that is assigned to the variable.</td>
</tr>
<tr>
<td>type</td>
<td>specifies the type of variable. Valid values are Numeric or Character.</td>
</tr>
</tbody>
</table>

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. To demonstrate the GET method, only numeric variables are added.

```xml
<DataSources>
    <DataSource name="DATASOURCE">
        <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>
    </DataSource>
</DataSources>
<CodeTemplate>
    <![CDATA[
        #if( $BY.size() > 0 )/* Sort $DATASOURCE for BY group processing. */
        PROC SORT DATA=$DATASOURCE OUT=WORK.SORTTEMP;
            BY #foreach($item in $BY ) #if($item.get('type') == 'Numeric' $item #end##end;
            #end
        RUN;]]>
</CodeTemplate>
How the Options Elements Appear in the Velocity Code

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.

**checkbox**

The Velocity variable for the checkbox input type 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.

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

**color**

The Velocity variable for the color input type holds the specified color.

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

```velocity
<Options>
  <Option name="colorEXAMPLE" defaultValue="white" inputType="color">Select a color</Option>
</Options>
<CodeTemplate>
```
The Velocity variable for the combobox input type 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".

```xml
<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>
```

```xml
<UI>
  <Container option="OPTIONSTAB">
    <OptionChoice option="HEADING">
      <OptionItem option="default"/>
      <OptionItem option="horizontal"/>
      <OptionItem option="vertical"/>
    </OptionChoice>
  </Container>
</UI>
```

```xml
<CodeTemplate>
  <![CDATA[
    #if ($HEADING && ($HEADING != "default"))
      HEADING=$HEADING
    #end
  ]]> 
</CodeTemplate>
```
datepicker

The Velocity variable for the `datepicker` input type holds the date that is specified in the datepicker control. By default, this variable is an empty string. If the user selects a date or you specify a default value for the date in the code, the variable holds the specified date. You specify the format of the date by using the `format` attribute.

This example outputs a date if one has been selected. If no date is selected, the “You have not selected a date.” message appears.

```html
<Options>
  <Option name="myDate" inputType="datepicker" format="monyy7.">
    Select a date:
  </Option>
</Options>

<CodeTemplate>
  <![CDATA[
    #if( $myDate == "" )
      You have not selected a date.
    #else
      The date you selected is: $myDate
    #end
  ]]> 
</CodeTemplate>
```

distinct

The Velocity variable for the `distinct` input type holds the information for the distinct control. By default, this variable is the first distinct value in the list.

In this example, the Response variable is Age, and the distinct value is 15. The Velocity script produces the line `Age(event=15)`.

```html
<DataSources>
  <DataSource name="Class">
    <Roles>
      <Role name="responseVariable" type="A" minVars="1" maxVars="1">Response</Role>
    </Roles>
  </DataSource>
</DataSources>

<Options>
  <Option name="referenceLevelCombo" inputType="distinct"
The Velocity variable for the `dualselector` input type holds the array of selected values.

This example is for a `dualselector` control that contains three values: anothertest1, anothertest2, and anothertest3. Any or all of these values can be selected. Only the values that are selected in the `dualselector` control appear in the Velocity code.

```velocity
<OptionChoice name="ANOTHERLIST" inputType="dualselector">
  <OptionItem option="anothertest1"/>
  <OptionItem option="anothertest2"/>
  <OptionItem option="anothertest3"/>
</OptionChoice>
```

The Velocity variable for the `inputtext` input type 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".

```velocity
<Option name="OBSHEADING" indent="1" defaultValue="Row number" inputType="inputtext">Column label:</Option>
```
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>
            <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>
        </DataSource>
    </DataSources>
    <UI>
        <Container option="modelTab">
            <Group open="true" option="modelGroup">
                <OptionItem option="modelBuilder"/>
            </Group>
        </Container>
    </UI>
</Metadata>

<CodeTemplate>
    <![CDATA[
        #macro ( ModelEffects )
        #if ( $modelBuilder )
            #foreach ( $item in $modelBuilder )
                ...
            #end
        #end
    ]]> <![CDATA[
</CodeTemplate>
multientry

The Velocity variable for the multientry input type holds the array of specified values.

