# Contents

*About This Book* ..................................................................................................................  v

**Chapter 1 • Introduction to the Common Task Model** .......................................................... 1  
  About the SAS Studio Tasks .................................................................................................. 1  
  Edit a Predefined Task ........................................................................................................ 3  
  Using Sample Tasks ........................................................................................................... 3  
  Create a New Task ................................................................................................................ 5  
  Create a Task with Default Option Settings ...................................................................... 6  
  Validation Steps for the Task ............................................................................................... 7  
  Testing a Task .................................................................................................................... 7  
  Sharing Tasks .................................................................................................................... 7  

**Chapter 2 • Working with the Registration Element** ......................................................... 9  
  About the Registration Element .......................................................................................... 9  
  Example: The Registration Element from the Sample Task ............................................ 10

**Chapter 3 • Working with the Metadata Element** ............................................................... 11  
  About the Metadata Element ............................................................................................... 11  
  Working with the DataSources Element ............................................................................ 11  
  Working with the Options Element .................................................................................... 14

**Chapter 4 • Working with the UI Element** ......................................................................... 49  
  About the UI Element ......................................................................................................... 49  
  Example: UI Element for the Sample Task ....................................................................... 50

**Chapter 5 • Working with the Dependencies Element** ..................................................... 53  
  About the Dependencies Element ....................................................................................... 53  
  Notes on Dependencies ....................................................................................................... 55  
  Creating Dependencies for Group Elements ...................................................................... 56  
  Using Radio Buttons as Targets of Dependencies ............................................................. 57  
  Example 1: Selecting a Check Box to Show a Group of Options ..................................... 58  
  Example 2: Using Radio Buttons to Create Dependencies .............................................. 59  
  Example 3: Using Combobox Controls ............................................................................ 64

**Chapter 6 • Working with the Requirements Element** ...................................................... 69  
  About the Requirements Element ....................................................................................... 69  
  Example: Using a Requirements Element for Roles ........................................................ 69

**Chapter 7 • Understanding the Code Template** ................................................................. 71  
  About the Code Template ................................................................................................... 72  
  Using Predefined Velocity Variables ................................................................................ 72  
  Predefined SAS Macros ....................................................................................................... 73  
  Working with the DataSource Element in Velocity ........................................................... 73  
  Working with Role Elements in the Velocity Code ............................................................ 78  
  How the Options Elements Appear in the Velocity Code ................................................ 79  
  Working with the OutputData Control in Velocity ............................................................ 95

**Appendix 1 • Common Utilities for CTM Writers** ............................................................... 97  
  SCTMMathUtil Variable ..................................................................................................... 97
About This Book

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.
About This Book
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.
The **Tasks** element has two attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>schemaVersion</strong></td>
<td>Specifies the schemaVersion associated with the task.</td>
</tr>
<tr>
<td><strong>fetchProductLicenses</strong></td>
<td>Specifies whether to retrieve the product license information. This attribute must be set to <strong>true</strong> if the <strong>isProductLicensed</strong> method is used in the Code Template, Requirements, or Dependencies elements. For more information, see “isProductLicensed Method” on page 99. The default value for this attribute is <strong>false</strong>.</td>
</tr>
</tbody>
</table>

The **Task** element 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.
Edit a Predefined 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 and Utilities section.
2. Expand the folder that contains the task.
3. Right-click the name of the task and select Add to My Tasks.
4. Specify the name of the new task and click OK. A copy of the task is added to your My Tasks folder.
5. Open the My Tasks folder and select the copied task.
6. Click . The XML and Velocity code for the task appears. You can now edit this code and save your changes to your My Tasks folder.

Using Sample Tasks

What Is the Difference between the Sample Task and the Advanced Task?

The Sample Task shows the controls that are available to you when writing a task. The Advanced Task shows some of the more complex functionality in the common task model. For example, the Advanced Task includes dependencies, the mixed effects builder, data linking, and return values.

View the Sample Task

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

View the Advanced Task

To view the Advanced Task:

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

2. Click and select Advanced Task.
The Advanced Task that is shipped with SAS Studio appears.

Create a New Task

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

To create a new task:
1. In the navigation pane, open the Tasks and Utilities section.
2. Click 🖖 and select New Task.
The new task appears in SAS Studio.

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

4. To save the task, click **Save**.

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 **Save**. 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**.
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 6.

You can share CTM and CTK files by attaching these files to an email 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 email.)
2. In SAS Studio, open the Folders section and click 🧾. The Upload Files window appears.
3. Specify where you want to upload the files and click Choose Files to select a file.
4. Click Upload.

Sharing a Task That You Created

If you save the CTM or CTK file to a shared network location, other users can create a folder shortcut to access the task from SAS Studio. The advantage to this approach is that you have only one copy of the CTM file.
To create a new folder shortcut, 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.
Chapter 2
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>
</tbody>
</table>
| Links        | A list of hyperlinks to help or resources related to this task.  
  Note: If you do not have any resources to link to, this element is optional. |
Example: The Registration Element from the Sample Task

Here is the Registration element from the Sample Task:

```xml
<Registration>
  <Name>Sample Task</Name>
  <Description>Demonstrates the Common Task Model functionality.</Description>
  <GUID>C6AC34BD-D14A-4CF5-BF2F-A110711BF819</GUID>
  <Procedures>PRINT</Procedures>
  <Version>3.8</Version>
  <Links>
    <Link href="http://documentation.sas.com/?softwareId=STUDIOMID&softwareVersion=3.8&softwareContextId=tasks&requestor=inapp" nlsKey="LinkKey999">SAS Studio documentation</Link>
  </Links>
</Registration>
```
Chapter 3
Working with the Metadata Element

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. Most tasks need only one data source, but multiple data sources can be defined. The DataSource element specifies the information about the data set 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>
</tbody>
</table>
### Working with the Roles Element

**About the Roles Element**

The `Roles` element is the only child of the `DataSource` element. The `Roles` element identifies the variables that must be assigned in order to run the task. This element groups 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>
</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.  
<p>| minVars   | Specifies the minimum number of columns that must be assigned to this role. If <code>minVars=“0”</code>, the role is optional. If <code>minVars=“1”</code>, a column is required to run this task and a red asterisk appears next to the label in the user interface. |</p>
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxVars</td>
<td>Specifies the maximum number of columns that can be assigned to this role. If <code>maxVars=&quot;0&quot;</code>, users can assign an unlimited number of columns to this role.</td>
</tr>
<tr>
<td>exclude</td>
<td>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.</td>
</tr>
<tr>
<td>order</td>
<td>Specifies that the user can order the columns that are assigned to this role. Valid values are <code>true</code> and <code>false</code>. If <code>order=&quot;true&quot;</code>, the user can use the up and down arrows in the user interface to modify the order.</td>
</tr>
<tr>
<td>fetchDistinct</td>
<td>Specifies whether to retrieve the distinct information for columns assigned to this role. The default value is <code>false</code>.</td>
</tr>
</tbody>
</table>

**Example: DataSources and Roles Elements from the Sample Task**

Here is an example of the `DataSources` and `Roles` elements from the Sample Task:

```xml
<DataSources>
  <DataSource name="DATASOURCE">
    <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>
```
When you run this code, you get the Data and Roles sections in this example:

```xml
<DATA OPTIONS OPTION>
  <DATA>
    SASHELP:PRICEDATA
  </DATA>
  <ROLES>
    <*Required variable: (1 item)> Column
    <Numeric variable: (1 item)>
    <Character variable: (1 item)>
  </ROLES>
</DATA OPTIONS OPTION>
```

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>
</tbody>
</table>
| **inputType** | Specifies the input control for this option. Here are the valid values:  
  - checkbox  
  - color  
  - combobox  
  - datepicker  
  - distinct  
  - dualselector  
  - inputtext  
  - mixedeffects  
  - multientry  
  - numstepper  
  - numbertext  
  - outputdata  
  - passwordtext  
  - radio  
  - sasserverpath  
  - select  
  - slider  
  - string  
  - textbox  
  - validationtext  
For more information, see “Supported Input Types” on page 16. |
| **indent** | Specifies the indentation for this option in the task interface.  
Here are the valid values:  
  - 1 – minimal indentation (about 17px)  
  - 2 – average indentation (about 34px)  
  - 3 – maximum indentation (about 51px) |
| **returnValue** | 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 42. |
| **width** | Specifies the width of the control. The width can be specified in %, em, or px. The default behavior is to autosize the control based on available width and content. |
**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:

```
<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:

```
<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
  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>
<tbody>
<tr>
<td><strong>required</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>selectMessage</strong></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><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>
<tr>
<td><strong>editable</strong></td>
<td>Specifies whether the user can enter a value in the combobox control. By default, users cannot enter a new value in the combobox control.</td>
</tr>
</tbody>
</table>

The code in the Sample Task 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:

![Combobox example](image)

**datepicker**

This input type has these attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>format</strong></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 mmdyys8. (12/24/93).</td>
</tr>
</tbody>
</table>
### Attribute Description

**required**
Specifies whether a date is required. By default, no date is required.

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

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 creates datepicker control with the label *Choose a date*:

```xml
<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:

![Datepicker Example](image)

### distinct
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. The default value is false.  
| **selectMessage** | 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. |

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.
When using the `distinct` control, remember this information:

- Missing values are excluded from the list of returned values.

- The `distinct` control is affected by any filter that is applied to the data source. For more information, see the `where` attribute in “About the DataSources Element” on page 11.

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>
<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>height</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>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>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>

You can specify default values for the dualselector control by using the `defaultValue` 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 `defaultValue` attribute.

This example shows how the dualselector control works.

