SAS® Studio 5.2: Developer's Guide to Writing Custom Tasks

SAS® Documentation
August 18, 2021

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2019. *SAS® Studio 5.2: Developer's Guide to Writing Custom Tasks*. Cary, NC: SAS Institute Inc.

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

August 2021

5.2-P1:webeditordg

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 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 12
Working with the Options Element 22

Chapter 4 / Working with the UI Element **79**
About the UI Element 79
Option References 79
Containers 80
Example: UI Element from Sample Task 81

Chapter 5 / Working with the Dependencies Element **85**
About the Dependencies Element 85
Notes on Dependencies 87
Creating Dependencies for Container and Group Elements 89
Using Radio Buttons as Targets of Dependencies 90
Example 1: Selecting a Check Box to Show a Group of Options 91
Example 2: Using Radio Buttons to Create Dependencies 93
Example 3: Using Combobox Controls 99
Using the OptionsDependencies Element 104

Chapter 6 / Working with the Requirements Element **107**
About the Requirements Element 107
Example: Using a Requirements Element for Roles 108

Chapter 7 / Understanding the Code Template **109**
About the Code Template 110
Using Predefined Velocity Variables 110
Working with the DataSource Element in Velocity 112
Working with Role Elements in the Velocity Code 117

How the Options Elements Appear in the Velocity Code 120
Working with the OutputData Control in Velocity 144

Appendix 1 / Common Utilities for CTM Developers **147**
$CTMMathUtil Variable 147
$CTMUtil Variable 149

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.

1

Introduction to the Common Task Model

About the SAS Studio Tasks . . . 1
Edit a Predefined Task . . . 3
Using Sample Tasks . . . 3
What Is the Difference between the Sample Task and the Advanced Task? . . . 3
View the Sample Task . . . 3
View the Advanced Task . . . 4
Create a Task . . . 5
Create a Task with Default Option Settings . . . 6
Validation Steps for the Task . . . 7
Testing a Task . . . 7
Sharing Tasks . . . 7
About CTM and CTK Files . . . 7
Accessing a Task Created by Another User . . . 7
Sharing a Task That You Created . . . 8

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 XML-based task model and the Velocity Template Language 2.0. 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:

Attribute	Description
`schemaVersion`	Specifies the schemaVersion associated with the task.
`fetchProductLicenses`	Specifies whether to retrieve the product license information. This attribute must be set to `true` if the `isProductLicensed` method is used in the `Code Template`, `Requirements`, or `Dependencies` elements. For more information, see "isProductLicensed Method" on page 150. The default value for this attribute is `false`.

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.

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 a condition that must be met in order for code to be generated for the entire task.

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** section.
2. Expand the folder that contains the task.
3. Right-click the name of the task and select **Copy as task template** ⇨ **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. Right-click the name of the task that you want to edit and select **Edit task template**. 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?

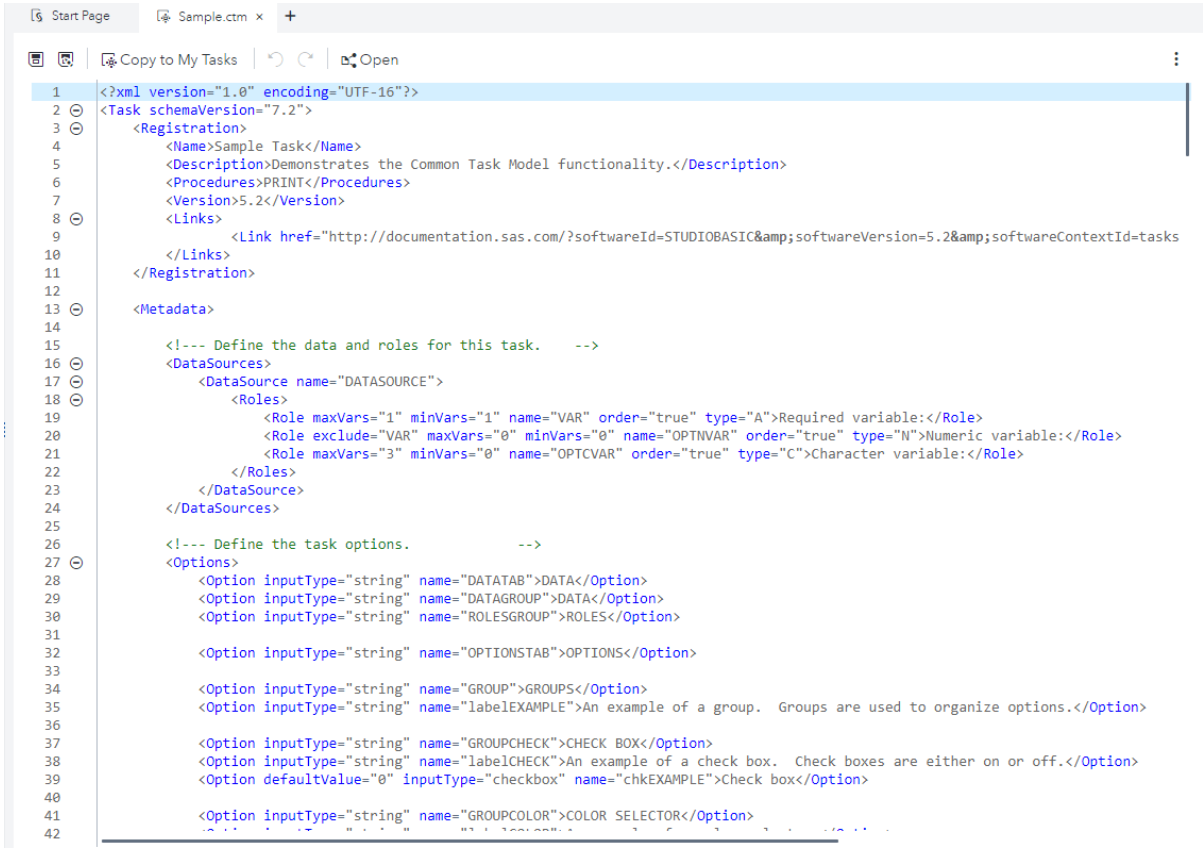
The Sample Task shows the controls that are available to you when writing a task. The Advanced Task shows some of the more complex functionality in the common task model. For example, the Advanced Task includes dependencies, the mixed effects builder, data linking, and return values.

View the Sample Task

To view the code of the Sample Task:

1. In the navigation pane, open the **Tasks** section.
2. Click  and select **Sample task**.

   The code for the Sample Task appears.

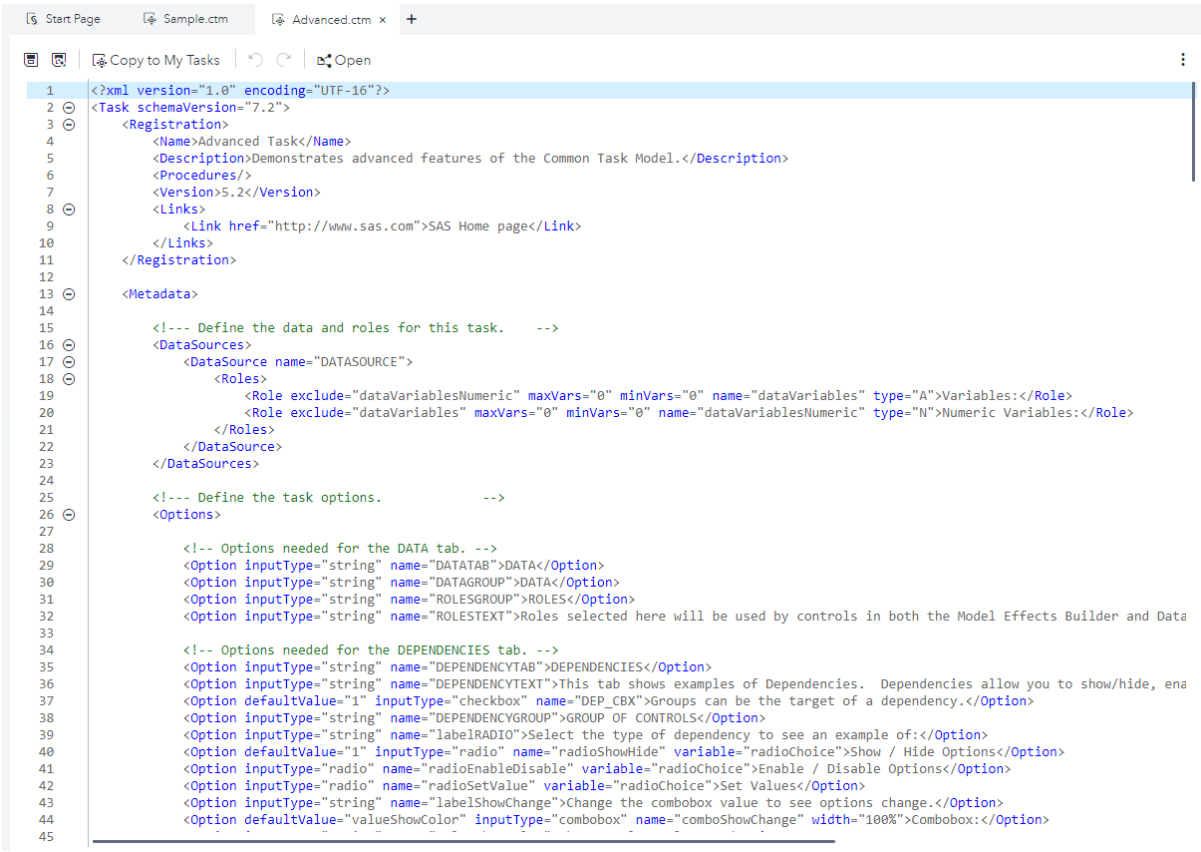


View the Advanced Task

To view the code for the Advanced Task:

1. In the navigation pane, open the **Tasks** section.
2. Click  and select **Advanced task**.

   The code for the Advanced Task appears.



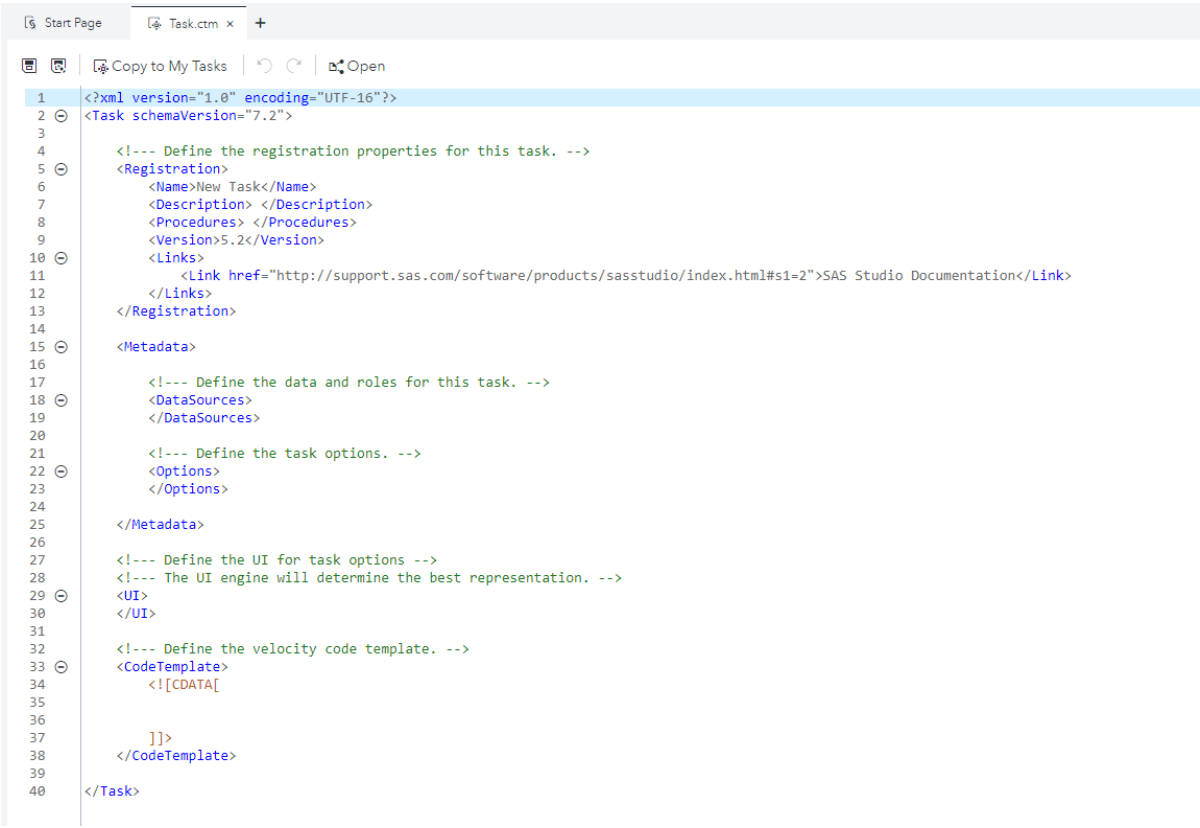
Create a Task

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

To create a task:

1. In the navigation pane, open the **Tasks** section.
2. Click  and select **Task**.

   The new task template 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 .

5 Enter a unique name for the task. The task is saved with the CTM file extension in your file system.

Create a Task with Default Option Settings

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

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

To save a task:

1 Click . The Save As window appears.

2 Select the location where you want to save the task file. You can save this file in the **Explorer** section or in your **My Tasks** folder. Specify a name for this file. For the file type, select **CTK Files (*.CTK)**. Click **Save**.

Note: In the **Tasks** section, you are still working with this task. If you save the task again, the CTK file in the **Explorer** section is updated.

Validation Steps for the Task

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

Testing a Task

To test your task, click **Open**. (Alternatively, you can press F3.) A new tab that contains the user interface for the task appears in your work area.

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 **Explorer** section and click . The Upload Files window appears.
3. Select the location of the files and click + 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 **Explorer** section. Click and select **Folder shortcut**. Enter the shortcut name and full path and click **OK**. The new shortcut is added to the list of folder shortcuts.