In this example, the multientry control contains the values of ONE, TWO, and THREE, so the array contains the values ONE, TWO, and THREE. Users can add new values (such as FOUR). Any new user-specified values are added to the array. In this example if the user specifies FOUR, the array contains the values ONE, TWO, THREE, and FOUR.

```velocity
# 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>
```

```
<UI>
    <Container option="OPTIONSTAB">
        <Group option="GROUP2">
            <OptionChoice name="multiExample" inputType="multientry">
                <OptionItem option="ONE"/>
                <OptionItem option="TWO"/>
                <OptionItem option="THREE"/>
            </OptionChoice>
        </Group>
        ...
    </Container>
```
numbertext

The Velocity variable for the numbertext input type holds the string specified in the numbertext option.

This example outputs the string AMOUNT and the value in the Number to order box. If the user enters 2 into the Number to order box, the string output is AMOUNT=5.

numstepper

The Velocity variable for the numstepper input type 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”.

outputdata

The Velocity variable for the outputdata control holds the string that appears in the text field. In this example, the name of the Velocity variable is $outputDSName, and the default name that appears in the Data set name: box is Outputds.

```xml
<Metadata>
  <Options>
    <Option inputType="string" name="outputGroup">OUTPUT DATA SET</Option>
    <Option defaultValue="Outputds" indent="1" inputType="outputdata"
      name="outputDSName" required="true">Data set name:</Option>
  </Options>
</Metadata>

UILocalization>
  <OptionItem option="outputDSName"/>
</Group>
</UI>

<CodeTemplate>
  <![CDATA[
    output = $outputDSName
  ]]>}
</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".
select

The Velocity variable for the select input type holds the array of selected values. This example shows a selection list that contains three options. Any or all of these options can be selected.

```xml
<UI>
  <Container option="OPTIONSTAB">
    <Group option="GROUP1">
      <OptionChoice name="SELECTLIST" inputType="select" multiple="true">
        <OptionItem option="Choice1"/>
        <OptionItem option="Choice2"/>
        <OptionItem option="Choice3"/>
      </OptionChoice>
    </Group>
    ...
  </Container>
</UI>
```

```xml
<CodeTemplate>
<![CDATA[
  #if ($SELECTLIST && $SELECTLIST.size() > 0)
  #foreach($item in $SELECTLIST) $item #end
  #end
]]>
</CodeTemplate>
```
slider

The Velocity variable for the slider input type 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)`.

```xml
<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>
```

string

This input type cannot be accessed within the Velocity script.

textbox

The Velocity variable for the textbox input type holds the current string in the text box.

The following example sets an internal Velocity variable, called `$Note`, to the current text string in the text box.

```xml
<Option name="textSimple" required="true" inputType="textbox"
  defaultValue="'Enter text here'">Text Box</Option>

<CodeTemplate>
  <![CDATA[
    #set($note = $textSimple)]]
</CodeTemplate>
```
The Velocity variable for the validationtext input type holds the string that was specified in the text box.

The following example outputs the string $\rho_0=$ and the text in the **Null hypothesis correlation** option. If the user specifies 0, the resulting string is $\rho_0=0$.

```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 greater than -1 and less than 1 for the null hypothesis correlation"
    regExp="[-+]?(0\.\d*)|(\d+\.\d+)|0">Null hypothesis correlation:</Option>
</Options>

<CodeTemplate>
  <![CDATA[
  rh0=$nullRho$]
</CodeTemplate>
Example: Task Definition for List Data Task

Open the List Data Task ................................................................. 93
View the XML Code for the List Data Task .............................. 94
Understanding the XML Code for the List Data Task ............... 95