```xml
<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>
```
When you run this code, the **Test choices** option appears in the user interface. In this example, the `defaultValue` 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><strong>required</strong></td>
<td>Specifies whether any input text is required. Valid values are <strong>true</strong> and <strong>false</strong>. The default is <strong>false</strong>.</td>
</tr>
<tr>
<td><strong>missingMessage</strong></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><strong>promptMessage</strong></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><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>

The code in the Sample Task 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.

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

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*X. You can have second-order, third-order, ...nth-order polynomial effects.

The mixedeffects control enables users to create various model effects. You can define fixed effects, random effects, repeated effects, means effects, and zero-inflated effects. For the control to work properly, you must specify at least one of the role attributes, roleContinuous or roleClassification. If no roles are specified, the control is displayed but the user has no variables to work with.

Here are the attributes for the mixedeffects input type:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>effects</td>
<td>Specifies the list of effects that you want available from the task interface:</td>
</tr>
<tr>
<td></td>
<td>• fixed – only fixed effects. This is the default value.</td>
</tr>
<tr>
<td></td>
<td>• fixedrandom – fixed effects and random effects.</td>
</tr>
<tr>
<td></td>
<td>• fixedrandomrepeated – fixed effects, random effects, and repeated effects.</td>
</tr>
<tr>
<td></td>
<td>• fixedrepeated – fixed and repeated effects.</td>
</tr>
<tr>
<td></td>
<td>• meanszero – means and zero-inflated effects.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>roleContinuous</td>
<td>Specifies the role that contains the continuous variables.</td>
</tr>
<tr>
<td>roleClassification</td>
<td>Specifies the role that contains the classification variables.</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>fixedInterceptVisible</td>
<td>Specifies whether the intercept option is available for fixed effects or mean effects. Valid values are true and false. The default value is true.</td>
</tr>
<tr>
<td>fixedInterceptDefaultValue</td>
<td>Specifies the default value for the intercept option if fixedInterceptVisible = true. Valid values are 0 and 1. The default value is 1.</td>
</tr>
<tr>
<td>randomInterceptVisible</td>
<td>Specifies whether the intercept option is available for random effects. Valid values are true and false. The default value is true.</td>
</tr>
<tr>
<td>randomInterceptDefaultValue</td>
<td>Specifies the default value for the intercept option if randomInterceptVisible = true. Valid values are 0 and 1. The default value is 1.</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>

Here is an example of the mixedeffects control from the Advanced Task:

```xml
<Options>
  <Option name="MECTAB" inputType="string">MIXED EFFECTS CONTROL</Option>
  <Option name="MECTEXT" inputType="string">This tab shows an example of the Mixed Effects control. The variables come from both the Variables and Numeric Variables roles.</Option>
  <Option name="mixedEffects" inputType="mixedeffects"
    roleContinuous="dataVariablesNumeric" roleClassification="dataVariables"
    excludeTools="POLYEFFECT, TWOFACT, THREEFACT, NFACTPOLY"></Option>
...
</Options>

<UI>
  <Container option="MECTAB">
```
If you run the Advanced Task, here is the resulting **Mixed Effects Control** tab:

![Mixed Effects Control Tab](image)

This tab shows an example of the Mixed Effects control. The variables come from both the Variables and Numeric Variables roles.

If you click **Edit**, the Model Effects Builder appears.

The component opens, but there are no variables to work with in the **Variables** pane. You must assign a variable to the continuous variable or classification variable role. You can assign variables to both roles.
In the Advanced Task, close the Model Effects Builder and click the Data tab. Select an input data source (such as Sashelp.Pricedata) and assign variables to the Variables and Numeric Variables roles.

Return to the Model Effects Control tab and click Edit. Now, the price and sale variables are available from the Variables pane.
**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 <strong>true</strong> and <strong>false</strong>. The default value is <strong>false</strong>.</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 <strong>true</strong> and <strong>false</strong>. The default value is <strong>false</strong>.</td>
</tr>
</tbody>
</table>

The code in the Sample Task creates the **Multiple entry** option.

```<Options>```
In this example, the **Multiple entry** option has three 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 +.

To enable users to reorder the values in this list, set the **reorderable** attribute to **true**, as shown in this example.
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 <code>decimalPlaces=&quot;0,3&quot;</code>. 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>
<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 <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>
</tbody>
</table>
This example code creates a field called **Number text**.

```
<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.
  <Option name="numberTextEXAMPLE" defaultValue="1"
    inputType="numbertext"
    minValue="0"
    maxValue="100"
    promptMessage="Enter a number between 0 and 100."
    missingMessage="Enter a number between 0 and 100."
    rangeMessage="This number is out of range. Enter a number between 0 and 100."
    invalidMessage="Invalid value. 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 <code>decimalPlaces=&quot;0,3&quot;</code>.</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>invalidMessage</td>
<td>Specifies the tooltip text that appears when the content in the field 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>
<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 field is empty but a value is required.</td>
</tr>
<tr>
<td>promptMessage</td>
<td>Specifies the tooltip text that appears when the field is empty and the mouse is positioned over the field.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>rangeMessage</td>
<td>Specifies the tooltip text 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>

The first example in the Sample Task creates an option with an assigned default value of 5.

```
<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: 5
```

The second example in the Sample Task 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: 80
```

**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 <strong>false</strong>, which means that no name is required.</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>
<tr>
<td>libraryEngineExclude</td>
<td>Specifies the engine types that are not valid for the data source. The engine types should be a comma-separated list.</td>
</tr>
<tr>
<td>libraryEngineInclude</td>
<td>Specifies the engine types that are valid for the data source. The engine types should be a comma-separated list.</td>
</tr>
<tr>
<td>unique</td>
<td>Specifies whether the value of the output control is unique. This is a Boolean value with a default value of <strong>false</strong>. If <strong>unique=true</strong>, the task ensures that the value of the outputdata control is unique when compared to all other outputdata control where <strong>unique=true</strong>.</td>
</tr>
</tbody>
</table>

**Note:** If you do not specify either the `libraryEngineExclude` parameter or the `libraryEngineInclude` parameter, all engine types are available for the data source control. If you need to limit the engine type, use either the `libraryEngineExclude` parameter or the `libraryEngineInclude` parameter. Do not specify both.

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 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.

The `defaultValue` attribute contains the initial value for the output data set that is created by the task. 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:

```
<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 \texttt{false}, which means that no name is required.</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>
```

When using this control, remember these restrictions:

- When you save the task file, the password that is currently entered is not saved.
- The \texttt{defaultValue} attribute is not supported by the passwordtext control.
- The value of the passwordtext control cannot be set by a dependency.

Here is an example:

```
<Option name="pswd" inputType="passwordtext">Enter Password:</Option>
```

\textit{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>
```

One radio button must be selected. If none of the values for the radio button list include the \texttt{defaultValue} attribute, the first button in the list is selected.

The example in the Sample Task creates an option called Radio button group label with Radio button 1 selected by default.

```
<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>
</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.
- Radio button 1
- Radio button 2
- Radio button 3

*sasserverpath*

The *sasserverpath* control enables the user to choose a file or folder location on the SAS server. The default folder location is the user’s SAS home directory.

This input type has these attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
</table>
| pathType    | Specifies the type of selector for this control. Here are the valid values:  
  - *file* enables the user to select a file from the SAS server.  
  - *folder* enables the user to select a folder from the SAS server. The folder must already exist on the server. The user can create a new folder by using the Folder Selection window.  
  - *project* enables the user to select a folder for a project. This folder does not have to exist yet, but the path to the folder must exist. The user can create a new folder by using the Folder Selection window.  |
| width       | Specifies the width of the control in percent (%), em, or px. |

Additional attributes are available if this control is a file selector or a project selector.

Here are the additional attributes for the file selector:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultExtension</td>
<td>Specifies the default extension for the file. If no value is specified, the default extension is <em>sas</em>.</td>
</tr>
<tr>
<td>defaultName</td>
<td>Specifies the default filename. If no value is specified, the default filename is <em>program</em>.</td>
</tr>
</tbody>
</table>
Here is the additional attribute for a project selector:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultName</td>
<td>Specifies the default name for the project folder. If no value is specified, the default filename is <code>project</code>.</td>
</tr>
</tbody>
</table>

Here are several examples of the `saserverpath` control:

```xml
<Option name="fileSelector" inputType="saserverpath" defaultFileName="myProgramFile" pathType="file" defaultExtension="sas">Select a new or existing file:</Option>
<Option name="folderSelector" inputType="saserverpath" pathType="folder">Select a folder:</Option>
<Option name="projectFolderName" inputType="saserverpath" pathType="project" defaultName="RPM">Select a project:</Option>
```

**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 <code>true</code> and <code>false</code>. The default value is <code>true</code>.</td>
</tr>
<tr>
<td>required</td>
<td>Specifies whether the user must select a value from the list. Valid values are <code>true</code> and <code>false</code>. The default value is <code>false</code>.</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 43.</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 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>

<UI>
  ...
  <OptionItem option="labelSELECT" />
  <OptionChoice option="selectEXAMPLE">
    <OptionItem option="value1"/>
    <OptionItem option="value2"/>
    <OptionItem option="value3"/>
  </OptionChoice>
</UI>
```
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 <code>discreteValues=&quot;3&quot;</code>, 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 <code>true</code> and <code>false</code>. The default value is <code>true</code>.</td>
</tr>
</tbody>
</table>

The first example in the Sample Task 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 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:

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 OptionChoice tag.</td>
</tr>
</tbody>
</table>

The code for the Sample Task 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

textarea

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>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>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>height</td>
<td>Specifies the height of the control. This value can be in em or px. By default, SAS Studio sizes the control based on the available height and content.</td>
</tr>
<tr>
<td>splitLines</td>
<td>Specifies whether to split the text into an array of lines. The split is determined by the newline character. The default value is false.</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>
</table>
| `required`      | Specifies whether any input text is required. Valid values are `true` and `false`. The default value is `false`.
| `invalidMessage` | Specifies the tooltip text to display when the content in the text box is invalid. By default, no message is displayed. |
| `missingMessage` | Specifies the tooltip text that appears when the text box is empty but text is required. By default, no message is displayed. |
| `promptMessage`  | Specifies the tooltip text that appears when the text box is empty and the text box is selected. By default, no message is displayed. |
| `regExp`         | Specifies the regular expression pattern to use for validation. This syntax comes directly from JavaScript Regular Expressions. |
| `width`          | Specifies the width of the control. This value can be in percent (%), em, or px. By default, SAS Studio sizes the control based on the available width and content. |

The code for the Sample Task creates a text box called **Validation text**.

```
<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"
    promptMessage="Enter a string 5 characters long."
    invalidMessage="Invalid value. You must specify a string of 5 characters."
    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 Enter a string 5 characters long message appears.

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

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

Organizing Options into a Table Component

The OptionTable element defines a table component that contains one or more custom-defined columns. Each column contains one CTM option. Each individual column can contain a different CTM option. Here are the available CTM options:

- checkbox
- combobox
- numbertext
- numstepper
- textbox

Each row in the column has the same CTM control. If you specify the addRemoveRowTools attribute, users can add and delete rows from the table.