2

Working with the Registration Element

About the Registration Element 9
Example: The Registration Element from the Sample Task 10

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:

Element Name	Description
`Name`	The name of the task. This name is used throughout the application to represent the task.
`Description`	A description of the task. This text could appear in the task properties or in tooltips for the task.
`GUID`	A unique identifier for the task.
`Procedures`	A list of SAS procedures that are used by this task.
`Version`	The version of SAS Studio that is used to write this task. The task version is a simple number value such as 3.8 or 5.2. The version value determines the structure of the underlying Velocity code. The structure of the Velocity code can change between releases. Newer releases of SAS Studio are backward compatible. However, if you change the version value for

Element Name	Description
	an existing task, you could see changes to the underlying Velocity code.
`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:

```
<Registration>
   <Name>Sample Task</Name>
   <Description>Demonstrates the Common Task Model functionality.</
Description>
   <Procedures>PRINT</Procedures>
   <Version>5.2</Version>
   <Links>
      <Link href=http://documentation.sas.com/?
softwareId=STUDIOMID&amp;

softwareVersion=5.2&amp;softwareContextId=tasks&amp;requestor=inapp">
         SAS Studio documentation</Link>
   </Links>
</Registration>
```

3

Working with the Metadata Element

About the Metadata Element 11

Working with the DataSources Element 12
- About the DataSources Element 12
- Working with the Roles Element 13
- Working with the Filters Element 15

Working with the Options Element 22
- About the Option Element 22
- Supported Input Types 24
- Organizing Options into a Table Component 67
- Specifying a Return Value Using the returnValue Attribute 70
- Populating the Values for a Select Control from a Source Control 71
- Specifying a Help Message 76

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 3.1 *Attributes for the DataSources Element*

Attribute	Description
`name`	Specifies the name assigned to this role.
`defaultValue`	Specifies the default data source to use. If this value is not specified, SAS Studio uses the most recently used data source. If the most recently used data source cannot be determined, this value is empty. To change this behavior, use the `defaultValue` attribute. ■ If the `defaultValue` attribute is an empty string, the data source control is not initialized with a data source. ■ If the `defaultValue` specifies a data source (such as SASHELP.CARS), that data source appears in the control. If the data source cannot be found, an error is displayed.
`libraryEngineExclude`	Specifies the engine types that are not valid for the data source. The engine types should be a comma-separated list.
`libraryEngineInclude`	Specifies the engine types that are valid for the data source. Possible values include `CAS` and `V9`. Multiple engine types should be in a comma-separated list.
`readOnly`	Specifies whether the user can change the input data source. The default value is `false`, so the data can be changed. If the value is set to `true`, the value can be displayed but not changed.
`where`	Specifies whether a filter is allowed for the data. The default value is `false`, and the user cannot filter the task from the task interface.

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.

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 3.2 *Attributes for the Roles Element*

Attribute	Description
`name`	Specifies the name assigned to this role.
`defaultValue`	Specifies the default roles for this control. By default, the Role control is empty. Use the `defaultValue` attribute to specify a comma-separated list of variable names. If the variables do not exist in the current data source, the Role control does not show the missing variables.
`type`	Specifies the type of column that can be assigned to this role. Here are the valid values: `A` All column types are allowed. In the user interface, all columns are identified by the icon. `N` Only numeric columns can be assigned to this role. In the user interface, numeric columns are identified by the icon. `C` Only character columns can be assigned to this role. In the user interface, character columns are identified by the icon.
`minVars`	Specifies the minimum number of columns that must be assigned to this role. If `minVars="0"`, the role is optional. If `minVars="1"`, a column is required to run this task and a red asterisk appears next to the label in the user interface.

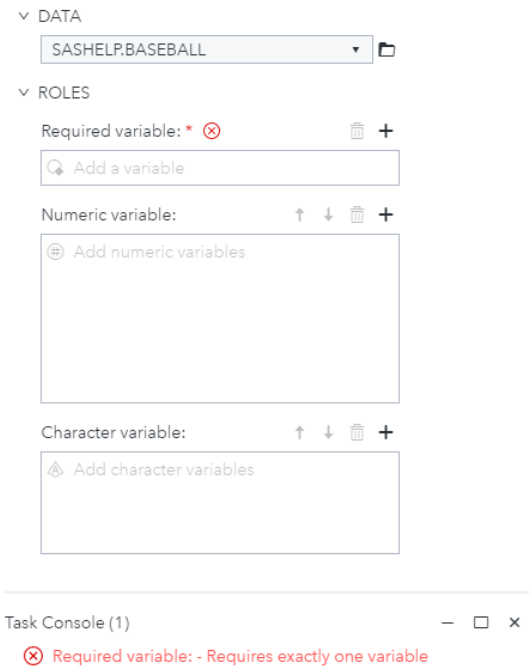
Attribute	Description
maxVars	Specifies the maximum number of columns that can be assigned to this role. If maxVars="0", users can assign an unlimited number of columns to this role.
exclude	Specifies the list of roles that are mutually exclusive to this role. If a column is assigned to a role in this list, the column does not appear in the list of available columns for this role.
order	Specifies that the user can order the columns that are assigned to this role. Valid values are true and false. If order="true", the user can use the up and down arrows in the user interface to modify the order.
fetchDistinct	Specifies whether to retrieve the distinct information for columns assigned to this role. The default value is false.
readOnly	Specifies whether the value can be edited. By default, the value is false, and the role can be edited. If this value is true, the current value is displayed but cannot be edited by the end user.

Example: DataSources and Roles Elements from the Sample Task

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

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

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



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 Filters Element

About the Filters Element

The `Filters` tag can have one or more `Filter` children elements. The `Filters` tag is a grouping mechanism for the individual `Filter` definitions. There are no attributes associated with the `Filters` tag.

Filters enable you to populate option controls with the values from a column in the data source. A simple filter might return values from a specific column in the data source. If this column is specified in the `Filter` element, the filter is static, which means that the end user of the task is forced to use that column. If the `Filter` element is written so that the user selects the column using a Role control, the filter is dynamic, which means that the end user can choose the column before running the task.

To create a more complex filter, add expressions. An expression specifies the condition that is applied to the filter to return specific values. The expression can use values from other options or columns to dynamically create the condition. For example, you could create a filter for the value of state. This filter defines that the value of the combobox control determines which observations in the data source are returned. At run time, the end user selects the state from the list of possible values in the combobox control.

Here is the XML hierarchy for a filter:

```
Filters (0..1)
   Filter(1..n)
      Column (1)
      Where (0...1)
         Expressions (1...n)
            Expression (1..n)
               Column (1)
               Value (1)
```

Filter Element

Each filter is a source of possible items. These items are values of a variable from the `DataSource` element. All of the values for a variable are returned unless a `Where` element is specified. Each `Filter` element must have a single `Column` element and can have a single `Where` element.

Table 3.3 *Attribute for Filter Element*

Attribute	Description
`name`	Specifies the name assigned to the filter.

These option types can use the `filter` attribute to populate their items from the `Filter`.

- combobox
- dualselector
- select

Column Element

The `Column` element specifies which column in the data source contains the values that you want to use. You can use the attributes for this element to define how the values are returned. For example, only the distinct values are returned or how the values are presented to the end user.

If you want to define the column to use in the task, use the `column` attribute, which creates a static column. The end user of the task has to use this column in the task. To enable users to select the column to use, use the `role` attribute. At run time, the user sees a Role control and can select the column to use.

Note: Either the `column` attribute or the `role` attribute must be specified. You cannot specify both attributes.

The label for the `Column` element can be a formatted value or a value from another column in the data source. To specify a static value, use the `labelColumn` attribute. The user can use only this column in the task. To enable the user to select the column from a Role control, use the `labelRole` attribute. By default, the formatted

value of the selected column is used. To overwrite this formatting, use the `format` attribute.

Note: If neither the `labelColumn` or `labelRole` are specified, the value of the `column` attribute or the `role` attribute is used as the label.

This table describes the attributes for the `Column` element.

Table 3.4 *Attribute for Column Element*

Attribute	Description
`column`	Specifies the column in the data source that provides the values for the option control.
`role`	Specifies the role that identifies the column in the data source that provides the values for the option control. If the role can accept multiple columns, only the first column is used.
`labelColumn`	Specifies the column in the data source that provides the labels for the values in the option control. The return value is the formatted value of the column.
`labelRole`	Specifies the role that identifies the column to use for the formatted value. The formatted value is used when the `display` attribute is set to either `formatted` or `both`. If the role can accept multiple columns, only the first column is used. If the `labeRole` is not specified, the formatted value is derived from the column specified by the `role` attribute.
`format`	Specifies the SAS format to apply to the label. This format overrides the format associated with the column specified by the `labelColumn`, `labelRole`, `column`, or `role` attribute.
`distinct`	Specifies whether to return only unique values. By default, this value is `true`, and only unique values are returned. When this attribute is set to `false`, all values are returned.
`display`	Determines how the values are displayed. Here are the valid values: ■ `formatted` uses the formatted value of the variable. If specified, the formatted value comes from the `labelRole` attribute. Otherwise, this value comes from the `role` attribute. ■ `unformatted` uses the unformatted value and ignores any column format defined in the data source.

Attribute	Description
	■ `both` displays both the formatted and unformatted values. If specified, the formatted value comes from the `labelRole` attribute. Otherwise, this value comes from the `role` attribute.
`max`	Specifies the maximum number of values to return. The default value is `100`.
`sortBy`	Specifies whether to sort by the unformatted or formatted values. By default, this value is `value`, which means the values are sorted by the unformatted values. If you specify `label`, the values are sorted by the formatted value.
`sortDirection`	Specifies whether to sort the data. Here are the valid values: ■ `none` does not sort the data. ■ `ascending` sorts the values in ascending order. ■ `descending` sorts the values in descending order.

Where Element

The optional `Where` element determines the values returned by the filter. Use the `Where` element to specify the values to return based on various options and how these variables compare to the specified variables in the data source.

When specified, the `Where` element must have a single `Expressions` element. There are no attributes associated with the `Where` element.

Expressions Element

The `Expressions` element must have one or more `Expression` children. The `Expressions` element is a grouping mechanism for the expression definitions. There are no attributes associated with the `Expressions` element.

When an `Expressions` element contains multiple `Expression` children, all the expressions must be satisfied for the row in the data source to provide values to the `Filter` element's output. The expressions are joined by the AND operator.

Expression Element

Each `Expression` element represents a part of the overall condition. For example, you could create an expression where the user sees only the values less than the number specified in the numbertext control.

Each `Expression` element must have a single `Column` element and a single `Value` element. The `Column` element specifies which column to use in the expression. Any column in the data source can be used.

Table 3.5 *Attributes for Expression Element*

Attribute	Description
`not`	Specifies which values to retrieve. By default, this value is `false`. When this attribute is set to `true`, the result of the condition is negated.
`operator`	Specifies the operator to use in the expression. Here are the valid values: `eq`, `ne`, `gt`, `ge`, `lt`, `le`, `contains`, `in`, and `like`. **Note:** Some operators work only with specific column or option types. For example, the contains and like operators work only with character variables and cannot be used with range values.

Note: If an option returns an ALL value to an expression. the expression is evaluated in this way:

- true if the operators are eq or in
- false if the operators are ne, gt, or lt
- an error is generated for all other operators

Column Element

The `Column` element specifies the variable to use in the comparison.

Table 3.6 *Attributes for Column Element*

Attribute	Description
`column`	Specifies the column in the data source that provides the values for the option.
`role`	Specifies the role that determines the column in the data source that provides the values for the option

Attribute	Description
	TIP Because the filter uses only the first variable, you might want to set `maxVars="1"` in the `Role` element.

Note: Specify either the `column` attribute or the `role` attribute, but not both.

Value Element

The `Value` element specifies the value to compare to the `Column` element for the expression.

Table 3.7 *Attributes for Value Element*

Attribute	Description
`value`	Enables the user to specify a string value.
`option`	Specifies the option that will provide the values for the comparison.

Options that provide lists or ranges of strings, numbers, or dates can be used if these values are supported by the operator and column type.

Example of a Filter

In this example, the user selects the value of Make from the Sashelp.Cars data set.

```
<Metadata>

    <DataSources>
       1 <DataSource name="CARDATA" defaultValue="SASHELP.CARS"
active="true">
          <Roles>
             <Role type="A" maxVars="1" order="true" minVars="1"
name="VAR">
         Choose a column to retrieve values for:</Role>
          </Roles>
         2 <Filters>
             <Filter name="filterMake">
                <Column column="make"/>
             </Filter>
             3 <Filter name="filterVar">
                <Column role="VAR" sortDirection="descending"/>
                <Where>
                   <Expressions>
```

```
                        4 <Expression operator="ge">
                            <Column column="cylinders"/>
                            <Value value="6"/>
                          </Expression>
                        5  <Expression operator="eq">
                            <Column column="make"/>
                            <Value option="comboMake"/>
                          </Expression>
                          6 <Expression operator="in">
                            <Column column="msrp"/>
                            <Value option="priceRange"/>
                          </Expression>
                       </Expressions>
                    </Where>
                 </Filter>
              </Filters>
           </DataSource>
    </DataSources>

    <!--- Define the task options.             -->
    <Options>

        <Option name="comboMake" defaultValue="default"
inputType="combobox"
      filter="filterMake">Select a car make:</Option>

        <Option name="priceRange" inputType="numericrange">
      Specify a price range:</Option>
        <Option name="OPTIONSTAB" inputType="string">OPTIONS</Option>
        <Option name="comboVar" defaultValue="default"
inputType="combobox"
      filter="filterVar">Select a value from Role:</Option>
        <Option name="message" inputType="string">This combobox
      shows values from the column chosen above.
      The values shown have been run through an expression where
      cylinders are greater than 6 and msrp is between the price
      range specified above.</Option>

    </Options>
</Metadata>

<UI>

    <Container option="OPTIONSTAB">
        <OptionChoice option="comboMake"/>
        <OptionItem option="priceRange"/>
        <RoleItem role="VAR"/>
        <OptionItem option="message"/>
        <OptionChoice option="comboVar"/>
    </Container>

</UI>
```

1 The `DataSource` element specifies that Sashelp.Cars is the default data source for this task.

2 The `Filter` element (filterMake) specifies that the values for this filter come from the Make column in the Sashelp.Cars data source. Because the `Column` element is used, the column is static.

3 The second `Filter` element specifies the `role` attribute. In this case, the column is dynamic. The filter values come from the column that the user selects at run time.

4 The first `Expression` element specifies that the value of the cylinder column must be greater than 6. Only column values that meet this criterion are considered for the next expression.

5 The second `Expression` element specifies that the value of the Make column must equal the value selected in the first `Filter` element (filterMake).

6 The third `Expression` element specifies that the value of MSRP must be in the range specified in the numericrange control called priceRange.

Working with the Options Element

About the Option Element

The `Options` element identifies the options that appear in the task's user interface. The `Option` tag, which is the only child of the `Options` element, describes the assigned option.

Here are the attributes of the `Option` element.

Table 3.8 *Attributes for the Option Element*

Attribute	Description
`active`	Specifies whether the option's Velocity value is accessible. The default value is `false`. When this attribute is set to `true`, the option value is available even when hidden, disabled, or when the value does not appear in the user interface.
`defaultValue`	Specifies the initial value for the option.
`helpMessageRef`	Specifies whether to display help content. This content could be a string or markdown. When this attribute is defined, a Help icon appears to the right of the control's label.
`hide`	Specifies whether to display the control in the user interface. By default, this attribute is `false`. If this attribute is set to `true`, the control is not displayed.

Attribute	Description
`indent`	Specifies the indention 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)
`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 24.
`name`	Specifies the name assigned to this option.
`readOnly`	Specifies whether the current value can be edited. The default value is `false`. If this attribute is `true`, the current value is displayed but cannot be edited.
`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.

Attribute	Description
	For more information, see “Specifying a Return Value Using the returnValue Attribute” on page 70.
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.

Here is an example of an Options element:

```
<Options>
   <Option name="PRINTOPTIONS" inputType="string">List Data Options</
Option>
   <Option name="BASICOPTIONS" inputType="string">Basic Options</
Option>

   <Option name="OBS" defaultValue="1" inputType="checkbox">
      Print row number</Option>
   <Option name="OBSHEADING" defaultValue="Row number"
      inputType="inputtext">Column heading:</Option>

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

   <Option name="PRINTNUMROWS" defaultValue="0" inputType="checkbox">
      Print number of rows</Option>
</Options>
```

Supported Input Types

checkbox

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

The `checkbox` input type does not have additional attributes.