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 in 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
<?xml version="1.0" encoding="UTF-8"?>
<Task schemaVersion="2.0">
  <Registration>
    <Name>List Data</Name>
    <Description>The List Data task displays the contents of a table</Description>
    <GUID>454536b-405b-493e-b21a-ef19d70a3e1c</GUID>
    <Procedures>PRINT SORT</Procedures>
    <Version>3.3</Version>
    <Links>
      <Link href="http://support.sas.com/odlsearch?nh=25400" />
      <Link href="http://support.sas.com/odlsearch?nh=55400" />
      <Link href="" />
      <Link href="" />
      <Link href="http://support.sas.com/dsearch?ct=52400" />
      <Link href="http://support.sas.com/dsearch?amp;ct=52400" />
      <Link href="" />
      <Link href="" />
      <Link href="http://support.sas.com/kb/?Find=Search" />
      <Link href="http://support.sas.com/kb/?Find=Search" />
    </Links>
    <Registration/>
  </Registration>
  <Metadata>
    <DataSources>
      <DataSource name="DATASOURCE">
        <Roles>
          <Role maxVars="0" minVars="0" name="" />
          <Role maxVars="0" minVars="0" name="" />
          <Role maxVars="0" minVars="0" name="" />
        </Roles>
      </DataSource>
    </DataSources>
  </Metadata>
</Task>
```
Chapter 8 / Example: Task Definition for List Data Task

The List Data task displays the contents of a table as a report.

GUID: 3C74D5C3-B845-4926-A749-E56CBA1283E2

Procedures: PRINT SORT

Version: 3.3

Links:
- PROC PRINT Documentation
- PROC SORT Documentation

Metadata

DataSources

DataSources

Options

Option:
- Rows to list:
- All rows
- First n rows
- First n percent of rows
- Every nth row
- Amount (n)
- Percent (n)

Options:

DATATAB
DATAGROUP
ROLESGROUP
Understanding the XML Code for the List Data Task

```xml
<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>
</UI>
```
Chapter 8 / Example: Task Definition for List Data Task

</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>

<OptionChoice option="ROWS2LIST">
<OptionItem option="all"/>
<OptionItem option="firstnrows"/>
</OptionChoice>

<OptionItem option="NVALUE"/>
Understanding the XML Code for the List Data Task

```xml
<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>

<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" )
#endif

/* Print the table */

proc print data=$TABLE
#if(ROWS2LIST.equalsIgnoreCase("firstnrows"))
  (obs=$NVALUE)
#endif
#if ($OBS == '1')
  obs="$OBSHEADING"
#else
  noobs
#endif
#if ($HEADING && ($HEADING !="default"))

```
The Registration element. For more information, see Chapter 2, “Working with the Registration Element,” on page 11.
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 Chapter 3, “Working with the Metadata Element,” on page 13.

The UI element specifies the layout of the task in the user interface. For more information, see Chapter 4, “Working with the UI Element,” on page 53.

The Dependencies element specifies the options that rely on one another in order for the task to work. For more information, see Chapter 5, “Working with the Dependencies Element,” on page 59.

The Code Template element generates the SAS code for the task. For more information, see Chapter 7, “Understanding the Code Template,” on page 77.
Chapter 8 / Example: Task Definition for List Data Task
Appendix 1

Common Utilities for CTM Writers

About the Predefined $CTMUtil Variable

The predefined $CTMUtil variable provides access to some common utilities. Several methods are currently available.

quotestring Method

**Short Description**
Encloses a string in single quotation marks.

**Syntax**
```
String quotestring(String input)
```

**Parameters**
- `input` the input string that you want to enclose in single quotation marks.
Return Value

This method returns a string that represents the quoted value. Single quotation marks are added to the input string. Any single quotation marks that are found in the original string are preserved by adding another single quotation mark.

Example

```java
#set( $input="Person's" )
$CTMUtil.quoteString($input);
/* string returned: 'Person''s' */
```

toSASName Method

Short Description

Transforms a string so that it uses SAS naming conventions.

Syntax

```java
String toSASName(String input)
```

Parameters

- `input` the input string to transform.

Return Value

This method returns a string that represents the transformed input string. For example, if the input string is ‘My Variables’, the returned string would be “'My Variables"n'.

Example

```java
#set( $input="My Variable" )
$CTMUtil.toSASname($input);
/* string returned: "My Variable"n */
```
Recommended Reading

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

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

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

support.sas.com/bookstore for additional books and resources.