Here is an example from the Pearson Correlation task (in the Power and Sample Size group):

SIGNIFICANCE LEVEL
Alpha values: (minimum 1 row) 0.05
Use these attributes to create the table:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Specifies the name assigned to the option.</td>
</tr>
<tr>
<td>label</td>
<td>Specifies the label for the table in the user interface.</td>
</tr>
</tbody>
</table>
| indent                     | Specifies the indentation for this option in the task interface. Here are the valid values:  
    • 1 – minimal indention (about 17px)  
    • 2 – average indention (about 34px)  
    • 3 – maximum indention (about 51px) |
| initialNumberOfRows        | Specifies the number of empty rows in a new table. This value must be greater than or equal to 1. By default, this value is 1. |
| addRemoveRowTools          | Specifies whether to enable the user to add and remove rows from the table. Valid values are true and false. When this value is set to true, icons for adding and removing rows appear above the table. By default, this value is false, so the task interface contains only the number of rows that you specified using the initialNumberOfRows attribute. |
| showColumnHeadings         | Specifies whether to show the column headings in the table. Valid values are true and false. The default value is false, and no column headings are displayed. |
| minimumRequiredRows        | Specifies the minimum number of rows that must be completed. This value must be greater than or equal to 1. The default value is 1. |
| maximumRows                | Specifies the maximum number of rows in the option table. The default value is 0. |
| noIncompleteRows           | Specifies whether incomplete rows are allowed in the table. Valid values are true and false. The default value is false. If this attribute is set to true, the task cannot run if there are any incomplete rows in the table. |

The OptionTable element can have only one child, the Columns element. The Columns element can contain multiple Column elements. Each Column element describes a column in the table.

Each column must be defined in an Option element in the metadata. In the Option element, the values for the name and width attributes are ignored. Specify the initial column width by using the width attribute in the Column element.
You can use these input types for the columns in the option table:

- checkbox
- combobox
- numbertext
- numstepper
- textbox

Here are the attributes for the Column element:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Specifies the name of the column. This attribute is required.</td>
</tr>
<tr>
<td>label</td>
<td>Specifies the label of the column.</td>
</tr>
<tr>
<td>defaultValues</td>
<td>Specifies a list of default values for the first several rows. These values apply only when the table is created. If this attribute is not specified for the column, the value of defaultValue for the cell is used instead. The defaultValues column attribute takes precedence over the defaultValue cell attribute.</td>
</tr>
<tr>
<td>width</td>
<td>Specifies the initial width of the column. This width is in pixels. If you do not specify a width, the column width is an estimate based on the properties of the column widget.</td>
</tr>
</tbody>
</table>

Here is an example that uses the OptionTable element:

```xml
<OptionTable name="optionTable" initialNumberOfRows="3" addRemoveRowTools="false">
  <Columns>
    <Column name="colNumberText" label="NumberText" labelKey="alphaKey">
      <Option inputType="numbertext" minValue="1" maxValue="10"
               decimalPlaces="0,4" required="true"/>
    </Column>

    <Column name="colNumStepper" label="NumStepper">
      <Option inputType="numstepper" minValue="0" maxValue="10"
               decimalPlaces="0" increment="1" required="true"
               missingMessage="Custom missing message: Enter a number between 0 and 10."
               invalidMessage="Custom invalid message: Enter a number between 0 and 10."/>
    </Column>

    <Column name="colCheckBox" label="CheckBox">
      <Option inputType="checkbox"/>
    </Column>

    <Column name="colTextBox" label="TextBox">
      <Option inputType="textbox"/>
    </Column>
  </Columns>
</OptionTable>
```
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. 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.</Option>
  <Option name="vegetables" inputType="select" multiple="true">
    Select the vegetables
  </Option>
  <Option name="carrots" inputType="string">Carrots</Option>
  <Option name="peas" inputType="string">Peas</Option>
  <Option name="corn" inputType="string">Corn</Option>
</Options>
<UI>
  <Container option="RETURNVALUETAB">
    <OptionItem option="labelReturnValue"/>
    <OptionChoice option="vegetables">
      <OptionItem option="carrots"/>
      <OptionItem option="peas"/>
      <OptionItem option="corn"/>
    </OptionChoice>
    ...
  </Container>
</UI>
```
If you run the Advanced Task, here is the resulting Return Value tab.

This tab shows an example of the option's `returnValue` attribute. This attribute can be used in `OptionChoice` controls to customize velocity return values.

Select the vegetables:
- Carrots
- Peas
- Corn

### 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 mixed effects control. If the select control links to anywhere else, any children in the `OptionChoice` element 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.

```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 MIXED EFFECTS CONTROL</Option>
<Option name="selectMEB" inputType="select" multiple="true" sourceLink="mixedeffects">This select is populated from the output of the Mixed Effects Control.</Option>

...
If you run the code for the Advanced Task, 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 type="N" maxVars="0" order="true" minVars="0" name="NUMVAR" exclude="VAR">Numeric Variable</Role>
    </Roles>
  </DataSource>
</DataSources>
```

**Linking to Roles**

This select control is populated with the variables selected on the Data tab.

**Linking to Mixed Effects Control**

This select control is populated from the output of the mixed effects control.

**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.
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 “Working with Role Elements in the Velocity Code” on page 78.

**Linking to Effects from the Mixed Effects Control**

If a select control is linked to a `mixedeffects` input type, the values in the select control are the list of effects in the mixed effects control.

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

The Velocity variable that is created for the select control is \$mecList. The contents of the \$mecList variable mimic the output of the mixed effects control. For more information, see “modelbuilder” on page 87.

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

The **Variables** pane in the Model Effects 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>
<Option inputType="mixedeffects" name="mixedEffects"
    excludeTools="POLYEFFECT, TWOFACT, THREEFACT, NFACTPOLY"
    roleClassification="classVariable"
    roleContinuous="continuousVariables"
    width="100%">Model</Option>

... 
<Option inputType="string" name="multCompareGroup">Multiple Comparisons</Option>
```
1 Creates the mixed effects control 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 Effects 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.
Chapter 4
Working with the UI Element

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 Container tag takes only one attribute. The string for this option is the value of the string input type in the Metadata element.</td>
</tr>
</tbody>
</table>
| Group     | A title for a group of options. The UI engine displays these options sequentially. This tag takes these attributes:  
  • The option attribute is an option name in the metadata. This string is the same as the string value for the metadata option.  
  • The open attribute specifies whether a group is expanded or collapsed. By default, open=false, and the group is collapsed in the user interface. To display the contents of a group by default, specify open=true. |
| DataItem  | A reference to an input data source. This tag has only one attribute. The string for this option is the value of the string input type in the Metadata element. |
### Child Description

<table>
<thead>
<tr>
<th>Child</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 string input type in the Metadata 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 option attribute only. The option attribute refers to the metadata name attribute for the option. The string for this option is taken from the metadata string value.</td>
</tr>
</tbody>
</table>
| OptionChoice   | A reference to an option that has a choice of values. The OptionChoice element uses the OptionItem or OptionValue element to represent the choice of values. These input types can use the OptionChoice element in the user interface:  
- combobox  
- distinct  
- dualselector  
- multiedit  
- select  

This tag takes the option attribute only. The option attribute refers to the metadata name attribute for the option. The string for this option is taken from the metadata string value. |
| OptionValue    | A value choice. This tag is valid only as a child of the OptionChoice element. |

---

**Example: UI Element for the Sample Task**

The code for the Sample Task 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="ROLESGROUP" open="true">
      <RoleItem role="VAR"/>
      <RoleItem role="OPTNVAR"/>
      <RoleItem role="OPTCVAR"/>
    </Group>
  </Container>
  <Container option="OPTIONSTAB">
    <Group option="GROUP" open="true">
      <OptionItem option="labelEXAMPLE"/>
    </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.)

<table>
<thead>
<tr>
<th>DATA</th>
<th>OPTIONS</th>
<th>OPTION TABLE</th>
<th>INFORMATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUPS</td>
<td>An example of a group. Groups are used to organize options.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHECK BOX</td>
<td>An example of a check box. Check boxes are either on or off.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHECK BOX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COLOR SELECTOR</td>
<td>An example of a color selector.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COLOR SELECTOR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMBOBOX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMBOBOX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATE PICKER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATE PICKER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISTINCT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISTINCT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LISTS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LISTS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUMERIC STEPPER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUMERIC STEPPER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RADIO BUTTONS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RADIO BUTTONS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SELECTORS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SELECTORS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SLIDER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SLIDER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEXT FIELDS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEXT FIELDS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 5

Working with the Dependencies Element

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>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>conditionResult</strong></td>
<td>specifies when to execute the action. The valid values for this attribute are <strong>true</strong> and <strong>false</strong>.</td>
</tr>
<tr>
<td>• If the condition is true and conditionResult=true, the action is executed.</td>
<td></td>
</tr>
<tr>
<td>• If the condition is false and conditionResult=false, the action is executed.</td>
<td></td>
</tr>
<tr>
<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>
<td></td>
</tr>
<tr>
<td><strong>action</strong></td>
<td>specifies the action to execute. Here are the valid values:</td>
</tr>
<tr>
<td>• show</td>
<td></td>
</tr>
<tr>
<td>• hide</td>
<td></td>
</tr>
<tr>
<td>• enable</td>
<td></td>
</tr>
<tr>
<td>• disable</td>
<td></td>
</tr>
<tr>
<td>• set</td>
<td></td>
</tr>
<tr>
<td>If the value of the <strong>action</strong> attribute is <strong>set</strong>, you must also specify these two attributes:</td>
<td></td>
</tr>
<tr>
<td>• The <strong>property</strong> attribute refers to the attribute of an element that was created from the metadata. The <strong>option</strong> element in the metadata has an <strong>inputType</strong> attribute that specifies what UI element is created.</td>
<td></td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td></td>
</tr>
<tr>
<td>Here are a few exceptions:</td>
<td></td>
</tr>
<tr>
<td>• In the <strong>UI</strong> element, any <strong>RoleItem</strong> element cannot be the target of a dependency where <strong>action=set</strong>.</td>
<td></td>
</tr>
<tr>
<td>• The <strong>required</strong>, <strong>width</strong>, <strong>indent</strong>, and <strong>variable</strong> (for the radio input type) attributes are invalid values for the <strong>property</strong> attribute of a <strong>Target</strong> element.</td>
<td></td>
</tr>
<tr>
<td>• The <strong>value</strong> attribute is the value to use for the target of the <strong>property</strong> attribute.</td>
<td></td>
</tr>
<tr>
<td>If the <strong>value</strong> attribute targets an item with the <strong>select</strong> input type, the <strong>value</strong> attribute can accept a single value or a comma-separated list of values.</td>
<td></td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>If the dependency has a comma-separated list of values and the <strong>select</strong> element that the dependency targets is set to <strong>multiple=&quot;false&quot;</strong>, only the first value in the comma-separated list is evaluated. The rest of the values in the list are ignored.</td>
</tr>
</tbody>
</table>
To understand how dependencies work, run the Advanced Task. Examples of dependencies are available from the Dependencies tab.

Notes on Dependencies

- 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 of the UI element 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"/>
    </Group>
  </Container>
</UI>
```
Creating Dependencies for Group Elements

A group element can be the target of a dependency. However, if you want a group element to be the target of a dependency and you also want a child of that group to be the target with a different set of conditions, you must include all of the conditional logic for the group and the child in one dependency.

This example demonstrates this behavior.