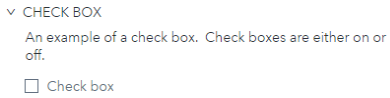
The default value for a `checkbox` is either `0` (unchecked) and `1` (checked).

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>

...
<UI>
   <Group option="GROUPCHECK">
```

```
        <OptionItem option="labelCHECK"/>
         <OptionItem option="chkEXAMPLE"/>
    </Group>
</UI>
```

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



color

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

The color input type has one additional attribute:

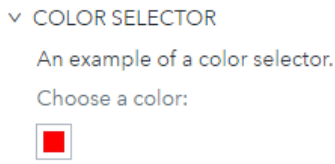
Table 3.9 *Attribute for color Input Type*

Attribute	Description
`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.

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>

...
<UI>
   <Group option="GROUPCOLOR">
       <OptionItem option="labelCOLOR"/>
        <OptionItem option="colorEXAMPLE"/>
    </Group>

</UI>
```

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



combobox

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

This input type has these additional attributes:

Table 3.10 *Attributes for combobox Input Type*

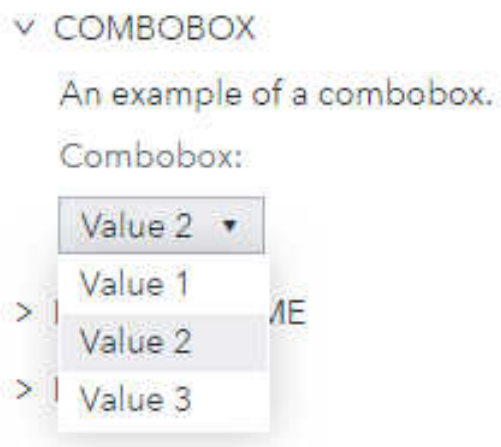
Attribute	Description
`allowAllValues`	Specifies whether to add an item to the list so that the user can select all possible values. By default, this attribute is `false`, and no item is added to the list. You might use this attribute when the list is generated by a filter.
`allowMissingValues`	Specifies whether to add an item to the list so that the user can select missing values. By default, this attribute is `false`, and no item is added to the list. You might use this attribute when the list is generated by a filter.
`editable`	Specifies whether the user can enter a value in the combobox control. By default, users cannot enter a new value in the combobox control.
`filter`	Specifies the filter to use for pulling data for this option. **Note:** If this attribute is specified, the `sourceLink` attribute and any children in the `OptionChoice` element are ignored.
`required`	Specifies whether a value is required. Valid values are `true` and `false`. The default value is `false`. **Note:** If the `required` attribute is set to `true` and no default value is specified, the combobox control displays the text specified in the `selectMessage` attribute.
`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`.

Attribute	Description
`sourceLink`	Specifies that the data for the combobox should be pulled from another source option. For more information, see "About Data Linking" on page 71.
`width`	Specifies the width of the control. This value can be in percent (%), em, or px. By default, SAS Studio sizes the control based on the available width and content.

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

```
<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>

...
<UI>
   <Group option="GROUPCOMBO">
      <OptionItem option="labelCOMBO"/>
      <OptionChoice option="comboEXAMPLE">
         <OptionItem option="value1"/>
            <OptionItem option="value2"/>
            <OptionItem option="value3"/>
        </OptionChoice>
    </Group>
</UI>
```

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



datepicker

This datepicker control returns all date values in the ISO8601 format (yyyy-MM-dd).

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

This input type has these additional attributes:

Table 3.11 *Attributes of datepicker Input Type*

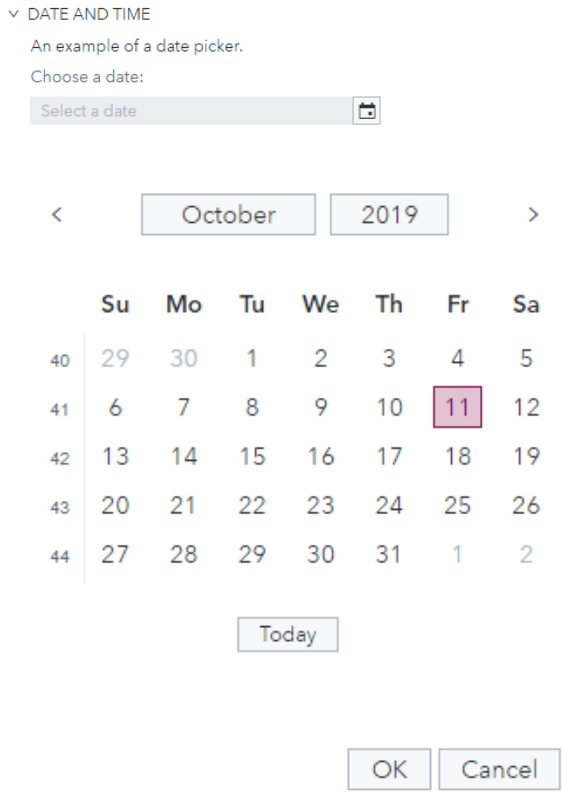
Attribute	Description
`displayFormat`	Specifies the visual formatting of the date. The valid values are `short` (default), `medium`, and `long`.
`required`	Specifies whether a date is required. By default, no date is required.
`minDate`	Specifies the minimum threshold for date values. This attribute is not set by default.
`maxDate`	Specifies the maximum threshold for date values. This attribute is not set by default.
`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.

If you specify the `defaultValue`, `minDate`, and `maxDate` 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:**.

```
<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>

...
<UI>
   <Group option="GROUPDATE">
      <OptionItem option="labelDATE"/>
      <OptionItem option="dateEXAMPLE"/>
   </Group>
</UI>
```

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



daterange

The `daterange` control enables the user to choose a date and time period.

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

This input type has these additional attributes:

Table 3.12 *Attributes for daterange Input Types*

Attribute	Description
`defaultFromValue`	Specifies the default value for the **From** input field. The default value is null.
`defaultToValue`	Specifies the default value for the **To** input field. The default value is null.
`displayFormat`	Specifies the visual formatting of the date. The valid values are `short` (default), `medium`, and `long`.
`dateType`	Specifies the type of control to display in the user interface. Here are the valid values: `date` (the default), `week`, `month`, `quarter`, `year`, `time`, and `datetime`.

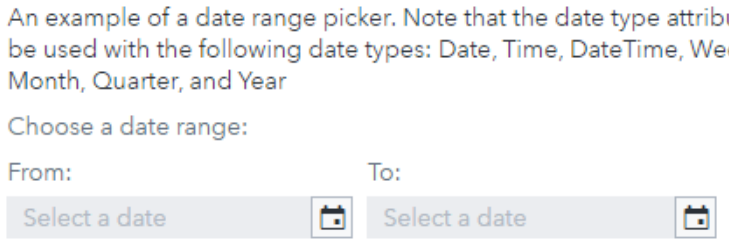
Attribute	Description
`fromLabel`	Overrides the label for the **From** input field.
`toLabel`	Overrides the label for the **To** input field.
`minValue`	Specifies the minimum threshold for values. Any values less than this threshold are invalid. This attribute is not set by default.
`maxValue`	Specifies the maximum threshold for values. Any values greater than this threshold are invalid. This attribute is not set by default.
`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.
`required`	Specifies whether a date is required. By default, no date is required.

Note: When using the `daterange` element, consider these items:

- The standard attribute, `defaultValue`, is not a valid attribute for the `daterange` element. Instead, use the `defaultFromValue` and `defaultToValue` attributes.
- All date and time attributes should be in an ISO format.
- The `displayFormat` attribute is not valid for week and year.
- The `minValue` and `maxValue` attributes are not honored when the date range is time. If the date range is datetime, the date portion of the `minValue` and `maxValue` attributes is honored, but not the time portion.

```
<Option name="labelDATERANGE" inputType="daterange">
       Specify the date range:
</Option>

<Option name="labelDATETIMERANGE" inputType="daterange"
displayFormat="short"
   dateType="DateTime"  maxValue="2019-12-31T19:15:22"
   defaultFromValue="2018-12-31T19:15:22">
   Specify the time range:
</Option>

<Option name="labelYEARRANGE" inputType="daterange" dateType="year"
   maxValue="2050" defaultFromValue="2018" defaultToValue="2020">
       Specify the year range:
</Option>

...

<UI>
   <OptionItem option="labelDATERANGE"/>
```

```
    <OptionItem option="labelDATETIMERANGE"/>
    <OptionItem option="labelYEARRANGE"/>
</UI>
```



datetimepicker

The `datetimepicker` element enables you to choose a date and time.

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

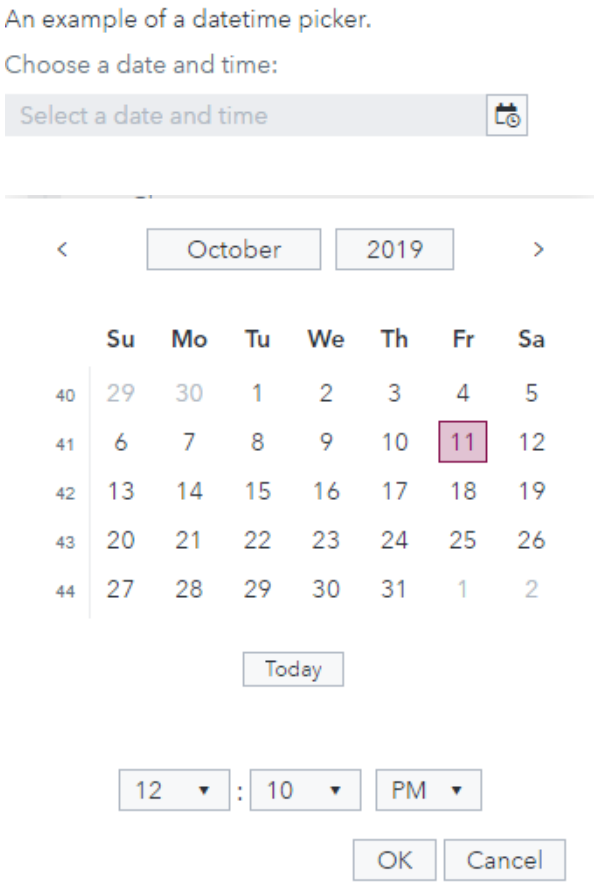
This input type has these additional attributes:

Table 3.13 *Attributes for datetimepicker Input Type*

Attribute	Description
`displayFormat`	Specifies the visual formatting of the date. The valid values are `short` (default), `medium`, and `long`.
`required`	Specifies whether a date is required. By default, no date is required.
`minDate`	Specifies the start date. All previous dates are disabled. The default value is null.
`maxDate`	Specifies the end date. All future dates are disabled. The default value is null.
`use24HourTime`	Specifies whether to display the time using the 24-hour format instead of AM and PM. By default, this attribute is `false`, and the AM and PM format is used.
`showSeconds`	Specifies whether to show seconds in the time. By default, this attribute is `false`, and the seconds are not displayed.
`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.

If you specify the `defaultValue`, `minDate`, and `maxDate` attribute for this input type, the value must be in ISO8601 format: yyyy-MM-ddTHH:mm:ss and yyyy-MM-dd.

```
<Option name="labelDATETIME" inputType="datetimepicker" required="true"
   defaultValue="2018-09-05T19:15:22" displayFormat="long"
   minDate="2018-09-01" maxDate="2018-09-30" use24HourTime="true"
   helpMessageRef="helpMessage">
   Select a Date and Time:
</Option>

...

<UI>
   <OptionItem option="labelDATETIME"/>
</UI>
```



distinct

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

This input type has these additional attributes:

Table 3.14 *Attributes for distinct Input Type*

Attribute	Description
`required`	Specifies whether a value is required. 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.
`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`.
`source`	Specifies the role to use to get the distinct values. The `maxVars` control for the role must be set to 1. In other words, users can assign only one variable to this role.
`max`	Specifies the maximum number of distinct values to obtain and display in the UI. By default, the maximum value is `100`. Larger maximum values might cause a long delay in populating the UI control. **Note:** Missing values are ignored, so missing values do not appear in the list of distinct values.
`width`	Specifies the width of the control. This value can be in percent (%), em, or px. By default, SAS Studio sizes the control based on the available width and content.

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 12.

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.

```
<DataSources>
   <DataSource name="DATASOURCE">
      <Roles>
         <Role type="A" maxVars="1" order="true" minVars="1"
            name="VAR">Response variable</Role>
```

```
      </Roles>
   </DataSource>
</DataSources>
<Options>
   <Option name="values" inputType="distinct" source="VAR"
      max="15">Age of interest:</Option>
</Options>

...
<UI>
   <Group option="GROUPDISTINCT">
       <OptionItem option="labelDISTINCT"/>
        <OptionChoice option="distinctEXAMPLE"/>
    </Group>
</UI>
```

Here is an example of the distinct control in the sample task:

dualselector

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

This input type has these additional attributes:

Table 3.15 *Attributes of the dualselector Input Type*

Attribute	Description
`allowAllValues`	Specifies whether to add an item to the list so that the user can select all possible values. By default, this attribute is `false`, and no item is added to the list. You might use this attribute when the list is generated by a filter.
`allowMissingValues`	Specifies whether to add an item to the list so that the user can select missing values. By default, this attribute is `false`, and no item is added to the list. You might use this attribute when the list is generated by a filter.
`dataType`	Specifies the type of data allowed in the dualselector control. If the `editable` attribute is set to `true`, the edit field uses the value for `dataType`. Here are the valid values:

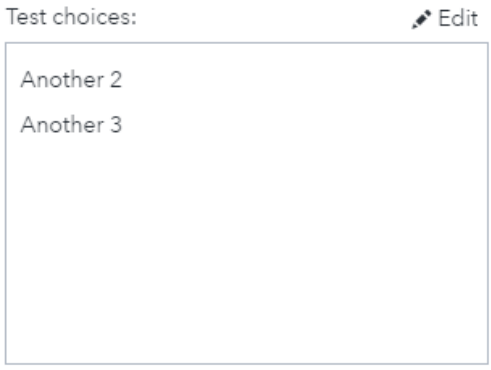
Attribute	Description
	■ `auto` (default). If the `dataType` attribute is used with a filter, `auto` chooses the data type based on the column type. If a filter is not applied, `auto` is the same as text. ■ `date` ■ `number` ■ `text` If you are using static values, the `returnValue` should be specified when the dataType is either date or number. Date value should be in the ISO format (yyyy-mm-dd). If the return value does not equal the data type or is forced into that data type, the value is not added to the list in the dualselector control.
`editable`	Specifies whether the user can enter a value. This attribute is used with the `dataType` attribute.
`filter`	Specifies the filter to use for pulling data for this option. **Note:** If this attribute is specified, the `sourceLink` attribute and any children in the `OptionChoice` element are ignored.
`height`	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.
`maxItems`	Specifies the maximum number of values that can be selected from the list of available values.
`minItems`	Specifies the minimum number of values that must be selected from the list of available values.
`reorderable`	Specifies whether the list of values can be reordered. By default, this attribute is set to `true`, and the values can be reordered.
`required`	Specifies whether any input text is required. Valid values are `true` and `false`. The default value is `false`.
`width`	Specifies the width of the control. This value can be in percent (%), em, or px. By default, SAS Studio sizes the control based on the available width and content.

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.