```
<UI>
  <Container option="data">
    <Group option="datagroup">
      <Option name="CheckBoxEnableTargetGroup" />
    </Group>
  </Container>
</UI>
```
In the first dependency, the $CheckBoxEnableTargetGroup Velocity variable is used to show or hide the option named targetGroup. The author of this task also wants the option named INPUTTEXT to be displayed based on the state of the $CHECKBOX Velocity variable.

In the second dependency, the logic for targeting the option named targetGroup is omitted. When writing a dependency that targets a group, you must decide whether you want to target the children of that group as well.

Using Radio Buttons as Targets of Dependencies

If a selected radio button is hidden or disabled because of a dependency, another radio button is selected using these criteria:

- If a default radio button that has been specified is visible or enabled, then the default radio button is selected.
- If a default radio button has not been specified or if the default radio button is hidden or disabled, the first available radio button is selected. The order of the radio buttons is determined in the UI element.
If you want to hide or disable a group of radio buttons, you must create a single dependency that targets the variable for the radio buttons. If you create a dependency for each radio button, the result is incorrect behavior.

This example demonstrates the correct and incorrect behavior:

```
<UI>
  <Container option="optionsTab">
    <Group option="RadioButtonGroup open="true">
      <OptionItem option="Radio1"/> <!-- variable="radioVariable1" -->
      <OptionItem option="Radio2"/> <!-- variable="radioVariable1" -->
      <OptionItem option="Radio3"/> <!-- variable="radioVariable1" -->
      <OptionItem option="Checkbox1"/>
    </Group>
    <Group option="RadioButtonGroup2 open="true">
      <OptionItem option="Radio4"/> <!-- variable="radioVariable2" -->
      <OptionItem option="Radio5"/> <!-- variable="radioVariable2" -->
      <OptionItem option="Radio6"/> <!-- variable="radioVariable2" -->
      <OptionItem option="Checkbox2"/>
    </Group>
  </Container>
</UI>

<Dependencies>

1<!-- Correct -->
  <Dependency condition="!($Checkbox1 =='1')">
    <Target option="radioVariable1" conditionResult="true" action="show"/>
    <Target option="radioVariable1" conditionResult="false" action="hide"/>
  </Dependency>

2<!-- Incorrect -->
  <Dependency condition="!($Checkbox2 =='1')">
    <Target option="Radio4" conditionResult="true" action="show"/>
    <Target option="Radio4" conditionResult="false" action="hide"/>
    <Target option="Radio5" conditionResult="true" action="show"/>
    <Target option="Radio5" conditionResult="false" action="hide"/>
    <Target option="Radio6" conditionResult="true" action="show"/>
    <Target option="Radio6" conditionResult="false" action="hide"/>
  </Dependency>
</Dependencies>

1 The first dependency creates a single dependency that targets the variable for the radio buttons.
2 The second dependency creates a dependency for each radio button, which results in the incorrect behavior.
```

---

**Example 1: Selecting a Check Box to Show a Group of Options**

From the Advanced Task, selecting the **Groups can be the target of a dependency** check box determines whether the options under the **Group of Controls** heading are available.
In this example, DEP_CBX is the name for the **Groups can be the target of a dependency** check box, and DEPENDENCYGROUP is the name of the group that contains the options.

```
<Option name="DEP_CBX" inputType="checkbox" defaultValue="1">Groups can be the target of a dependency.</Option>
<Option name="DEPENDENCYGROUP" inputType="string">GROUP OF CONTROLS</Option>

<Dependency condition="($DEP_CBX == '1'">
  <Target option="DEPENDENCYGROUP" conditionResult="true" action="show"/>
  <Target option="DEPENDENCYGROUP" conditionResult="false" action="hide"/>
</Dependency>
```

When the **Groups can be the target of a dependency** check box is not selected, here is what appears on the **Options** tab:

```
DATA    DEPENDENCIES    MIXED EFFECTS CONTROL    
This tab shows examples of Dependencies. Dependencies allow you to show/hide, enable/disable, or in some cases set the values of controls.

Groups can be the target of a dependency.
```

If you select the **Groups can be the target of a dependency** check box, the **Group of Controls** heading and all the options in this group are displayed. Here are the results that appear on the **Options** tab:

```
DATA    DEPENDENCIES    MIXED EFFECTS CONTROL    
This tab shows examples of Dependencies. Dependencies allow you to show/hide, enable/disable, or in some cases set the values of controls.

Groups can be the target of a dependency.
```

**GROUP OF CONTROLS**

Select the type of dependency to see an example of:
- Show / Hide Options
- Enable / Disable Options
- Set Values

Change the combobox value to see options change.

Combobox:

```
Show a color selector
```

[Choose a color]

---

**Example 2: Using Radio Buttons to Create Dependencies**

**About This Example**

The Advanced Task shows how you can use radio buttons to create dependencies. This example has three radio buttons:

- **Show/Hide Options**, which is named radioShowHide in the code.
- **Enable/Disable Options**, which is named radioEnableDisable in the code.
- **Set Values**, which is named radioSetValue in the code.
Here is the code from the Advanced Task:

```xml
<Dependency condition="$radioChoice == 'radioShowHide'">
    <Target action="show" conditionResult="true" option="labelShowChange"/>
    <Target action="show" conditionResult="true" option="comboShowChange"/>
    <Target action="hide" conditionResult="true" option="labelEnableChange"/>
    <Target action="hide" conditionResult="true" option="comboEnableChange"/>
    <Target action="hide" conditionResult="true" option="colorControl"/>
    <Target action="hide" conditionResult="true" option="labelShowSet"/>
    <Target action="hide" conditionResult="true" option="comboSetChange"/>
    <Target action="hide" conditionResult="true" option="checkboxCheckUncheck"/>
</Dependency>

<Dependency condition="$radioChoice == 'radioEnableDisable'">
    <Target action="show" conditionResult="true" option="labelEnableChange"/>
    <Target action="show" conditionResult="true" option="comboEnableChange"/>
</Dependency>
```

Chapter 5 • Working with the Dependencies Element
Selecting the Show/Hide Options Button

As you can see from the XML code, the defaultValue attribute is set to 1 for the radioShowHide option. By default, the Show/Hide Options radio button is selected.

When the Show/Hide Options radio button is selected, the conditions for this dependency are met:

As a result, these lines of code determine the instructional text and label for the combobox:

Example 2: Using Radio Buttons to Create Dependencies
Here are the options that are available when the **Show/Hide Options** radio button is selected:

This tab shows examples of Dependencies. Dependencies allow you to show/hide, enable/disable, or in some cases set the values of controls. Groups can be the target of a dependency.

- **GROUP OF CONTROLS**
  - Select the type of dependency to see an example of:
    - Show / Hide Options
    - Enable / Disable Options
    - Set Values
  - Change the combobox value to see options change.

Selecting the Enable/Disable Options Button

The XML code shows that the name for the **Enable/Disable Options** radio button is `radioEnableDisable`.

```xml
<Option name="radioEnableDisable" variable="radioChoice" inputType="radio">
  Enable / Disable Options
</Option>
```

When the **Enable/Disable Options** radio button is selected, the conditions for this dependency are met:

```xml
<Dependency condition="$radioChoice == 'radioEnableDisable'">
  <Target action="show" conditionResult="true" option="labelEnableChange"/>
  <Target action="show" conditionResult="true" option="comboEnableChange"/>
  <Target action="hide" conditionResult="true" option="labelShowChange"/>
  <Target action="hide" conditionResult="true" option="comboShowChange"/>
  <Target action="show" conditionResult="true" option="colorControl"/>
  <Target action="show" conditionResult="true" option="dateControl"/>
  <Target action="hide" conditionResult="true" option="labelShowSet"/>
  <Target action="hide" conditionResult="true" option="comboSetChange"/>
  <Target action="hide" conditionResult="true" option="checkboxCheckUncheck"/>
</Dependency>
```

As a result, these lines of code determine the instructional text and label for the combobox:

```xml
<Option name="labelEnableChange" inputType="string">Change the combobox value to see options become enabled or disabled.</Option>
<Option name="comboEnableChange" defaultValue="valueEnableColor" inputType="combobox" width="100%">Combobox:</Option>
```
Here are the options that are available when the **Enable/Disable Options** radio button is selected:

This tab shows examples of Dependencies. Dependencies allow you to show/hide, enable/disable, or in some cases set the values of controls.

☐ Groups can be the target of a dependency.

**GROUP OF CONTROLS**

Select the type of dependency to see an example of:

- Show / Hide Options
- Enable / Disable Options
- Set Values

Change the combobox value to see options become enabled or disabled.

Combobox:

- Enable the color selector
- Choose a color
- Choose a date:
- Slider with buttons

---

**Selecting the Set Values Button**

The XML code shows that the name for the **Set Values** radio button is radioSetValue.

```xml
<Option name="radioSetValue" variable="radioChoice" inputType="radio">Set Values</Option>
```

When the **Set Values** button is selected, the conditions for this dependency are met:

```xml
<Dependency condition="$radioChoice == 'radioSetValue'">
  <Target action="hide" conditionResult="true" option="labelShowChange"/>
  <Target action="hide" conditionResult="true" option="comboShowChange"/>
  <Target action="hide" conditionResult="true" option="labelEnableChange"/>
  <Target action="hide" conditionResult="true" option="comboEnableChange"/>
  <Target action="hide" conditionResult="true" option="colorControl"/>
  <Target action="hide" conditionResult="true" option="dateControl"/>
  
  <Target action="show" conditionResult="true" option="labelShowSet"/>
  <Target action="show" conditionResult="true" option="comboSetChange"/>
  <Target action="show" conditionResult="true" option="checkboxCheckUncheck"/>
</Dependency>
```

As a result, these lines of code determine the instructional text and label for the combobox:

```xml
<Option name="labelShowSet" inputType="string">Change the combobox value to change the value of the checkbox.</Option>
```

```xml
<Option name="comboSetChange" defaultValue="valueSetCheck" inputType="combobox" width="100%">Combobox</Option>
```
Here are the options that are available when the Set Values radio button is selected:

This tab shows examples of Dependencies. Dependencies allow you to show/hide, enable/disable, or in some cases set the values of controls.

- Groups can be the target of a dependency.

**GROUP OF CONTROLS**

Select the type of dependency to see an example of:

- Show / Hide Options
- Enable / Disable Options
- Set Values

Change the combobox value to show the value of the checkbox.

**Combobox:**

- Check the checkbox
- Checkbox

---

**Example 3: Using Combobox Controls**

**Using a Value to Show or Hide Additional Options**

In the Advanced Task if you select the Show/Hide Options radio button, the values in the combobox control are determined by these lines of code:

```xml
<Option name="comboShowChange" defaultValue="valueShowColor" inputType="combobox" width="100%">Combobox:</Option>
<Option name="valueShowColor" inputType="string">Show a color selector</Option>
<Option name="valueShowDate inputType="string">Show a date picker</Option>
<Option name="valueShowSlider" inputType="string">Show a slider control</Option>
```

Here is how these options appear in the user interface:

**Combobox:**

- Show a color selector
- Show a date picker
- Show a slider control

If you select **Show a color selector** from the combobox control, the conditions for this dependency are met:

```xml
<Dependency condition="$comboShowChange == 'valueShowColor'">
  <Target action="show" conditionResult="true" option="colorControl"/>
  <Target action="hide" conditionResult="true" option="dateControl"/>
  <Target action="hide" conditionResult="true" option="sliderControl"/>
</Dependency>
```
As a result, the Color control (named colorControl in the XML code) appears in the user interface. (According to the conditions defined in the dependency, the date picker and slider controls are hidden.) Here is the XML code for colorControl. The defaultValue attribute specifies that red is selected in the color control by default.

```xml
<Option name="colorControl" defaultValue="red" inputType="color">
  Choose a color
</Option>
```

If you select Show a date picker from the combobox control, the conditions for this dependency are met:

```xml
<Dependency condition="$comboShowChange == 'valueShowDate'">
  <Target action="hide" conditionResult="true" option="colorControl"/>
  <Target action="show" conditionResult="true" option="dateControl"/>
  <Target action="hide" conditionResult="true" option="sliderControl"/>
</Dependency>
```

The date picker control appears in the user interface.

```xml
<Option name="dateControl" inputType="datepicker" format="monyy7.">
  Choose a date:
</Option>
```

**Using a Value to Enable or Disable Additional Options**

This example is similar to using a value to show or hide options. However, in this example, the options are already visible in the user interface. Selecting a value from the combobox control enables these additional options, so the user can set these options.

In the Advanced Task if you select the Enable/Disable Options radio button, the values in the combobox are determined by these lines of code:

```xml
<Option name="comboEnableChange" defaultValue="valueEnableColor" inputType="combobox" width="100%">Combobox:
</Option>
<Option name="valueEnableColor" inputType="string">Enable the color selector</Option>
<Option name="valueEnableDate" inputType="string">Enable the date picker</Option>
<Option name="valueEnableSlider" inputType="string">Enable the slider control</Option>
```

The dependency code for the Enable/Disable Options radio button (referred to as radioEnableDisable in the XML) shows that when this radio button is selected, five options (labelEnableChange, comboEnableChange, colorControl, dateControl, and sliderControl) appear in the user interface:
Here is the dependency code:

```xml
<Dependency condition="$radioChoice == 'radioEnableDisable'">
  <Target action="show" conditionResult="true" option="labelEnableChange"/>
  <Target action="show" conditionResult="true" option="comboEnableChange"/>
  <Target action="hide" conditionResult="true" option="labelShowChange"/>
  <Target action="hide" conditionResult="true" option="comboShowChange"/>
  <Target action="show" conditionResult="true" option="colorControl"/>
  <Target action="show" conditionResult="true" option="dateControl"/>
  <Target action="show" conditionResult="true" option="sliderControl"/>
  <Target action="hide" conditionResult="true" option="labelShowSet"/>
  <Target action="hide" conditionResult="true" option="comboSetChange"/>
  <Target action="hide" conditionResult="true" option="checkboxCheckUncheck"/>
</Dependency>
```

Here is the resulting user interface:

- **GROUP OF CONTROLS**
  - Select the type of dependency to see an example of:
    - Show / Hide Options
    - Enable / Disable Options
    - Set Values
  
  Change the combobox value to see options become enabled or disabled.

  **Combobox:**
  - Enable the color selector

  Choose a color:

  Choose a date:

  Slider with buttons

The user interface shows the colorControl (labeled **Choose a color**), the dateControl (labeled **Choose a date**), and the sliderControl (labeled **Slider with buttons**) options. However, only the **Choose a color** option is enabled because **Enable the color selector** option is selected in the **Combobox** control, which means this dependency code is met:

```xml
<Dependency condition="$comboEnableChange == 'valueEnableColor'">
  <Target action="enable" conditionResult="true" option="colorControl"/>
  <Target action="disable" conditionResult="true" option="dateControl"/>
  <Target action="disable" conditionResult="true" option="sliderControl"/>
</Dependency>
```

If you select **Enable the date picker** from the combobox control, the conditions for this dependency are met:

```xml
<Dependency condition="$comboShowChange == 'valueShowDate'">
  <Target action="disable" conditionResult="true" option="colorControl"/>
  <Target action="enable" conditionResult="true" option="dateControl"/>
  <Target action="disable" conditionResult="true" option="sliderControl"/>
</Dependency>
```
The date picker control is enabled in the user interface.

```xml
<Option name="dateControl" inputType="datepicker" format="monyy7.">
    Choose a date:
</Option>
```

The color and slider controls are still visible in the user interface, but they are disabled.

### Using a Value to Set the Value of Another Option

In the Advanced Task if you select the **Set Values** radio button, the values in the combobox are determined by these lines of code:

```xml
<Dependency condition="$comboSetChange == 'valueSetCheck'">
    <Target action="set" conditionResult="true" option="checkboxCheckUncheck" property="value" value="1"/>
    <Target action="set" conditionResult="false" option="checkboxCheckUncheck" property="value" value="0"/>
</Dependency>
```

As a result, the **Checkbox** option is selected in the user interface. If you select the **Uncheck the checkbox** option from the combobox control, the conditionResult is false, and the **Checkbox** option is not selected.
Chapter 6

Working with the Requirements Element

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, BVAR, 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>
  </Roles>
</Metadata>
```
<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>
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>$sasOS</td>
<td>The operating system for the SAS server.</td>
</tr>
<tr>
<td>$sasVersion</td>
<td>The version of the SAS server.</td>
</tr>
<tr>
<td>$MathTool</td>
<td>The Java object for the Apache Velocity MathTool. For more information, see “Floating Point Math” on page 72.</td>
</tr>
<tr>
<td>$CTMUtil</td>
<td>This tool holds a Java object that provides common utility methods for the common task models.</td>
</tr>
<tr>
<td>$CTMMathUtil</td>
<td>This tool holds a Java object that provides access to basic math utilities.</td>
</tr>
</tbody>
</table>

**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. SPCT contains a value between 1 and 100.

```xml
<Options>
  <Options name="PCT" defaultValue="10" inputType="inputtext">Value used
```
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(library-name, table-name)</td>
<td>Drops the specified table. Specifying the library name is optional.</td>
</tr>
<tr>
<td>%web_open_table(library-name, table-name)</td>
<td>Opens the specified table. Specifying the library name is optional.</td>
</tr>
<tr>
<td>%web_open_file(filename, type)</td>
<td>Opens the specified file with the specified MIME type.</td>
</tr>
<tr>
<td>%web_open_url(url)</td>
<td>Opens the specified URL.</td>
</tr>
</tbody>
</table>

**Working with the DataSource Element in Velocity**

*About the DataSource Element*

You can specify only 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.

If you reference the name of the data source in Velocity (for example, $datasource), you see the value of the active Library.Table.

*Note:* If the name contains spaces or special characters, the return value could be n-literalized.

You can use the `columnExists`, `getLibrary`, `getRowsCount`, and `getTable` methods to get more information about the data source. For more information, see Appendix 1, “Common Utilities for CTM Writers,” on page 97.
**columnExists Method**

<table>
<thead>
<tr>
<th>Short Description</th>
<th>Determines whether the specified value already exists as the name of a column in the data source.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>input</td>
</tr>
<tr>
<td></td>
<td>the input string that you want to check to see whether it exists.</td>
</tr>
<tr>
<td>Return Value</td>
<td>Returns a Boolean value that specifies whether the column already exists.</td>
</tr>
</tbody>
</table>

**Example**

```xml
<DataSource name="DATASOURCE">
  <Roles>
    <Role name="analysisVariables" type="A" maxVars="0" minVars="0">
      Analysis variables:
    </Role>
  </Roles>
</DataSource>

#if ($DATASOURCE.columnExists("MAKE")) ... #end /* If data set is Sashelp.Cars, the return value is true. */
```

---

**get Method**

<table>
<thead>
<tr>
<th>Short Description</th>
<th>Provides information about the datasource control. This method takes a string parameter that accepts one of these values:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• table – the name of the table that is used for the data source</td>
</tr>
<tr>
<td></td>
<td>• value – the name of the data source in Library.Table format.</td>
</tr>
<tr>
<td>Return Value</td>
<td>Returns the attributes of the datasource control.</td>
</tr>
</tbody>
</table>

**Note:** The return values are not in n-literal form if the table contains special characters or spaces. For n-literal notation, use the `getTable()` method or use the Velocity variable value.

**Example**

```xml
<DataSource name="DATASOURCE">
  <Roles>
    <Role name="analysisVariables" type="A" maxVars="0" minVars="0">
      Analysis variables:
    </Role>
  </Roles>
</DataSource>

$DATASOURCE.get('table'); /* if dataset is SASHELP.CARS, return value is CARS*/
$DATASOURCE.get('value'); /* if dataset is SASHELP.CARS, return value is SASHELP.CARS*/
```
**getDistinctCount Method**

To use this method, specify `fetchDistinct = "true"` in the Role element. For more information, see “About the Roles Element” on page 12.