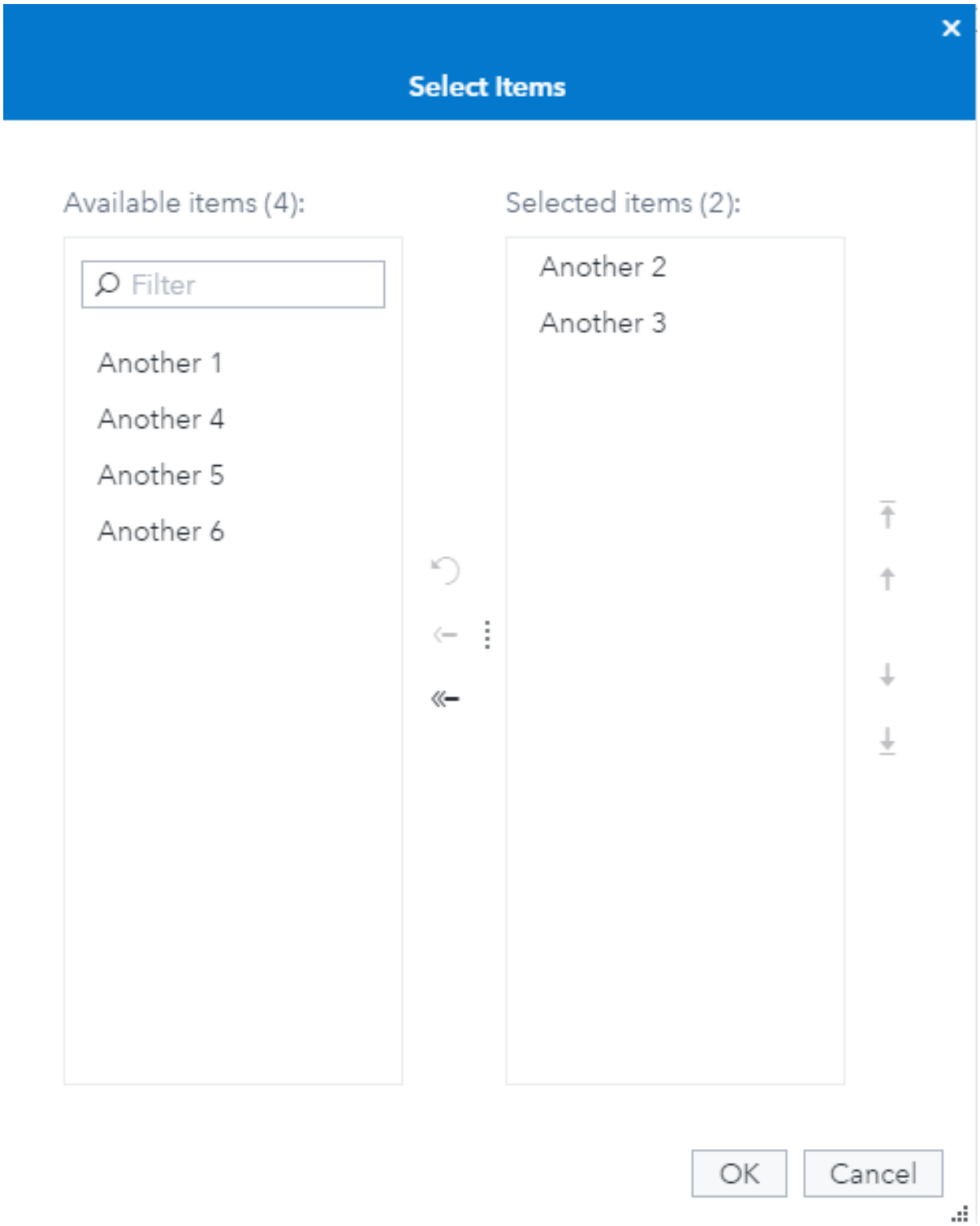
This example shows how the dualselector control works.

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

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 items** 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

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

This input type has these additional attributes:

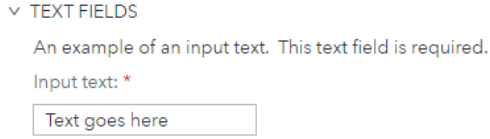
Table 3.16 *Attributes for the inputtext Input Type*

Attribute	Description
`hintMessage`	Specifies the placeholder text to display when the text box is empty.
`missingMessage`	Specifies the tooltip text that appears when the text box that the user has selected is empty but input text is required (specified by `required="true"`).
`promptMessage`	Specifies the tooltip text that appears when the text box that the user has selected is empty but input text is not required. The `promptMessage` attribute is used for fields that are not required. Use `missingMessage` for field that are required.
`required`	Specifies whether any input text is required. Valid values are `true` and `false`. The default is `false`.
`width`	Specifies the width of the control. This value can be in percent (%), em, or px. By default, SAS Studio sizes the control based on the available width and content.

The 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"
   missingMessage="Text goes here.">Input text:</Option>
...
<UI>
   <OptionItem option="textEXAMPLE"/>
</UI>
```

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



markdown

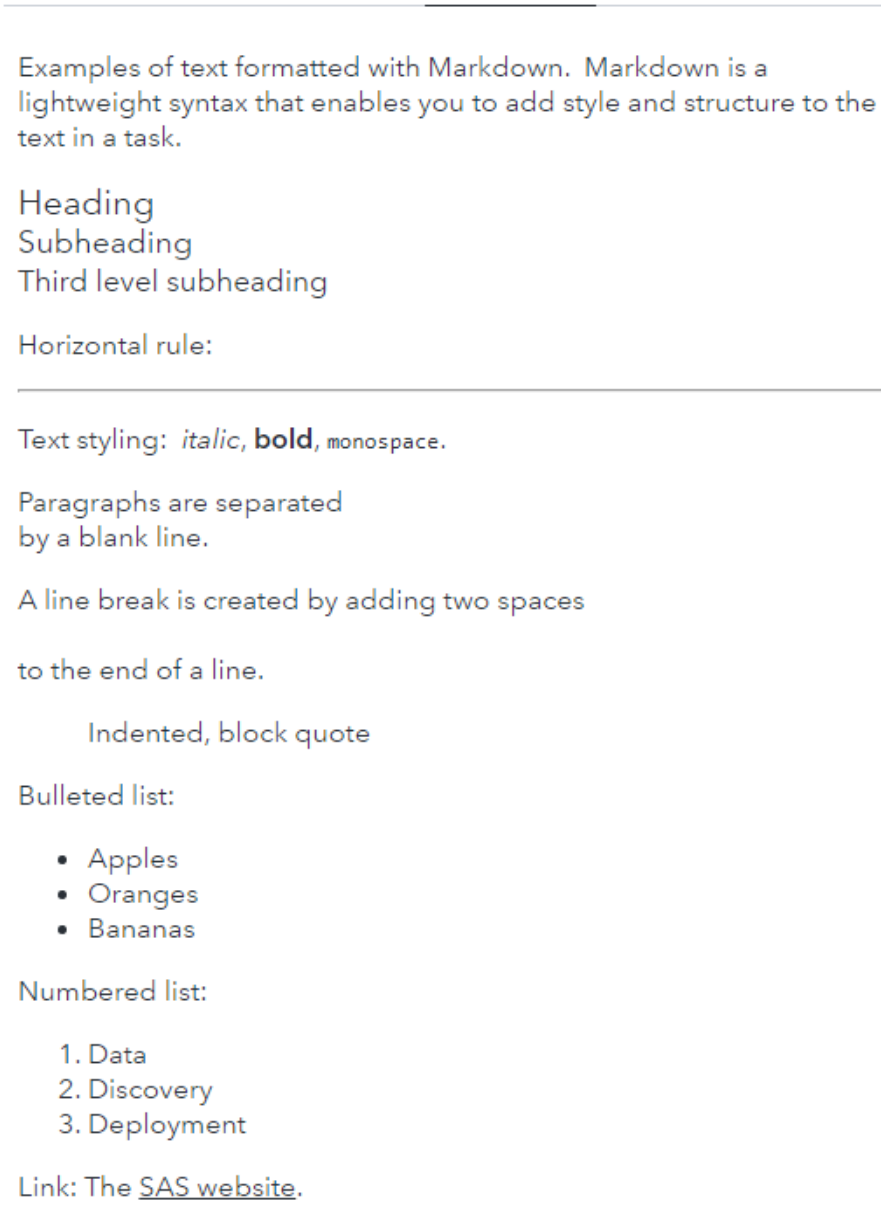
This input type is used to display formatted text to the user. Standard markdown is accepted. You should use markdown's syntax of punctuation of characters rather than an HTML tag because the HTML tags are removed for security reasons.

Use the `markdown` attribute to display formatted text to the user. You can also refer to the `markdown` attribute from an option's `helpMessageRef` attribute.

```
<Option name="markdownTextEXAMPLE" inputType="markdown">
Heading
=======

## Sub-heading

Horizontal rule:

---

Text styling:  _italic_,
**bold**, `monospace`.

Paragraphs are separated
by a blank line.

A line break is created by adding
two spaces to the end of a line.

 &gt; Indented, block quote

Bulleted list:

  * Apples
  * Oranges
  * Pears

Numbered list:

  1. Data
  2. Discovery
  3. Deployment

Link: The [SAS website](http://www.sas.com/).

Image: ![SAS](https://brand.sas.com/en/home/our-identity/visual-
elements/
   logo/_jcr_content/par/styledcontainer/par/tabwrapper/
tabwrapperpar/tab/
   tabpar/image.img.png/1500576966295.png)
</Option>

<UI>
   <Container option="MARKDOWNTAB">
      <OptionItem option="markdownTextEXAMPLE"/>
```

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

Here is a subset of the contents of the Markdown tab in the Sample Task.



Examples of text formatted with Markdown. Markdown is a lightweight syntax that enables you to add style and structure to the text in a task.

Heading
Subheading
Third level subheading

Horizontal rule:

Text styling: *italic*, **bold**, `monospace`.

Paragraphs are separated
by a blank line.

A line break is created by adding two spaces

to the end of a line.

> Indented, block quote

Bulleted list:

- Apples
- Oranges
- Bananas

Numbered list:

1. Data
2. Discovery
3. Deployment

Link: The SAS website.

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, ...*n*th-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.

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

Here are the additional attributes for the `mixedeffects` input type:

Table 3.17 *Attributes for the mixedeffects Input Type*

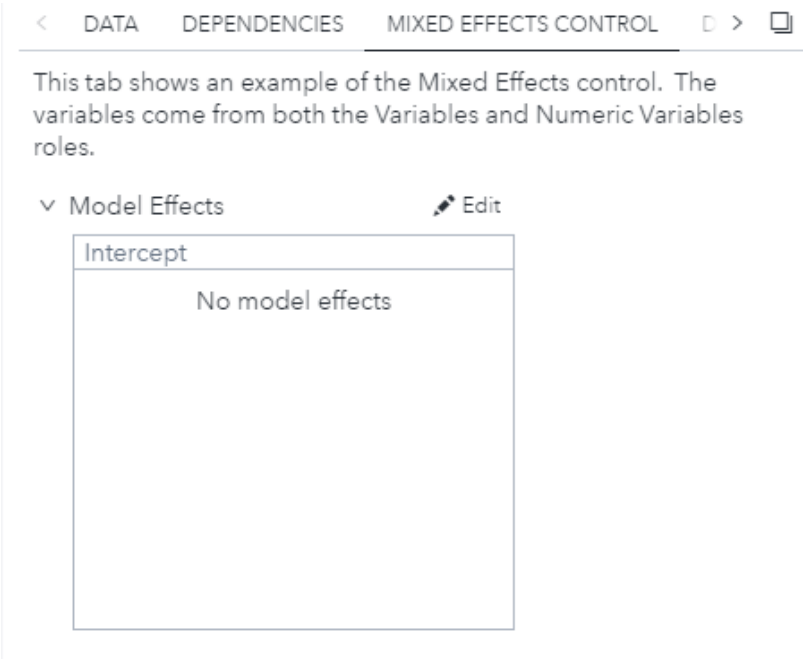
Attribute	Description
`effects`	Specifies the list of effects that you want available from the task interface: ■ `fixed` – only fixed effects. This is the default value. ■ `fixedrandom` – fixed effects and random effects. ■ `fixedrandomrepeated` – fixed effects, random effects, and repeated effects. ■ `fixedrepeated` – fixed and repeated effects. ■ `meanszero` – means and zero-inflated effects.
`roleContinuous`	Specifies the role that contains the continuous variables.
`roleClassification`	Specifies the role that contains the classification variables.
`excludeTools`	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.

Attribute	Description
fixedInterceptVisible	Specifies whether the intercept option is available for fixed effects or mean effects. Valid values are true and false. The default value is true.
fixedInterceptDefaultValue	Specifies the default value for the intercept option if fixedInterceptVisible = true. Valid values are 0 and 1. The default value is 1.
randomInterceptVisible	Specifies whether the intercept option is available for random effects. Valid values are true and false. The default value is true.
randomInterceptDefaultValue	Specifies the default value for the intercept option if randomInterceptVisible = true. Valid values are 0 and 1. The default value is 1.
width	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.

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

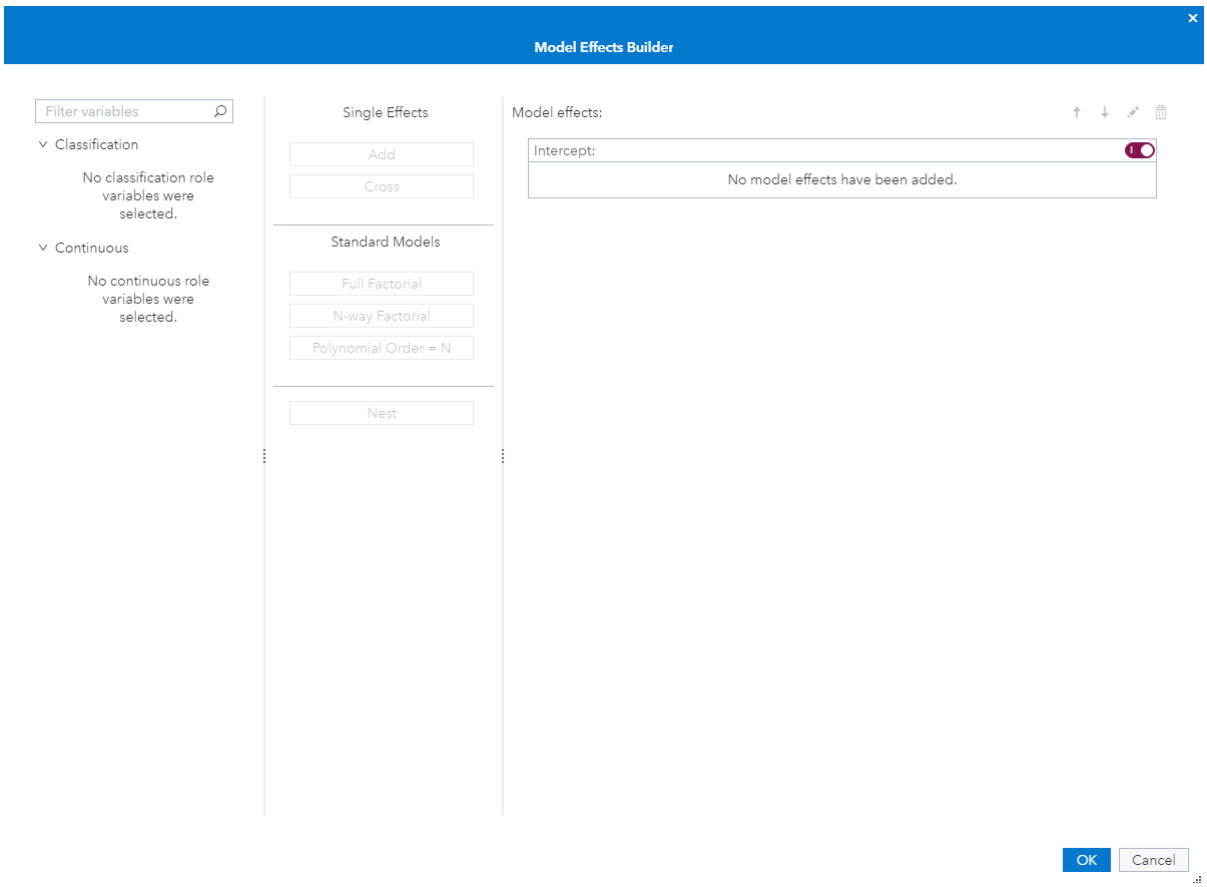
```
<Options>
   <Option name="MECTAB" inputType="string">MIXED EFFECTS CONTROL</Option>
    <Option name="MECTEXT" inputType="string">This tab shows an example
     of the Mixed Effects control.The variables come from both the
     Variables and Numeric Variables roles.</Option>
    <Option name="mixedEffects"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