As shown in the example, you can use this function in your dependency code and your Velocity code to control other behaviors.

| Short Description | Returns the count of distinct values for a given column name for the current data source.  
                          Note: For optimal performance, the maximum number of distinct values is 100. |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Return Value</td>
<td>Returns the number of distinct values. If there are no distinct values or the distinct values are not available, the return value is –1.</td>
</tr>
</tbody>
</table>

**Example**

```velocity
<DataSource name="DATASOURCE">
  <Roles>
    <Role name="VAR" fetchDistinct="true" type="A" maxVars="0" minVars="0">
      Analysis variables:</Role>
    <Roles>
  </DataSource>

  <Dependencies>
    <Dependency condition="$VAR.size() &gt; 0 &amp;&amp; $DATASOURCE.getDistinctCount($VAR[0]) &gt; 0">
      <Target action="show" conditionResult="true" option="targetComboBox"/>
      <Target action="hide" conditionResult="false" option="targetComboBox"/>
    </Dependency>
  </Dependencies>

  #if ($VAR.size() &gt; 0 &amp;&amp; $DATASOURCE.getDistinctCount($VAR[0]) &gt; 0  ... #end
```

**getDistinctValues Method**

To use this method, specify `fetchDistinct = "true"` in the Role element. For more information, see “About the Roles Element” on page 12.

| Short Description | Returns an array of the distinct values for a given column name for the current data source.  
                          Note: For optimal performance, the maximum number of distinct values is 100. |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Return Value</td>
<td>Returns the set of distinct values in an array. If there are no distinct values or the column is not available, this method returns an empty array.</td>
</tr>
</tbody>
</table>
Example

```xml
<DataSource name="DATASOURCE">
    <Roles>
        <Role name="VAR" fetchDistinct="true" type="A" maxVars="0" minVars="0">
            Analysis variables:</Role>
        <Roles>
    </DataSource>
```

//if DATASOURCE is SASHELP.CLASS and the SEX variable is assigned to VAR, the return value from Velocity is 'F M'

```velocity
#if ($VAR.size() > 0)
    #foreach ($item in $DATASOURCE.getDistinctValues($VAR[0]))
        $item
    #end
#end
```

---

**getLibrary Method**

<table>
<thead>
<tr>
<th>Short Description</th>
<th>Returns the name of the library for the data source.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return Value</td>
<td>Returns a string that contains the name of the library for the data source.</td>
</tr>
</tbody>
</table>

Example

```xml
<DataSource name="DATASOURCE">
    <Roles>
        <Role name="analysisVariables" type="A" maxVars="0" minVars="0">
            Analysis variables:</Role>
        <Roles>
    </DataSource>
```

```velocity
$DATASOURCE.getLibrary() /* If data set is Sashelp.Cars, the return value is Sashelp. */
```

---

**getRowsCount Method**

<table>
<thead>
<tr>
<th>Short Description</th>
<th>Returns the number of rows in the data source.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return Value</td>
<td>Returns a value of 0 or greater if the data source is available. If this information is not available, –1 is the return value. For example, in SAS Studio when the selected data source is a data view, the row count is not available, so the return code for this function is –1.</td>
</tr>
</tbody>
</table>

Example

```xml
<DataSource name="DATASOURCE">
    <Roles>
        <Role name="analysisVariables" type="A" maxVars="0" minVars="0">
            Analysis variables:</Role>
        <Roles>
    </DataSource>
```

```velocity
#if (($DATASOURCE.getCount() > 0) ... #end /* If data set is Sashelp.Cars, the return value is 19. */
```
**getTable Method**

<table>
<thead>
<tr>
<th>Short Description</th>
<th>Returns the table name for the data source.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return Value</td>
<td>Returns a string that contains the table name for the data source.</td>
</tr>
</tbody>
</table>

**Example**

```xml
<DataSource name="DATASOURCE">
  <Roles>
    <Role name="analysisVariables" type="A" maxVars="0" minVars="0">
      Analysis variables:</Role>
  <Roles>
</DataSource>

$DATASOURCE.getTable() /* If data set is Sashelp.Cars, the return value is Cars. */
```

*Note:* If the table name contains spaces or special characters, the method returns the n-literal form of the string. If you do not want the n-literal form, use the `get('table')` method.

**getWhereClause Method**

To use this method, you must specify `where = "true"` in the `DataSource` element. Any filter that is added to a data source can affect any distinct controls (such as the `getDistinctCount` and `getDistinctValues` methods) that are associated with the same data source.

<table>
<thead>
<tr>
<th>Short Description</th>
<th>Returns the filter of the currently assigned data source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return Value</td>
<td>Returns a string that contains the filter of the currently assigned data source</td>
</tr>
</tbody>
</table>

**Example**

```xml
<DataSource name="DATASOURCE" where="true">
  <Roles>
    <Role name="analysisVariables" type="A" maxVars="0" minVars="0">
      Analysis variables:</Role>
  <Roles>
</DataSource>

$DATASOURCE.getWhereClause() /* If data set is Sashelp.Cars, the return value is the filter value that the user specifies. */
```

**getDataType Method**

<table>
<thead>
<tr>
<th>Short Description</th>
<th>Returns the type of data set. This value corresponds to the ‘typemem’ value in Sashelp.Vtable.</th>
</tr>
</thead>
</table>

### Return Value

Is the type of data set. This method defaults to null if the value is not available.

### Example

```
<DataSource name="DATASOURCE" where="true">
  <$DATASOURCE.getDataType()>
</DataSource>
```

---

## Working with Role Elements in the Velocity Code

### How Role 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. In the `Role` element, the `minVars` and `maxVars` attributes specify how many variables can be assigned to a specific role. Because roles can have 1 to $n$ number of variables, the corresponding Velocity variable is an array. The syntax for an array is `$variable-name[index-number]`. In this example, `$subsetRole` is the Velocity variable for the `Subset by` role (which is defined in the metadata):

```
proc rank data=$dataset (where=($subsetRole[0]="$filterValue")) descending
```

### get Method

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>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 <code>Numeric</code> or <code>Char</code>.</td>
</tr>
<tr>
<td>value</td>
<td>Specifies the name of the variable.</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.

```
<DataSources>
```
<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>

  Note: The return string obtained by looping through the Role’s item array is n-literalized. Use the get('value') method to obtain a value that is not n-literalized.

---

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 returns the character N if the Print row numbers check box is selected.

<Options>
  <Option name="PRINTNUMROWS" defaultValue="1"
    inputType="checkbox">Print row numbers</Option>
</Options>

<CodeTemplate>
  <![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`.

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

combobox

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 returns 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>
<UI>
  <Container option="OPTIONSTAB">
    <OptionChoice option="HEADING">
      <OptionItem option="default"/>
      <OptionItem option="horizontal"/>
      <OptionItem option="vertical"/>
    </OptionChoice>
  </Container>
</UI>
<CodeTemplate>
  <![CDATA[
  #if ($HEADING && ($HEADING !="default"))
    HEADING=$HEADING
  #end
  ]]>}
</CodeTemplate>
```

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

```xml
<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)`.

```xml
<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" source="responseVariable">Event of interest:</Option>
</Options>
<CodeTemplate>
  <![CDATA[
    #foreach( $item in $responseVariable ) $item (event='$referenceLevelCombo')#end
  ]]>"
</CodeTemplate>
```

### dualselector

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.

```xml
<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 returns 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".

mixedeffects

The Velocity variable that holds the output of the mixed effects control is a data structure containing two members, modelSummaryValues and mixedEffectsModels.

The modelSummaryValues member summarizes the user’s interaction with the mixed effects control. Here are members for the mixed effects control:

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>randomEffectsSetCount</td>
<td>Specifies the number of random effects model sets that were created.</td>
</tr>
<tr>
<td>repeatedEffectsSetCount</td>
<td>Specifies the number of repeated effects model sets that were created.</td>
</tr>
<tr>
<td>fixedEffectsCount</td>
<td>Specifies the number of fixed effects that were created.</td>
</tr>
<tr>
<td>fixedContinuousMainEffectsCount</td>
<td>Specifies the number of main fixed effects that were created for a continuous variable.</td>
</tr>
<tr>
<td>Member</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>fixedClassificationMainEffectsCount</td>
<td>Specifies the number of main fixed effects that were created for a classification variable.</td>
</tr>
<tr>
<td>fixedInterceptValue</td>
<td>Specifies the value of the intercept of the fixed effects model set. Valid values are \textit{true}, \textit{false}, or null.</td>
</tr>
<tr>
<td>fixedModelsetInvalidStateCount</td>
<td>Specifies the number of fixed effects model sets with an invalid context.</td>
</tr>
<tr>
<td>randomModelsetInvalidStateCount</td>
<td>Specifies the number of random effects model sets with an invalid context.</td>
</tr>
<tr>
<td>repeatedModelsetInvalidStateCount</td>
<td>Specifies the number of repeated effects model sets with an invalid context.</td>
</tr>
<tr>
<td>meansModelsetInvalidStateCount</td>
<td>Specifies the number of means effects model sets with an invalid context.</td>
</tr>
<tr>
<td>zeroInflatedEffectsCount</td>
<td>Specifies the number of zero-inflated effects that were created.</td>
</tr>
<tr>
<td>zeroInflatedEffectsSetCount</td>
<td>Specifies the number of zero-inflated effects model sets that were created.</td>
</tr>
<tr>
<td>zeroInflatedModelsetInvalidStateCount</td>
<td>Specifies the number of zero-inflated effects model sets with an invalid context.</td>
</tr>
</tbody>
</table>

The \texttt{mixedEffectsModels} member describes the detailed results of the interactions with the mixed effects control. This member is an array of models created by the user. The models are in the order in which they were created.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>emtype</td>
<td>Specifies the type of model.</td>
</tr>
<tr>
<td>intercept</td>
<td>Specifies whether the intercept is visible to the user. Valid values are \textit{true}, \textit{false}, or null.</td>
</tr>
</tbody>
</table>
### Member Description

**modelEffects**
- Specifies the array of effects that create this model.
- **effectType**: main, interaction, or nested
- **effectName**: the display name
- **memberSet1**: the members for this effect
- **memberSet2**: for nested effects, the inner members within the outer members

**groupEffect**
- Contains information about the group effect if one is defined. Otherwise, the value is null.
- **effectType**: main, interaction, or nested
- **effectName**: the display name
- **memberSet1**: the members for this effect
- **memberSet2**: for nested effects, the inner members within the outer members

**subjectEffect**
- Contains information about the subject effect if one is defined. Otherwise, the value is null.
- **effectType**: main, interaction, or nested
- **effectName**: the display name
- **memberSet1**: the members for this effect
- **memberSet2**: for nested effects, the inner members within the outer members

**covarianceStructures**
- Specifies the array that contains the covariance structure, if one is defined. Only one covariance structure can be defined. If no structure is defined, the array is empty.
- **csType** specifies the type of covariance structure.
- **csParameterValues** specifies the parameter value for the covariance structure. If no parameter value is needed, **csParameterValues** is set to null.

The following Velocity code does not generate SAS code. The purpose of this code is to demonstrate how to parse the Velocity structure for mixed effects.

```Velocity
<CodeTemplate>
<![CDATA[

/* ======== MEC Summary Values START ========*/
#if ( $mixedEffects.modelSummaryValues )
Random Effects Set Count: $mixedEffects.modelSummaryValues.randomEffectsSetCount;
Repeated Effects Set Count: $mixedEffects.modelSummaryValues.repeatedEffectsSetCount;
Fixed Effects Count: $mixedEffects.modelSummaryValues.fixedEffectsCount;
Fixed Continuous Main Effects Count: $mixedEffects.modelSummaryValues.fixedContinuousMainEffectsCount;
Fixed Classification Main Effects Count: $mixedEffects.modelSummaryValues.fixedClassificationMainEffectsCount;
#else ( $mixedEffects.modelSummaryValues.fixedInterceptValue )
#endif
]]>
```

84  Chapter 7 • Understanding the Code Template
Fixed Intercept Value: $mixedEffects.modelSummaryValues.fixedInterceptValue;
#else
Fixed Intercept Value: null;
#end

Fixed Classification Main Effects Count:
$mixedEffects.modelSummaryValues.fixedClassificationMainEffectsCount;
/* Model set invalid state count: */
Fixed: $mixedEffects.modelSummaryValues.fixedModelsetInvalidStateCount;
Random: $mixedEffects.modelSummaryValues.randomModelsetInvalidStateCount;
Repeated: $mixedEffects.modelSummaryValues.repeatedModelsetInvalidStateCount;
Means: $mixedEffects.modelSummaryValues.meansModelsetInvalidStateCount;
Zero-Inflated: $mixedEffects.modelSummaryValues.zeroInflatedModelsetInvalidStateCount;
#else
/* No summary values found. */
#end

/* ======== MEC Summary Values END =========
* ======== MEC models START ========*/
#if ( $mixedEffects.mixedEffectsModels )
#foreach( $model in $mixedEffects.mixedEffectsModels )
/* **** Begin $model.emtype effects model ****
*/
#if ( $model.intercept == "True" )
/* This model has an intercept. */
#elseif ( $model.intercept == "False" )
/* This model has no intercept. */
#end
/* User has generated $model.modelEffects.size() model effects */
#if ( $model.modelEffects.size() > 0 )
foreach( $modelEffect in $model.modelEffects )
## if the effectType is 'nested', then this is a nested effect
#if ( $modelEffect.effectType == 'nested' )
$velocityCount $modelEffect.effectType effect: #foreach( $subitem1 in $modelEffect.memberSet1 )$subitem1#if($velocityCount <$modelEffect.memberSet1.size())*#end#end(#foreach($subitem2 in $modelEffect.memberSet2)$subitem2#if($velocityCount <$modelEffect.memberSet2.size())*#end#end);
## handle 'main' or 'interaction' effects
#else
$velocityCount $modelEffect.effectType effect: #foreach( $subitem in $modelEffect.memberSet1 )$subitem#end
#end
#else
/* User hasn't generated any model effects yet */
#end
#if ( $model.subjectEffect )
/* user has generated a subject effect */
#set ( $modelEffect = $model.subjectEffect )
## if the effectType is 'nested', then this is a nested effect
#if ( $modelEffect.effectType == 'nested' )
$modelEffect.effectType effect: #foreach( $subitem1 in $modelEffect.memberSet1 )$subitem1#end
#else
/* No summary values found. */
#end

How the Options Elements Appear in the Velocity Code
# handle 'main' or 'interaction' effects
#else
$modelEffect.effectType effect: #foreach( $subitem in $modelEffect.memberSet1 )$subitem#if($velocityCount < $modelEffect.memberSet1.size())*#end#end;
#endif
#endif
#if ( $model.groupEffect )
/* user has generated a group effect */
#set ( $modelEffect = $model.groupEffect )
#if ( $modelEffect.effectType == 'nested' )
$modelEffect.effectType effect: #foreach( $subitem1 in $modelEffect.memberSet1 )$subitem1#if($velocityCount < $modelEffect.memberSet1.size())*#end#end(#foreach($subitem2 in $modelEffect.memberSet2)$subitem2#if($velocityCount < $modelEffect.memberSet2.size())*#end#end);
#elif ( $model.covarianceStructures )
/* User has generated $model.covarianceStructures.size() covariance structures */
#if ( $model.covarianceStructures.size() > 0 )
#foreach( $covStruct in $model.covarianceStructures )
$velocityCount $covStruct.csType parameters: [#foreach( $subitem in $covStruct.csParameterValues )$subitem#if($velocityCount < $covStruct.csParameterValues.size()),#end#end],
#endif
#else
/* User hasn't generated any covariance structures yet */
#endif
#endif
/* **** End $model.emtype effects model **** */
#endif
#else
/* User hasn't included any effects models yet */
#endif
/* ======== MEC models END =========

* ======== MEC DEBUG START ========
 modelSummaryValues property */
contents: $mixedEffects.modelSummaryValues;
/* Number of effects models: $mixedEffects.mixedEffectsModels.size() */
* mixedEffectsModels property */
The modelbuilder control will be removed in a later release. All SAS Studio tasks that used the modelbuilder control have been revised to use the mixedeffects control.

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.

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

How the Options Elements Appear in the Velocity Code
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.

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

numbertext

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

This example returns 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**.

```xml
<Options>
  <Option name="AMT" defaultValue="1" minValue="0" maxValue="100"...
The Velocity variable for the `numstepper` input type holds the string specified in the number control box.

This example returns 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"`.

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

**optiontable**

The Velocity variable for the option table holds information about the option’s current state. This variable has two members, `rows` and `columns`.

The `rows` member accesses the contents of the option table in an array of rows. The following information can be retrieved from each item in a row:

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
</table>
| values | Specifies an array of values for each row. Each `values` array element contains these members:  
| • `id` – the ID of the row, which correlates to the row number. The row numbers start at 1.  
| • the column name as defined in the `Column` element. |

The `columns` member accesses the contents of the option table in an array of columns. The following information can be retrieved from each item in a column:
<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column name as defined in the <strong>Column</strong> element</td>
<td>Specifies the information specific to that column. This structure has these members:</td>
</tr>
<tr>
<td>• <strong>values</strong> – an array of the current values.</td>
<td></td>
</tr>
<tr>
<td>• <strong>isValid</strong> – a Boolean value (1 or 0) that indicates whether the column is currently valid.</td>
<td></td>
</tr>
<tr>
<td>• <strong>numValues</strong> – the current number of values for this column.</td>
<td></td>
</tr>
</tbody>
</table>

This code uses the metadata that you specified for the **OptionTable** element in Chapter 3, “Working with the Metadata Element,” on page 11. This code does not generate SAS code. Instead, it demonstrates how to parse the Velocity structure of the option table.

```Velocity
<CodeTemplate>
<!CDATA[
/* Print option table content - rows array */
$optionTable.rows;

/* Iterate over each row to obtain values */
#foreach($item in $optionTable.rows.values)
row[$item.id] = $item
#end

/* Print option table content - this time using a columns array */
$optionTable.columns;

/* Iterate over each column to obtain values */
#foreach($columnName in $optionTable.columns.keySet())
column[$columnName] = $optionTable.columns[$columnName]
#end

/* cell[row][column_name]*/
/* Row 2, Column TextBox */
$optionTable.rows.values[1].colTextBox;

/* cell[column][row]*/
/* Column ComboBox, Row 3 */
$optionTable.columns["colComboBox"].values[2];
]]>
</CodeTemplate>
```

**passwordtext**

The **passwordtext** variable holds the password that is currently entered in the password control. The password is SAS002 encoded.
This example returns the string `%put PASSWORD=` and then the value of the password text control.

```xml
<Options>
  <Option name="pswd" inputtype="passwordtext">Enter password:</Option>
</Options>
<CodeTemplate>
  <![CDATA[
  %put PASSWORD = $pswd;
  ]]> 
</CodeTemplate>
```

If you entered a password before running this code, here is an example of the possible output:

```
%put PASSWORD = {SAS002}05C6153E3289264E53C70981;
```

**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"`.

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

**sasserverpath**

The Velocity variable for the `sasserverpath` control is a data structure that contains one or two members.

- If the `sasserverpath` control is a folder selector, the Velocity variable holds only the path member.
- If the `sasserverpath` control is a file selector, the Velocity variable holds both the path and file member.
<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Specifies the path of the folder or file selected.</td>
</tr>
<tr>
<td>name</td>
<td>Specifies the name of the selected file for a file selector or the name of the selected project for a project selector.</td>
</tr>
<tr>
<td></td>
<td><em>Note:</em> This member does not exist for a folder selector.</td>
</tr>
<tr>
<td>fullPath</td>
<td>Specifies the complete path of the selected file for the file selector or the complete path of the selected project for the project selector.</td>
</tr>
<tr>
<td></td>
<td>The full path is the combination of the path and name attributes.</td>
</tr>
<tr>
<td></td>
<td><em>Note:</em> This member does not exist for a folder selector.</td>
</tr>
</tbody>
</table>

This example shows how to specify a folder, file, and project selector in the Velocity code:

```velocity
<Options>
    <Option name="folderSelector" inputType="sasserverpath" pathType="folder">Select a folder:</Option>
    <Option name="fileSelector" inputType="sasserverpath" defaultFileName="myProgramFile" pathType="file" defaultExtension="sas">Select a new or existing file:</Option>
    <Option name="projectSelector" inputType="sasserverpath" pathType="project" defaultName="RPM">Select a project folder:</Option>
</Options>

<CodeTemplate>
  <![CDATA[
%let folderPath=$folderSelector.path;
%let filePath=$fileSelector.path;
%let fileName=$fileSelector.name;
%let projectPath=$projectSelector.path;
$let projectFolderName=$projectSelector.name;
]]>
</CodeTemplate>
```

**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.

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

...  
</Container>
</UI>
<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 returns the string `datalabelattrs=(size=n)`, where `n` is the value of the **Label Font Size** option. If the value of the **Label Font Size** option is 10, the output is `datalabelattrs=(size=10)`.  

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

**string**

A Velocity variable is created for the string input type. Here is an example:

<CodeTemplate>
<![CDATA[
%put string=$str;
]]>
</CodeTemplate>

**textbox**

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

In this example, the **splitLines** attribute is set to false, so newline characters are preserved in the Velocity object.

<CodeTemplate>
<![CDATA[

```
If the user entered a phrase with a newline character in the text box, that newline character is preserved. Here is an example. In the text box, you entered this phrase:

Hello
World

Here is the resulting Velocity code:

```velocity
%put Text entered: 'Hello
World';
```

In this example, the `splitLines` attribute is set to true, so the Velocity variable is an array of each line.

```velocity
<![CDATA[
#set($line = 1)
#if ( $text2.size() > 0 )
  #foreach( $item in $text2 )
    %put Text line $line: $item;
    #set($line = $line+1)
  #end
#end
]]>
```

Now if you enter

Hello
World

in the text box, here is the resulting Velocity code:

```velocity
%put Text line 1: Hello;
%put Text line 2: World;
```

`validationtext`

The Velocity variable for the `validationtext` input type holds the string that was specified in the text box.

The following example returns the string `rho0=0` and the text in the Null hypothesis correlation option. If the user specifies 0, the resulting string is `rho0=0`.

```velocity
<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"
    invalidMessages="Enter a number greater than -1 and less than 1
      for the null hypothesis correlation"
    missingMessages="Enter a number greater than -1 and less than 1
      for the null hypothesis correlation"
    regExp="[-+]?(0|\.|\d*|\d+)(\d+)?">Null hypothesis correlation:</Option>
</Options>
```
Working with the OutputData Control in Velocity

About the OutputData Control

The Velocity variable for the outputdata control holds the string that appears in the text field. If you reference the name of the outputdata control in Velocity (for example, $outputdata), you see the Library.Table value that is currently set.

Note: If the name contains spaces or special characters, the return value might be in n-literal form. If you do not want the n-literal form, use the `get("value")` method.

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.