...
</Options>
   <UI>
      <Container option="MECTAB">
            <OptionItem option="MECTEXT"/>
            <OptionItem option="mixedEffects"/>
        </Container>
   </UI>
```

If you run the Advanced Task, here is the resulting **Mixed Effects Control** tab:

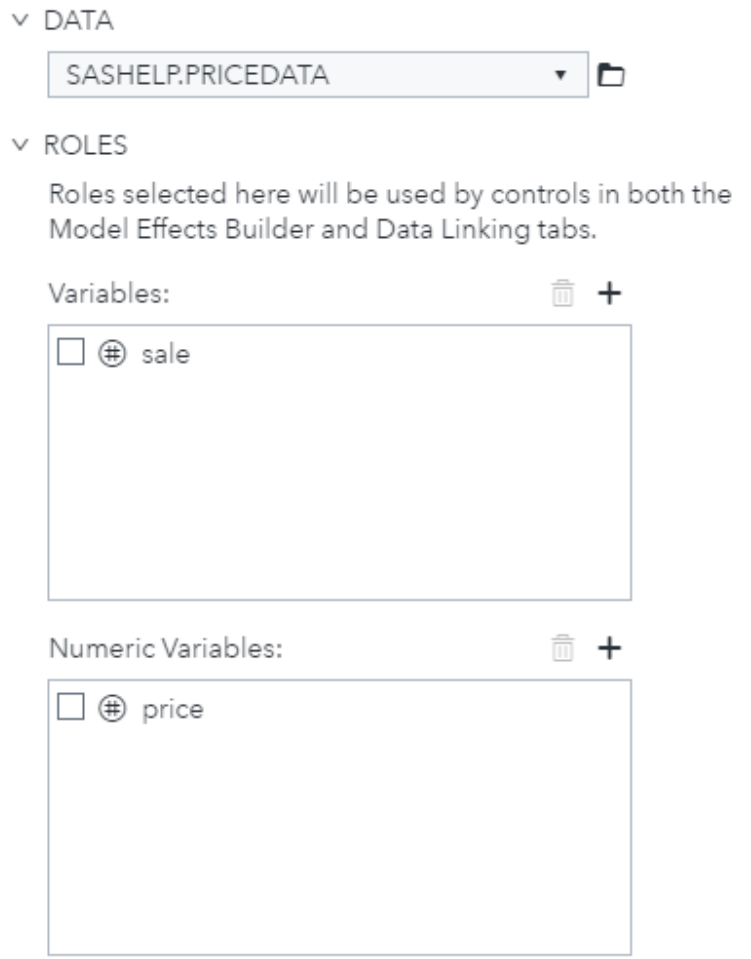


If you click **Edit**, the Model Effects Builder appears.

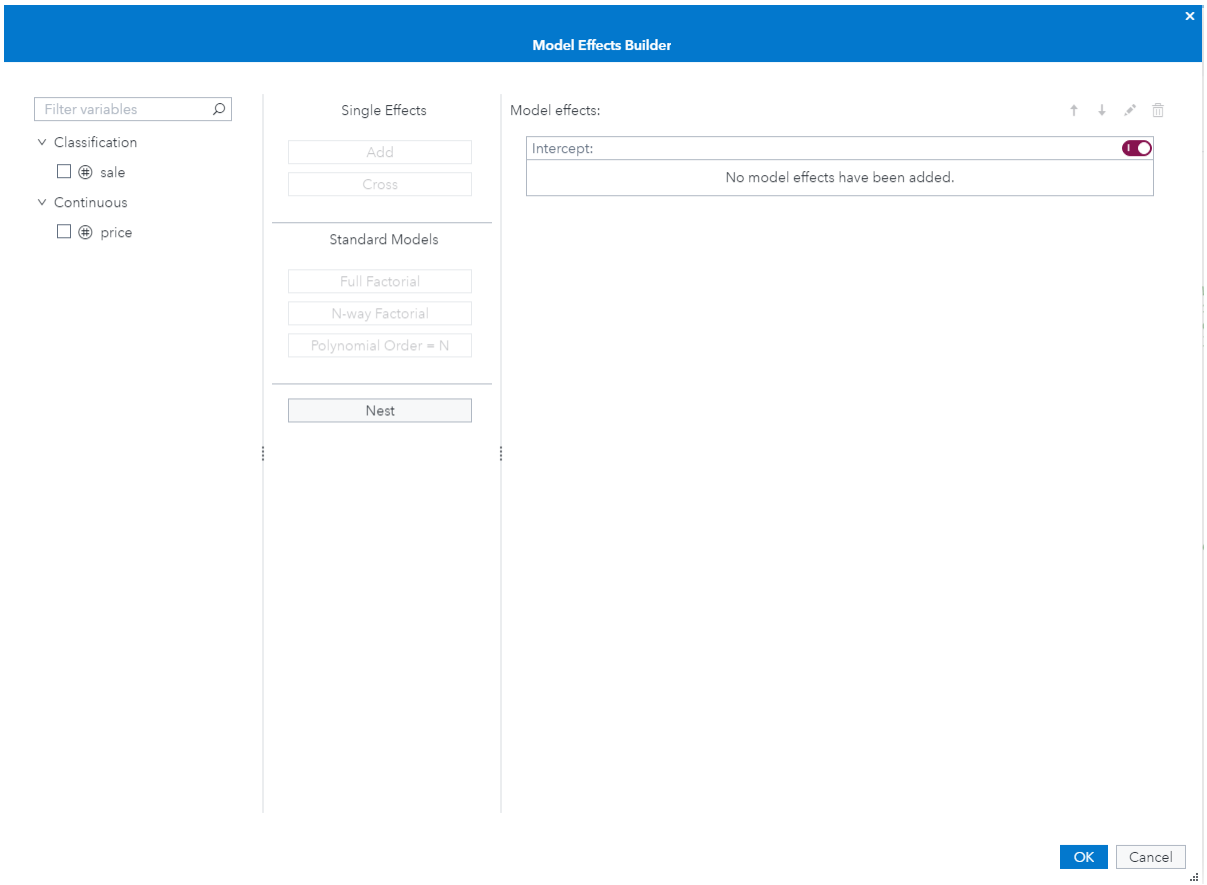
The component opens, but there are no variables available. You must assign a variable to the continuous variable or classification variable role. You can assign variables to both roles.



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.



monthpicker

The `monthpicker` attribute creates a control that enables the user to choose a month and year.

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

This input type has these additional attributes:

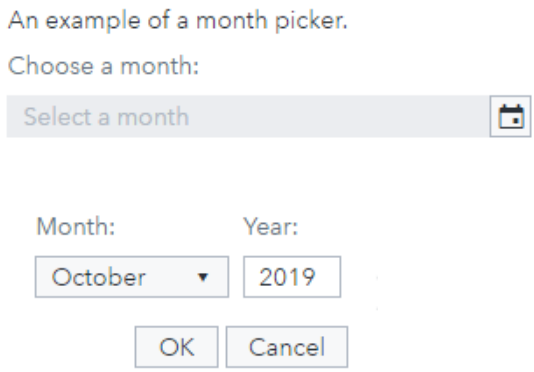
Table 3.18 *Attributes for monthpicker Input Type*

Attribute	Description
`displayFormat`	Specifies the visual formatting of the date. The valid values are `short` (default), `medium`, and `long`. If you are using static values, the `returnValue` should be specified when the dataType is either date or number. Date value should be in the ISO format (yyyy-mm-dd). If the return value does not equal the data type or be forced into that data type, the value is not added to the list in the dualselector control.

Attribute	Description
required	Specifies whether a date is required. By default, no date is required.
minValue	Specifies the minimum threshold for the month values. This attribute is not set by default.
maxValue	Specifies the maximum threshold for month values. This attribute is not set by default.
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.

If you specify the `defaultValue`, `minDate`, and `maxDate` attribute for this input type, the value must be in ISO8601 format: yyyy-MM.

```
<Option inputType="monthpicker" name="monthEXAMPLE"
defaultValue="2020-07"
   minValue="2019-15" maxValue="2020-25" displayFormat="long">
   Select a month:
</Option>
...
<UI>
   <OptionItem option="monthEXAMPLE"/>
</UI>
```



multientry

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

This input type has these additional attributes:

Table 3.19 *Attributes of the multientry Input Type*

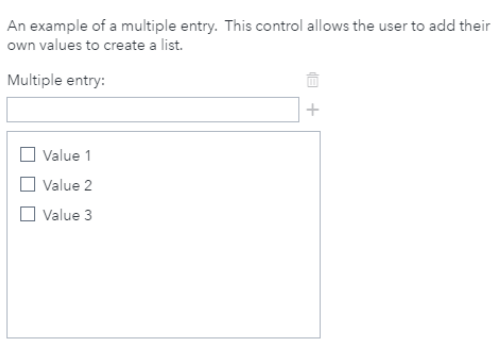
Attribute	Description
`dataType`	Specifies the data type that is allowed for user-specified values. The edit field matches the data type. The valid values are `date`, `number`, and `text` (default).
`required`	Specifies whether a value is required. Valid values are `true` and `false`. The default value is `false`.
`width`	Specifies the width of the control. This value can be in percent (%), em, or px. By default, SAS Studio sizes the control based on the available width and content.
`reorderable`	Specifies whether the user can reorder the values in the list. Valid values are `true` and `false`. The default value is `false`.

You can assign default values to the multientry control by using the `OptionsChoice` element.

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

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

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

...
```

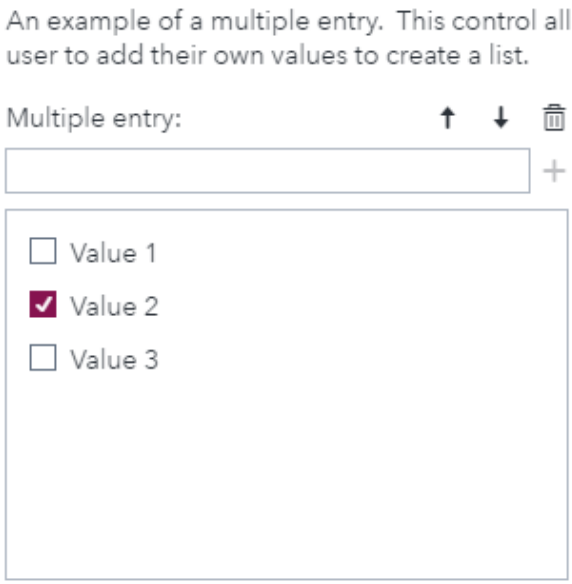
In this example, the **Multiple entry** option has three values: **Value 1**, **Value 2**, and **Value 3**. To add additional values to the list, enter the name of the new value in the text box and click +.



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

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

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

...
```

Now, the multientry control includes up and down arrows.



numbertext

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

This input type has these additional attributes:

Table 3.20 Attributes for numbertext Input Type

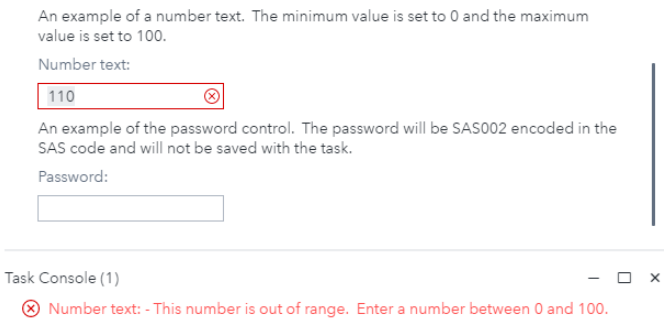
Attribute	Description
`decimalPlaces`	Specifies the number of decimal places to display. Valid values include a single value or a range. To create a field that allows 0 to 3 decimal places, specify `decimalPlaces="0,3"`. The maximum number of decimal places is 15.
`formatValue`	Specifies whether the number should be formatted to include locale-specific delimiters in the user interface. The default value is `true`. **Note:** Setting this attribute does not change the numeric value in the Velocity variable.
`hintMessage`	Specifies the placeholder text when the control is empty.
`invalidMessage`	Specifies the tooltip text that appears when the content is invalid.
`maxValue`	Specifies the maximum value that is allowed. If the user tries to exceed this value, a message appears. The default value is `9000000000000`.
`maxInclusive`	Specifies whether to include the maximum value in the range of values. By default, the `maxInclusive` attribute is set to `true`.
`minValue`	Specifies the minimum value that is allowed. If the user specifies a value that is below the minimum value, a message appears.
`minInclusive`	Specifies whether to include the minimum value in the range of values. By default, the `minInclusive` attribute is set to `true`.
`missingMessage`	Specifies the tooltip text that appears when the text box that the user has selected is empty but input text is required (specified by `required="true"`).
`promptMessage`	Specifies the tooltip text that appears when the text box that the user has selected is empty but input text is not required. The `promptMessage` attribute is used for fields that are not required. Use `missingMessage` for field that are required.
`rangeMessage`	Specifies the tooltip text that appears when the value in the text box is outside the specified range.
`required`	Specifies whether a value is required. Valid values are `true` and `false`. The default value is `false`.

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

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."
   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>

...
<UI>
   <OptionItem option="labelNUMBERTEXT"/>
   <OptionItem option="numberTextEXAMPLE"/>
</UI>
```

Here is an example of the numbertext control in the user interface. In this example, 110 is higher than the maximum value allowed for this field. A tooltip with an error message appears in the user interface and an error message appears in the Task Console.