```
<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>

<UI>
  <Group option="outputGroup" open="true">
    <OptionItem option="outputDSName"/>
  </Group>
</UI>
```

```
<!CDATA[
  output = $outputDSName
]>
```

get Method

<table>
<thead>
<tr>
<th>Short Description</th>
<th>Provides information about the outputdata control. This method takes a string parameter that accepts one of these values:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• <strong>table</strong> – the name of the table that is used for the data source</td>
</tr>
<tr>
<td></td>
<td>• <strong>value</strong> – the name of the data source in <em>Library.Table</em> format.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Returns the attributes of the outputdata control.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Note: The return values are not in n-literal form if the table contains special characters or spaces. For n-literal notation, use the <code>getTable()</code> method or use the Velocity variable value.</td>
</tr>
</tbody>
</table>
Example  

```xml
<Options>
  <Option inputType="outputdata" name="OUTPUTDATA" required="true"
    defaultValue="WORK.Output2" indent="1" width="100">OutputData:</Option>
</Options>

$OUTPUTDATA.get("table"); /* if the target outputdata's value is WORK.CARS, return value is CARS. */
$OUTPUTDATA.get("value"); /* if the target outputdata's value is WORK.CLASS, return value is WORK.CLASS. */
```

### getLibrary Method

**Short Description**  
Returns the name of the library for the outputdata control.

**Return Value**  
Returns a string that contains the name of the library.

**Example**  

```xml
<Options>
  <Options inputType="outputdata" name="OUTPUTDATA" required="true"
    defaultValue="Work.Output2" indent="1" width="100">Output Data:</Option>
</Options>

$OUTPUTDATA.getLibrary();/* If the target outputdata value is WORK.CLASS, the return value is Work. */
```

### getTable Method

**Short Description**  
Returns the table name for the outputdata control.

**Return Value**  
Returns a string that contains the table name for the outputdata control.

**Example**  

```xml
<Options>
  <Options inputType="outputdata" name="OUTPUTDATA" required="true"
    defaultValue="Work.Output2" indent="1" width="100">Output Data:</Option>
</Options>

$OUTPUTDATA.getTable();/* if the target outputdata value is WORK.CLASS, the return value is CLASS. */
/* If the target outputdata value is Work.'my CLASS'n, the return value is 'my Class'n. */
```

**Note:** If the table name contains spaces or special characters, the method returns the n-literal form of the string. If you do not want the n-literal form, use the `get('table')` method.
Appendix 1
Common Utilities for CTM Writers

$CTMMathUtil Variable

The predefined $CTMMathUtil variable provides access to basic math utilities.

getMin Method

Short Description
Returns the smallest value of an array of doubles.

Syntax
Double getMin(ArrayList<Double> inputArray)

Parameter
input
an array of double values.

Return Value
Returns the double value that is the smallest in the input array. This function returns NaN if the inputArray is null or if an exception occurs while trying to process the array.

Example
#set($array = [1.0, 2.0, 3.0])
$CTMMathUtil.getMin($array)
/* double returned: 1.0 */
**getMax Method**

<table>
<thead>
<tr>
<th>Short Description</th>
<th>Returns the largest value of an array of doubles.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>Double getMax(ArrayList&lt;Double&gt; inputArray)</td>
</tr>
<tr>
<td>Parameter</td>
<td>input an array of double values.</td>
</tr>
<tr>
<td>Return Value</td>
<td>Returns the double value that is the largest in the input array. This function returns NaN if the inputArray is null or if an exception occurs while trying to process the array.</td>
</tr>
<tr>
<td>Example</td>
<td>#set($array = [1.0, 2.0, 3.0]) $CTMMathUtil.getMax($array) /* double returned: 3.0 */</td>
</tr>
</tbody>
</table>

**getSum Method**

<table>
<thead>
<tr>
<th>Short Description</th>
<th>Returns the smallest value of an array of doubles.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>Double getSum(ArrayList&lt;Double&gt; inputArray)</td>
</tr>
<tr>
<td>Parameter</td>
<td>input an array of double values.</td>
</tr>
<tr>
<td>Return Value</td>
<td>Returns the double value that is the sum of all the values in the input array. This function returns NaN if the inputArray is null or if an exception occurs while trying to process the array.</td>
</tr>
<tr>
<td>Example</td>
<td>#set($array = [1.0, 2.0, 3.0]) $CTMMathUtil.getSum($array) /* double returned: 6.0 */</td>
</tr>
</tbody>
</table>

**$CTMUtil Variable**

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

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

**Syntax**

```
String doubleQuoteString(String input)
```

**Parameter**

- `input`: the input string that you want to enclose in double quotation marks.

**Return Value**
Returns a string that represents the quoted value. Double quotation marks are added to the input string.

**Example**
```
#set( $input="Bar Chart" )
$CTMUtil.doubleQuoteString($input);
/* string returned: "Bar Chart" */
```

### quoteString Method

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

**Syntax**

```
String quoteString(String input)
```

**Parameter**

- `input`: the input string that you want to enclose in single quotation marks.

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

### isProductLicensed Method

**Short Description**
Checks to see whether a specific product is installed. A minimum or maximum version can be specified.

**Syntax**

```
Boolean isProductLicensed(int sasProductNumber, double minimumProductRelease, double maximumProductRelease)
```
### isProductLicensed Method

Parameter

- sasProductNumber
  - the product number to check.
- minimumProductRelease
  - the minimum version number for the product. To specify that there is no minimum version, enter `-1`.
- maximumProductRelease
  - the maximum version number for the product. To specify that there is no maximum version, enter `-1`.

Return Value

Returns a Boolean value of `true` if the product with the specified version is licensed. If the product is not licensed, the return value is `false`.

Example

```
$CTMUtil.isProductLicensed(0, -1, 9.3)
/* Boolean returned - true if Base SAS 9.3 or earlier is licensed; false if a version greater than SAS 9.3 is licensed. */
```

### toSASName Method

Short Description

Transforms a string so that it uses SAS naming conventions.

Syntax

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

Parameter

- input
  - the input string to transform.

Return Value

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

```
#set( $input="My Variable" )
$CTMUtil.toSASName($input);

/* string returned: "My Variable"n */
```
Recommended Reading

• *SAS Studio: Writing Your First Custom Task*
• *SAS Studio: Administrator’s Guide*
• *Getting Started with Programming in SAS Studio*
• *SAS Studio: User’s Guide*

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

SAS Books
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-0025
Fax: 1-919-677-4444
Email: sasbook@sas.com
Web address: sas.com/store/books
Index

**Special Characters**
- SCTMUtil 72
- SMathTool 72
- $sasOS 72
- $sasVersion 72

**A**
- Advanced Task 4
- Apache Velocity code 72
- MathTool 72
- predefined variables 72

**B**
- blank task 5

**C**
- checkbox controls 16
  - example of dependency 58
    - Velocity code 79
- Code Template element 72
- Code Template elements 1
- color controls 16
  - Velocity code 80
- columnExists methods 74
- combobox controls 17
  - example of dependency 64
    - Velocity code 80
- Container elements 49
  - controls
    - checkbox 16, 58, 79
    - color 16, 80
    - combobox 17, 80
    - datepicker 17, 80
    - distinct 18, 81
    - dualselector 20, 81
    - inputtext 22, 82
    - modelbuilder 87
    - multientry 27, 88
    - numbertext 29, 88
    - numstepper 30, 89
- outputdata 31
- radio 33, 59, 91
- select 35, 92
- slider 36, 93
- string 37, 93
- textbox 37, 93
- validationtext 38, 94
- CTK files 6, 7
- CTM files 7

**D**
- data source 11
- DataSource Element
  - Velocity code 73
- DataSources element 11
  - example 13
- datepicker controls 17
  - Velocity code 80
- dependencies
  - notes 55
- Dependencies elements 1, 53
  - examples 56
- disabling options 53, 62, 65
- distinct controls 18
  - Velocity code 81
- distinct values 75
- dualselector controls 20
  - Velocity code 81

**E**
- editing tasks 3
- elements
  - Code Template 1
  - Dependencies 1
  - Metadata 1
  - Options 1
  - Registration 1
  - Roles 1
  - UI 1
- enabling options 53, 62, 65
Index

F  
files 34, 91
folders 34, 91
My Tasks 3

G  
gGetDistinctCount methods 75
getDistinctValue method 75
getLibrary method 76, 96
getMax Method 98
getMin Method 97
getRowsCount method 76
getSum Method 98
getTable method 77, 96
getWhereClause method 77
grouping options 49

H  
hiding options 53, 61, 64

I  
inputtext controls 22
Velocity code 82

L  
licensing
product 99

M  
mathematical expressions 72
Metadata element 11
Metadata elements 1
methods
columnExists 74
gGetDistinctCount 75
gGetDistinctValue 75
getLibrary 76, 96
getMax 98
getMin 97
getRowsCount 76
getSum 98
table 77, 96
gGetWhereClause 77
quotestring 99
toSASName 100
mixedeffects controls
linking to effects 45
Velocity code 82
modelbuilder controls
Velocity code 87
multientry controls 27

N  
numbertext controls 29
Velocity code 82
numstepper controls 30
Velocity code 89

O  
options
disabling 53, 62, 65
enabling 53, 62, 65
grouping 49
hiding 53, 61, 64
setting 53, 63, 67
showing 53, 61, 64
Options elements 1, 14
Velocity code 79
OptionTable control 39
Velocity code 89
OptionTable elements 39
outputdata controls 31

P  
passwords 33, 90
predefined tasks 1
editing 3
product
licenses 99
projects 34, 91

Q  
quotestring Method 99

R  
radio controls 33
eexample of dependency 59
Velocity code 91
Registration element 9
eexample 10
Registration elements 1
Requirements element 69
eexample 69
return values
specifying 42
returnValue attribute 42
roles
assigning variables 12
Roles elements 1, 12
eexample 13
linking to data 44

S
Sample Task 3
SAS macros 73
sasserverpath 34, 91
select controls 35
  linking to data 43
  Velocity code 92
server path 34, 91
setting options 53, 63, 67
showing options 53, 61, 64
slider controls 36
  Velocity code 93
string controls 37
  Velocity code 93

T
Target elements 53
tasks
  about 1
  advanced 4
  blank 5
  creating 5
editing 3
folder shortcuts 7
name 9
requirements to run 69
sample 3
saving with default settings 6
sharing 7
testing 7
uploading 7
validating 7
textbox controls 37
  Velocity code 93
toSASName method 100

U
UI element 49
  example 50
UI elements 1

V
validationtext controls 38
  Velocity code 94
variables
  assigning to roles 12
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.