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.

numericrange

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

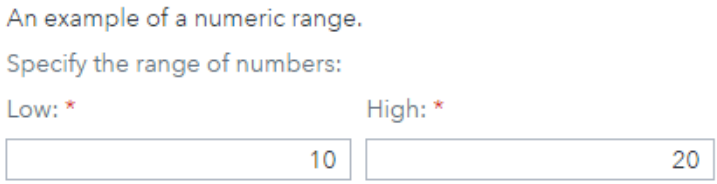
This input type has these additional attributes:

Table 3.21 *Attributes for numericrange Input Type*

Attribute	Description
`decimalPlaces`	Specifies the range of acceptable decimal places (minimum, maximum). For example, `decimalPlaces="0,3"`.
`defaultFromValue`	Specifies the default value for the **From** input field. The default value is null.
`defaultToValue`	Specifies the default value for the **To** input field. The default value is null.
`fromLabel`	Specifies the label for the **From** input field.
`toLabel`	Specifies the label for the **To** input field.
`minInclusive`	Specifies whether the minimum value is in the range of values in the **From** input field. By default, the `minInclusive` attribute is set to `true`, and these values can be equal.
`maxInclusive`	Specifies whether the maximum value is in the range of values in the **To** input field. By default, the `maxInclusive` attribute is set to `true`, and these values can be equal.
`maxValue`	Specifies the highest value that the user can specify. Any values greater than the maximum value are invalid. By default, the `maxValue` attribute is not set.
`minValue`	Specifies the lowest value that the user can specify. Any values less than the minimum value are invalid. By default, the `minValue` attribute is not set.
`required`	Specifies whether a value is required. By default, this attribute is `false`.
`width`	Specifies the width of the control. This value can be in percent (%), em, or px. By default, SAS Studio sizes the control based on the available width and content.

Here is an example:

```
<Option name="labelNUMERICRANGE" inputType="numericrange"
minValue="-10"
   maxValue="100000" minInclusive="true" maxInclusive="true"
   decimalPlaces="0,3"
   defaultFromValue="10" defaultToValue="20"
helpMessageRef="helpMessage"
   required="false" fromLabel="Low:" toLabel="High:">
   Specify the range of numbers
</Option>
```



numstepper

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

This input type has these additional attributes:

Table 3.22 *Attributes of numstepper Input Type*

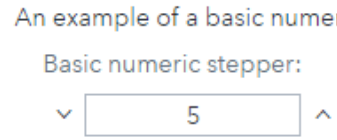
Attribute	Description
`decimalPlaces`	Specifies the number of decimal places to display. Valid values include a single value or a range. To create a field that allows 0 to 3 decimal places, specify `decimalPlaces="0,3"`.
`formatValue`	Specifies whether the formatted number should include locale-specific delimiters. The default value is `true`. **Note:** The numeric value in the Velocity variable is not formatted.
`increment`	Specifies the number of values that the option increases or decreases when a user clicks the up or down arrow. The default value is `1`.
`invalidMessage`	Specifies the tooltip text that appears when the content in the field is invalid.
`maxValue`	Specifies the maximum value that is allowed. If the user tries to exceed this value, a message appears. The default value is `9000000000000`.
`maxInclusive`	Specifies whether to include the maximum value in the range of values. By default, this attribute is `true`.
`minValue`	Specifies the minimum value that is allowed. If the user specifies a value that is below the minimum value, a message appears.
`minInclusive`	Specifies whether to include the minimum value in the range of values. By default, this attribute is `true`.

Attribute	Description
missingMessage	Specifies the tooltip text that appears when the text box that the user has selected is empty but input text is required (specified by required="true").
promptMessage	Specifies the tooltip text that appears when the text box that the user has selected is empty but input text is not required. The promptMessage attribute is used for fields that are not required. Use missingMessage for field that are required.
rangeMessage	Specifies the tooltip text when the value in the text box is outside the specified range.
required	Specifies whether a value is required. Valid values are true and false. The default value is false.
width	Specifies the width of the control. This value can be in percent (%), em, or px. By default, SAS Studio sizes the control based on the available width and content.

The first example in the Sample Task 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>

...

<UI>
   <OptionItem option="labelNumStepperEXAMPLE1"/>
    <OptionItem option="basicStepperEXAMPLE"/>
</UI>
```

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

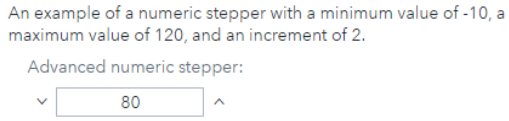


The second example in the Sample Task creates an option with a specified minimum value, maximum value, and increment.

```
<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>

...

<UI>
        <OptionItem option="labelNumStepperEXAMPLE2"/>
        <OptionItem option="advancedStepperEXAMPLE"/>
</UI>
```

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



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. The `outputdata` element enforces 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*.

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

This input type has these additional attributes:

Table 3.23 *Attributes of outputdata Input Type*

Attribute	Description
`required`	Specifies whether a name is required. The default value for this attribute is `false`, which means that no name is required.
`width`	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.
`libraryEngineExclude`	Specifies the engine types that are not valid for the data source. The engine types should be a comma-separated list. Possible values are `V9` and `CAS`.
`libraryEngineInclude`	Specifies the engine types that are valid for the data source. The engine types should be a comma-separated list. Possible values are `V9` and `CAS`.

Attribute	Description
`unique`	Specifies whether the value of the output control is unique. This is a Boolean value with a default value of `false`. If `unique=true`, the task ensures that the value of the outputdata control is unique when compared to all other outputdata control where `unique=true`.

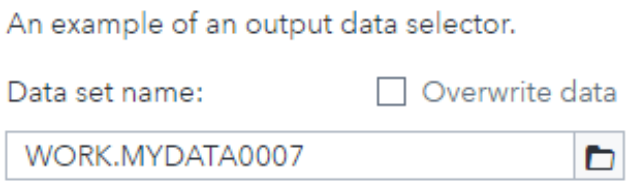
Note: If you do not specify either the `libraryEngineExclude` attribute or the `libraryEngineInclude` attribute, all engine types are available for the data source control. If you need to limit the engine type, use either the `libraryEngineExclude` attribute or the `libraryEngineInclude` attribute. Do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

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 Work.MyData. If no existing data sets use this name, Work.MyData appears as the name in the outputdata control. If a Work.MyData data set already exists, SAS Studio uses the suffix to create a unique name, such as Work.MyData0001. Using this technique prevents SAS Studio from overwriting an existing data set.

```
<Option defaultValue="Work.MyData" indent="1" inputType="outputdata"
   name="outputDSName" required="true">Data set name:</Option>
```

Here is an example of the outputdata control from the Sample Task:



passwordtext

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

This input type has these additional attributes:

Table 3.24 Attributes of passwordtext Input Type

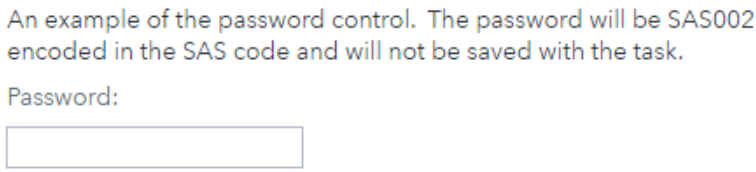
Attribute	Description
`required`	Specifies whether a name is required. The default value for this attribute is `false`, which means that no name is required.
`width`	Specifies the width of the control. The width can be specified in (percent) %, em, or px. By default, SAS Studio determines the size of the control based on the available width and content.

When using this control, remember these restrictions:

- When you save the task file, the password that is currently entered is not saved.
- The `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">Password:</Option>

...

<UI>
   <OptionItem option="pswd"/>
</UI>
```



quarterpicker

The `quarterpicker` attribute enables the user to choose a quarter and year.

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

This input type has these additional attributes:

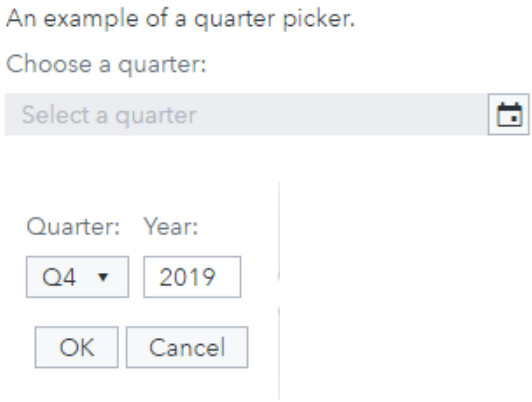
Table 3.25 Attributes of quarterpicker Input Type

Attribute	Description
`displayFormat`	Specifies the visual formatting of the date. The valid values are `short` (default), `medium`, and `long`.

Attribute	Description
required	Specifies whether a date is required. By default, no date is required.
minValue	Specifies the minimum threshold for the quarter values. This attribute is not set by default.
maxValue	Specifies the maximum threshold for quarter values. This attribute is not set by default.
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.

If you specify the `defaultValue`, `minValue`, and `maxValue` attribute for this input type, the value must be in yyyyQq format.

```
<Option name="qtrEXAMPLE" inputType="quarterpicker"
defaultValue="2020Q2"
   minValue="2019Q1" maxValue="2020Q4" displayFormat="long" >
   Select the quarter:
</Option>
...

<UI>
   <OptionItem option="qtrEXAMPLE"/>
</UI>
```



radio

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

This input type has these additional attributes:

Table 3.26 Attributes of radio Input Type

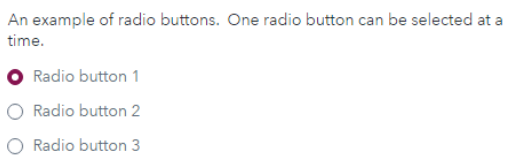
Attribute	Description
`returnValue`	Specifies that the value of the Velocity variable is the return value rather than the name of the radio control.
`variable`	Specifies the Velocity variable that contains the name (or return value) of the currently selected radio button.

One radio button in a group must be selected. If none of the values for the radio button list include the `defaultValue` attribute, the first button in the user interface 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>
   <Option name="radioButton3" variable="radioEXAMPLE"
      inputType="radio">Radio button 3</Option>
</Options>

...
<UI>
   <Group option="GROUPRADIO">
       <OptionItem option="labelRADIO"/>
        <OptionItem option="radioButton1"/>
        <OptionItem option="radioButton2"/>
        <OptionItem option="radioButton3"/>
    </Group>
</UI>
```

Here is how this radio control appears in the user interface:



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.

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

This input type has these additional attributes:

Table 3.27 Attributes of sasserverpath Input Type

Attribute	Description
`pathType`	Specifies the type of selector for this control. Here are the valid values: ■ `file` (default) enables the user to select a file from the SAS server. ■ `folder` enables the user to select a folder. The folder must already exist. The user can create a new folder by using the Folder Selection window.
`defaultExtension`	Specifies the default extension for the file. If no value is specified, the default extension is `sas`. **Note:** Available only if `pathType="file"`.
`defaultName`	Specifies the default filename. If no value is specified, the default filename is `program`. **Note:** Available only if `pathType="file"`.
`required`	Specifies whether a selection is required. The default value is `false`.
`width`	Specifies the width of the control in percent (%), em, or px.

Here is an example of the `sasserverpath` control:

```
<Option name="fileSelector" inputType="sasserverpath"  defaultFileName="myProgramFile"
   pathType="file" defaultExtension="sas">An example of a SAS Server Path control.
      This example allows the user to select a SAS program file.</Option>
```

An example of a SAS Server Path control. This example allows the user to select a SAS program file.

/Public/score.sas

select

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

This input type has these additional attributes:

Table 3.28 Attributes of select Input Type

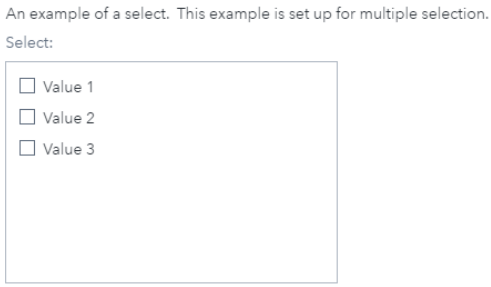
Attribute	Description
`allowAllValues`	Specifies whether to add an item to the list so that the user can select all possible values. By default, this attribute is `false`, and no item is added to the list. You might use this attribute when the list is generated by a filter.
`allowMissingValues`	Specifies whether to add an item to the list so that the user can select missing values. By default, this attribute is `false`, and no item is added to the list. You might use this attribute when the list is generated by a filter.
`filter`	Specifies that the data for this option should come from the specified filter. **Note:** If you specify the `filter` attribute, the `sourceLink` attribute and children of the `OptionChoice` element are ignored.
`height`	Specifies the height of the control in em or px.
`multiple`	Specifies whether users can select one or multiple items from the list. Valid values are `true` and `false`. The default value is `true`.
`required`	Specifies whether the user must select a value from the list. Valid values are `true` and `false`. The default value is `false`.
`sourceLink`	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 71.
`width`	Specifies the width of the control in percent (%), em, or px.

Use the `defaultValue` attribute to specify the items that should be selected at run time. If you need to specify multiple items, use a comma-separated list.

The Sample Task creates an option called **Select**.

```
<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>
```



slider

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

This input type has these additional attributes:

Table 3.29 *Attributes of slider Input Type*

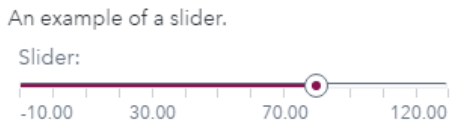
Attribute	Description
`discreteValues`	Specifies the number of discrete values in the slider. For example, if `discreteValues="3"`, the slider has three values: a minimum value, a maximum value, and a value in the middle.
`maxValue`	Specifies the maximum value for this option.
`minValue`	Specifies the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

The first example in the Sample Task creates a slider option with buttons.

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

...
<UI>
   <Group option="GROUPSLIDER">
      <OptionItem option="labelSliderEXAMPLE1"/>
        <OptionItem option="sliderEXAMPLE1"/>
   </Group>
```

```
</UI>
```

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` element, to define string values that are used by the Velocity code, and to define text values to use for the Help Message feature.

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

This input type includes this additional attribute:

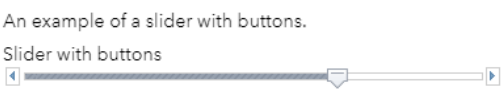
Table 3.30 *Attribute for string Input Type*

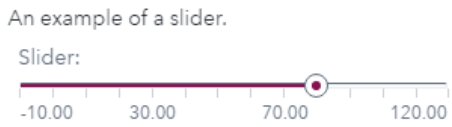
Attribute	Description
`returnValue`	Is the string that is returned in the control's Velocity variable (instead of the control's name). This attribute applies only when the string is used in an `OptionChoice` tag.

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.**) is created by the string input type.

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

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





textbox

The `textbox` input type enables the user to enter multiple lines of text.

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

This input type has these additional attributes:

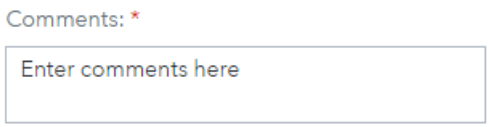
Table 3.31 *Attributes for textbox Input Type*

Attribute	Description
`required`	Specifies whether any input text is required. Valid values are `true` and `false`. The default value is `false`.
`width`	Specifies the width of the control. This value can be in percent (%), em, or px. By default, SAS Studio sizes the control based on the available width and content.
`height`	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.
`splitLines`	Specifies whether to split the text into an array of lines. The split is determined by the newline character. The default value is `false`.

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.



timepicker

The `timepicker` element enables you to choose a time.

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

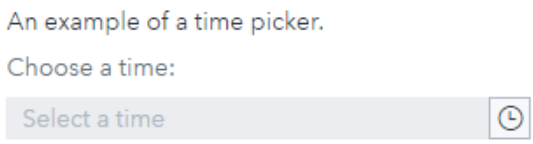
This input type has these additional attributes:

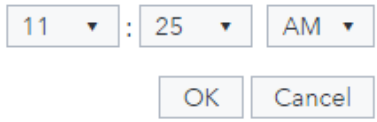
Table 3.32 *Attributes of timepicker Input Type*

Attribute	Description
`required`	Specifies whether a date is required. By default, no date is required.
`use24HourTime`	Specifies whether to display the time using the 24-hour format instead of AM and PM. By default, this attribute is `false`, and the AM and PM format is used.
`showSeconds`	Specifies whether to show seconds in the time. By default, this attribute is `false`, and the seconds are not displayed.
`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.

If you specify the `defaultValue` attribute for this input type, the value must be in ISO8601 format (HH:mm:ss).

```
<Option name="timeEXAMPLE" inputType="timepicker" required="true"
   defaultValue="23:15:22" use24HourTime="true"
   helpMessageRef="timePickerLabel">Select the time:
</Option>

...
<UI>
   <OptionItem option="timeEXAMPLE"/>
</UI>
```





validationtext

This input type enables the user to enter a string value. A regular expression can be used with this option to restrict the entered string to a specific format.

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

This input type has these additional attributes:

Table 3.33 *Attributes of validationtext Input Type*

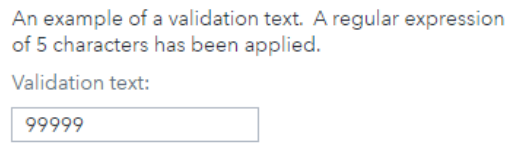
Attribute	Description
`hintMessage`	Specifies the placeholder text to display when the text box is empty.
`invalidMessage`	Specifies the tooltip text to display when the content in the text box is invalid. By default, no message is displayed.
`missingMessage`	Specifies the tooltip text that appears when the text box that the user has selected is empty but input text is required (specified by `required="true"`).
`promptMessage`	Specifies the tooltip text that appears when the text box that the user has selected is empty but input text is not required. The `promptMessage` attribute is used for fields that are not required. Use `missingMessage` for field that are required.
`regExp`	Specifies the regular expression pattern to use for validation. This syntax comes directly from JavaScript Regular Expressions.
`required`	Specifies whether any input text is required. Valid values are `true` and `false`. The default value is `false`.
`width`	Specifies the width of the control. This value can be in percent (%), em, or px. By default, SAS Studio sizes the control based on the available width and content.

The 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>

...
<UI>
   <OptionItem option="labelVALIDATIONTEXT"/>
   <OptionItem option="validationTextEXAMPLE"/>
</UI>
```

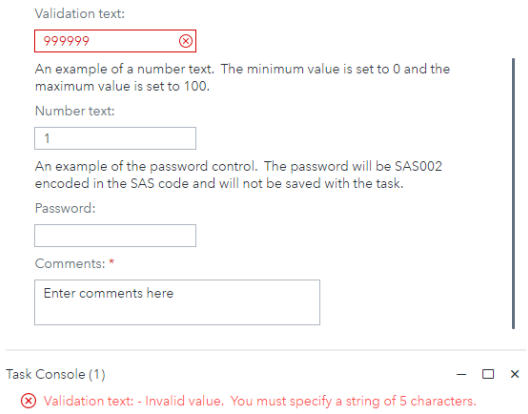
When you run the code, here is the resulting user interface:



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 as a tooltip and in the Task Console.



weekpicker

The `weekpicker` element enables you to choose a week and a year.

Common attributes for the Option element, such as the `defaultValue`, can be used with this input type. These common attributes are listed in Table 3.8 on page 22.

This input type has these additional attributes:

Table 3.34 *Attributes of weekpicker Input Type*

Attribute	Description
`required`	Specifies whether a value is required. By default, no value is required.
`minValue`	Specifies the minimum value. Any values less than the minimum value are disabled. By default, the `minValue` attribute is not set.
`maxValue`	Specifies the maximum value. Any values greater than the maximum value are disabled. By default, the `maxValue` attribute is not set.
`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.

If you specify the `defaultValue`, `minValue`, and `maxValue` attributes for this input type, the value must be in ISO8601 format yyyy-Www.

```
<Option name="oWeek" inputType="weekpicker" defaultValue="2020-W07"
   minValue="2019-W15" maxValue="2020-W25">
   Select a week:
</Option>
...
<UI>
   <OptionItem option="oWeek"/>
</UI>
```

Organizing Options into a Table Component

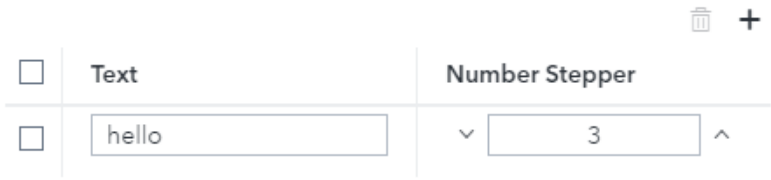
The `OptionTable` element defines a table component that contains one or more custom-defined columns. Each column contains one CTM option. Each individual column can contain a different CTM option. Here are the available CTM options:

- `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 sample task:

An example of an option table control. This control allows for each column of the table to render a different CTM control. Valid control types are: checkbox, comobobx, numbertext, numstepper, inputtext.



Use these attributes to create the table:

Attribute	Description
`name`	Specifies the name assigned to the option.
`label`	Specifies the label for the table in the user interface.
`indent`	Specifies the indention 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)
`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.
`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`.
`maximumRows`	Specifies the maximum number of rows in the option table. The default value is `0`.
`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`.
`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.
`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.

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:

Attribute	Description
`name`	Specifies the name of the column. This attribute is required.
`label`	Specifies the label of the column.
`defaultValues`	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.
`width`	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.

Here is an example that uses the `OptionTable` element:

```
<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" required="true"/>
      </Column>

      <Column name="colComboBox" label="ComboBox">
         <Option inputType="combobox" defaultValue="average"
required="true">
            <Option name="none" inputType="string">None</Option>
            <Option name="average" inputType="string">Average of
values</Option>
            <Option name="total" inputType="string">Sum of values</
Option>
         </Option>
      </Column>
   </Columns>
</OptionTable>
```

Specifying a Return Value Using the returnValue Attribute

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.

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.

For options that support the `dataType` attribute (such as `dualSelector` and `mutlientry`), the `returnValue` attribute must be specified when the data type is `number` or `date`.

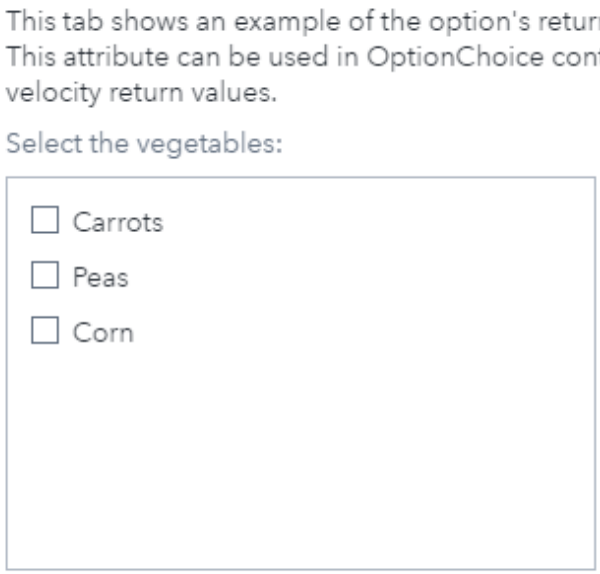
- When the data type is number, the return value must be a number.
- When the data type is date, the return value must be in the ISO format (yyy-mm-dd).

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.

```
<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" returnValue="1" inputType="string">Carrots</Option>
   <Option name="peas" returnValue="2" inputType="string">Peas</Option>
   <Option name="corn" returnValue="3" inputType="string">Corn</Option>
</Options>
<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.



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.

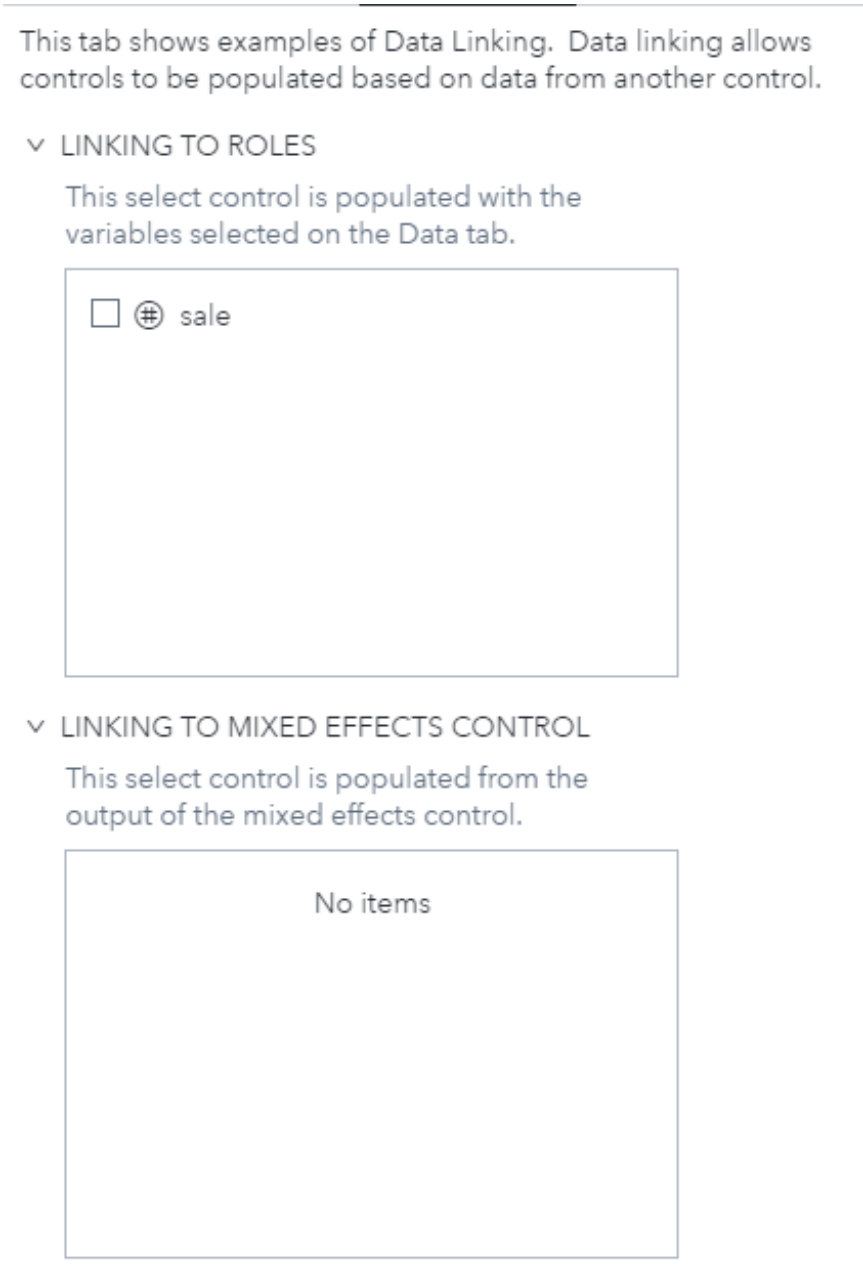
The combobox and select controls can be the recipient of the data. For these controls, specify the source by defining the `sourceLink` attribute and using the name of the source control. When the receiving option is linked to a source option, any OptionChoice children are ignored.

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.

```
<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>

...
<UI>
   <Container option="DATALINKINGTAB">
      <OptionItem option="DATALINKINGTEXT"/>
      <Group option="ROLELINKING" open="true">
         <OptionChoice option="selectRoles"/>
      </Group>
      <Group option="MEBLINKING" open="true">
         <OptionChoice option="selectMEB"/>
      </Group>
   </Container>
...
</UI>
```

If you run the code for the Advanced Task, 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.

```
<DataSources>
   <DataSource name="PRIMARYDATA">
      <Roles>
         <Role type="N" maxVars="0" order="true" minVars="0" name="NUMVAR"
```

```
            exclude="VAR">Numeric Variable</Role>
      </Roles>
   </DataSource>
</DatatSources>
<Options>
   <Option name="roleList" inputType="select" sourceLink="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 117.

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.

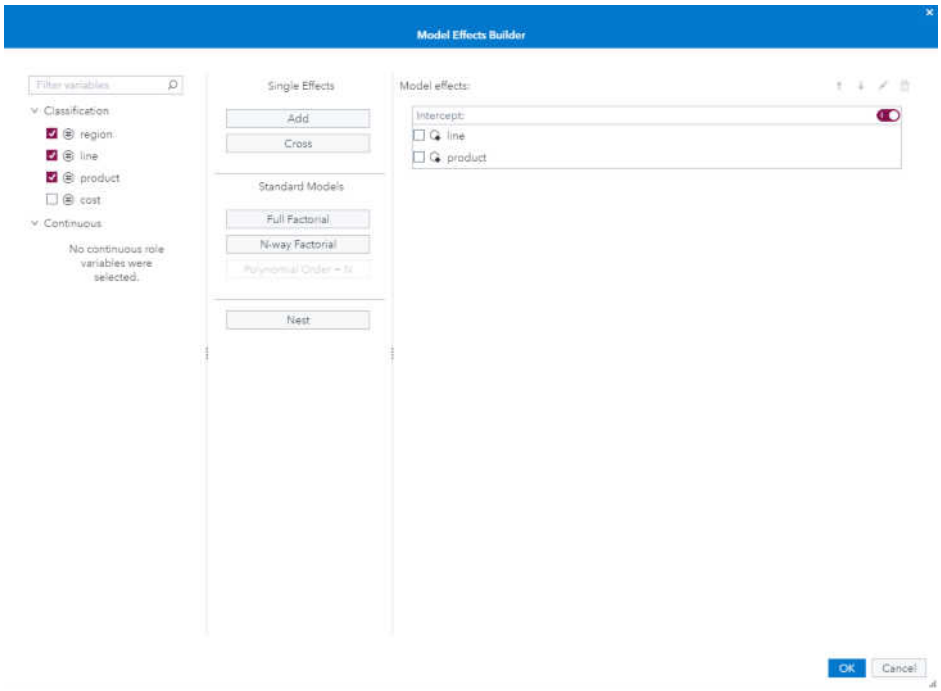
In this example, the mixedeffects control is named MEC. In the select control, the `sourceLink` attribute links to MEC, and the `sourceType` attribute specifies the ‘filterClassification’ filter. As a result, only classification effects appear in the source control.

```
<Options>
   <Option name="mec" inputType="mixedeffects" roleContinuous="CONTVARS"
      roleClassification="CLASSVARS"/>
   <Option name="mecList" inputType="select" sourceLink="MEB"
      sourceType="filterClassification"/>
</Options>
```

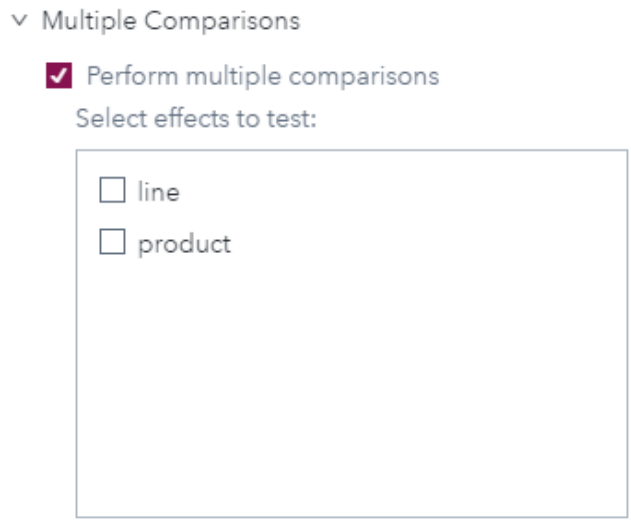
The Velocity variable that is created for the select control is $mecList. The contents of the $mecList variable mimic the output of the mixed effects control. For more information, see “mixedeffects” on page 126.

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:

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

1<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>
```

```
2 <Option indent="1" inputType="select" multiple="true"
name="multCompareList"
   sourceLink="mixedEffects" sourceType="filterClassification">
   Select effects to test</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.

Specifying a Help Message

A help message can be associated with most options as well as roles by using the `helpMessageRef` attribute. This attribute can refer to a string or a markdown option. When a help message is associated with an option, a help icon is displayed to the right of the option's label.

This example shows adding a help message to the textbox input type.

```
<Options>
   <Option name="Cont1" inputType="string">TAB</Option>
   <Option name="helpString" inputType="string">This is a helpful
message</Option>

  <Option name="helpMarkdown" inputType="markdown">
Numbered list:

  1. wash
  2. rinse
  3. repeat

## Links

[example]: http://sas.com "Optional title here"

This is an example of a reference-style link: [link][example].

[link text](http://support.sas.com)

## Images

![Software](https://www.sas.com/en_us/solutions/analytics/_jcr_content/
```

```
   par/tabwrapper_696d/tabwrapperpar/tab_4a95/tabpar/
styledcontainer_d4c2/
   par/styledcontainer_fe28/par/image_b3d6.img.png/1527781906411.png)
  </Option>

  // Wire the helpMarkdown into the 'textbox' input type using the
  // 'helpMessageRef' attribute
  <Option name="txtMarkdown" inputType="textbox"
helpMessageRef="helpMarkdown">
      Click the '?' to see the help message in markdown:</Option>

  // Wire the helpMarkdown into the 'textbox' input type
  // using the 'helpMessageRef' attribute
  <Option name="txtString" inputType="textbox"
helpMessageRef="helpString">
      Click the '?' to see the help message as a simple string:</
Option>

</Options>
</Metadata>

<UI>
  <Container option="Cont1">
     <OptionItem option="txtMarkdown"/>
     <OptionItem option="txtString"/>
  </Container>
</UI>
```

4

Working with the UI Element

About the UI Element 79
Option References 79
Containers 80
Example: UI Element from Sample Task 81

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.

Option References

To include an option in the UI section, use one of these option tags.

Child	Description
`DataItem`	A reference to an input data source. This tag has only one attribute, `data`. The string for this option is the value of the `string` input type in the `Metadata` element.
`RoleItem`	A reference to a role. This tag has only one attribute, `role`. The string for this option is the value of the `string` input type in the `Metadata` element.

Child	Description
`OptionItem`	A reference to an option that has a single state. This type of option is either on or off, or has a single value (such as a series of radio buttons). This tag takes the `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.
`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` ■ `multientry` ■ `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.

Containers

Options can be grouped into a container (which appears as a tab in the user interface) or a group (which appears as a collapsible pane in the window). By default, the options are laid out linearly. To lay out the option horizontally, use the `HorizontalLayout` element.

Child	Description
`Container`	A page or 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.

Child	Description
`Group`	A title for a group of options. The UI engine displays these options sequentially. The group can be open or collapsed. 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="true"`, and the group is open in the user interface. To collapse the contents of a group by default, specify `open="false"`.
`HorizontalLayout`	Options are displayed horizontally. The `HorizontalLayout` element has only one attribute, `option`. The `option` attribute is optional. You need to specify it only if the `HorizontalLayout` tag needs to be targeted through a dependency.

Example: UI Element from Sample Task

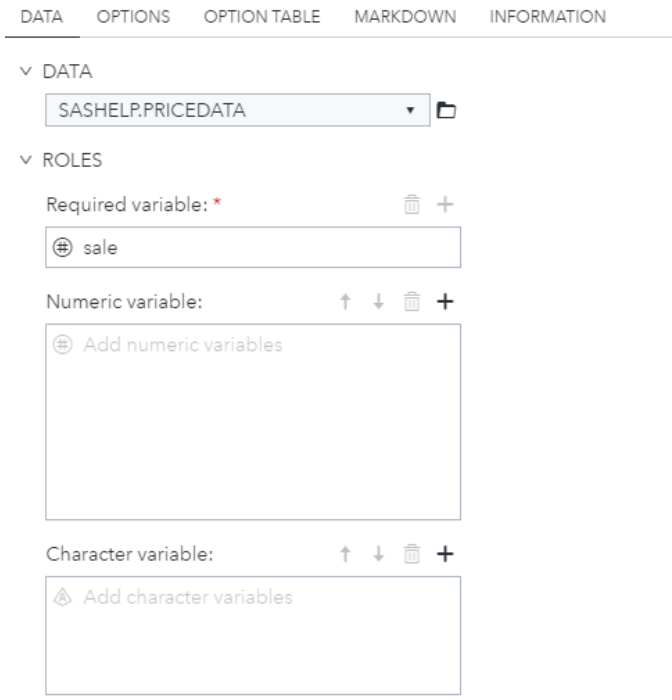
The code for the Sample Task creates a group for each input type. Here is the code for the first three groups:

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

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

      <Group option="GROUPCHECK">
         <OptionItem option="labelCheck"/>
         <OptionItem option="chkEXAMPLE"/>
      </Group>

      <Group option="GROUPCOLOR">
         <OptionItem option="labelCOLOR"/>
```

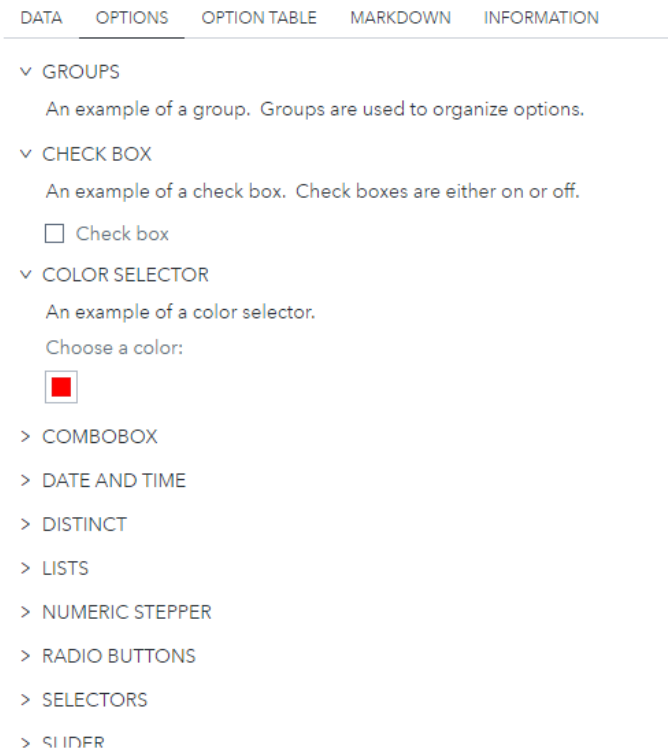
```
            <OptionItem option="colorEXAMPLE"/>
        </Group>

...
    </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 includes code to create the remaining groups on the **Options** tab.)


DATA
OPTIONS
OPTION TABLE
MARKDOWN
INFORMATION
GROUPS
An example of a group. Groups are used to organize options.
CHECK BOX
An example of a check box. Check boxes are either on or off.
Check box
COLOR SELECTOR
An example of a color selector.
Choose a color:
COMBOBOX
DATE AND TIME
DISTINCT
LISTS
NUMERIC STEPPER
RADIO BUTTONS
SELECTORS
SLIDER

5

Working with the Dependencies Element

About the Dependencies Element . . . 85
Notes on Dependencies . . . 87
Creating Dependencies for Container and Group Elements . . . 89
Using Radio Buttons as Targets of Dependencies . . . 90
Example 1: Selecting a Check Box to Show a Group of Options . . . 91
Example 2: Using Radio Buttons to Create Dependencies . . . 93
About This Example . . . 93
Selecting the Show/Hide Options Button . . . 95
Selecting the Enable/Disable Options Button . . . 96
Selecting the Set Values Button . . . 98
Example 3: Using Combobox Controls . . . 99
Using a Value to Show or Hide Additional Options . . . 99
Using a Value to Enable or Disable Additional Options . . . 101
Using a Value to Set the Value of Another Option . . . 103
Using the OptionsDependencies Element . . . 104
About the OptionsDependencies Element . . . 104
OptionDependency Element . . . 104
Example: OptionDependencies Element . . . 105

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 5.1 Attributes for the Target Element

Attribute	Description
option	references the Velocity option that receives the action. Valid values are OptionItem, Role, OptionChoice, or Group element.
conditionResult	specifies when to execute the action. The valid values for this attribute are true and false. ■ If the condition is true and conditionResult="true", the action is executed. ■ If the condition is false and conditionResult="false", the action is executed. ■ If the value of the condition and conditionResult do not match (for example, one is true and one is false), the action is ignored.
action	specifies the action to execute. Here are the valid values: ■ show ■ hide ■ enable ■ disable ■ set If the value of the action attribute is set, you must also specify these two attributes: □ The property attribute refers to the attribute of an element that was created from the metadata. The option element in the metadata has an inputType attribute that specifies what UI element is created. **Note:** Here are a few exceptions: ■ In the UI element, any RoleItem element cannot be the target of a dependency where action="set". ■ The required, width, indent, and variable (for the radio input type) attributes are invalid values for the property attribute of a Target element. □ The value attribute is the value to use for the target of the property attribute. If the value attribute targets an item with the select input type, the value attribute can accept a single value or a comma-separated list of values. **Note:** If the dependency has a comma-separated list of values and the select element that the dependency

Attribute	Description
	targets is set to `multiple="false"`, only the first value in the comma-separated list is evaluated. The rest of the values in the list are ignored.

In this example, the OBSHEADING text field is enabled only when the OBS checkbox is selected.

```
<UI>
   <Container option="basic options">
      <OptionItem option="OBS"/>
      <OptionItem option="OBSHEADING"/>
   </Container>
</UI>
<Dependencies>
   <Dependency condition="$OBS=='1'">
      <Target conditionResult="true" option="OBSHEADING"
action="enable"/>
      <Target conditionResult="false" option="OBSHEADING"
action="disable"/>
   </Dependency>
</Dependencies>
```

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:

```
<UI>
   <Container option="options">
```

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

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

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

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

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

Creating Dependencies for Container and Group Elements

A container or 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" />
   </Container>

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

<Dependencies>
1<!-- Correct -->
   <Dependency condition="$CheckBoxEnableTargetGroup=='1'">
      <Target conditionResult="true" option="targetGroup"
action="show"/>
      <Target conditionResult="false" option="targetGroup"
action="hide"/>
   </Dependency>
   <Dependency condition="$CheckBoxEnableTargetGroup=='1' &amp;&amp;
$CHECKBOX=='1'">
      <Target conditionResult="true" option="INPUTTEXT"
action="enable"/>
      <Target conditionResult="false" option="INPUTTEXT"
action="disable"/>
   </Dependency>

2<!-- Incorrect -->
   <Dependency condition="$CheckBoxEnableTargetGroup=='1'">
      <Target conditionResult="true" option="targetGroup"
action="show"/>
      <Target conditionResult="false" option="targetGroup"
action="hide"/>
```

```
   </Dependency>
   <Dependency condition="$CHECKBOX=='1'">
      <Target conditionResult="true" option="INPUTTEXT"
action="enable"/>
      <Target conditionResult="false" option="INPUTTEXT"
action="disable"/>
   </Dependency>
</Dependencies>
```

1. 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.
2. 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

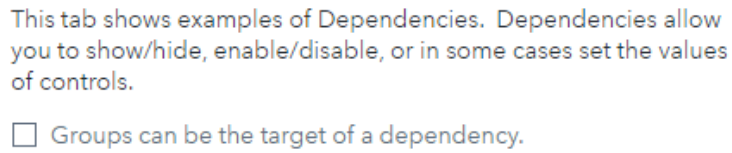
From the Advanced Task, selecting the **Groups can be the target of a dependency** check box determines whether the options under the **Group of Controls** heading are available.

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.

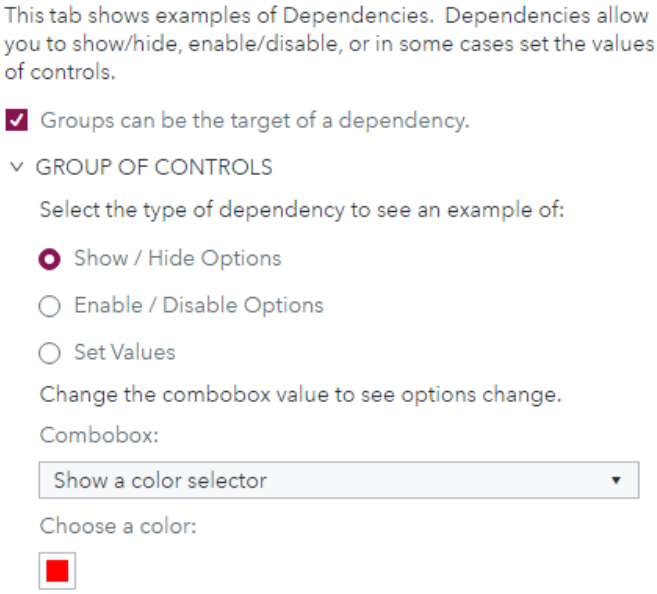
```
<Option name="DEP_CBX" inputType="checkbox" defaultValue="1">Groups
can be the
   target of a dependency.</Option>
<Option name="DEPENDENCYGROUP" inputType="string">GROUP OF CONTROLS</
Option>
```

```
<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:



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:



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:

```
<Option name="radioShowHide" variable="radioChoice" defaultValue="1"
   inputType="radio">Show / Hide Options</Option>
<Option name="radioEnableDisable" variable="radioChoice"
inputType="radio">
   Enable / Disable Options</Option>
<Option name="radioSetValue" variable="radioChoice" inputType="radio">
   Set Values</Option>
<Option name="labelShowChange" inputType="string">Change the combobox
value
   to see options change.</Option>
<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>
<Option name="colorControl" defaultValue="red" inputType="color">Choose
   a color</Option>
<Option name="dateControl" inputType="datepicker" format="monyy7.">
   Choose a date:</Option>
<Option name="sliderControl" defaultValue="80.00" inputType="slider"
   discreteValues="14" minValue="-10" maxValue="120">Slider with
buttons</Option>
<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>
<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>
<Option name="labelShowSet" inputType="string">Change the combobox
value
   to change the value of the checkbox.</Option>
<Option name="comboSetChange" defaultValue="valueSetCheck"
inputType="combobox"
   width="100%">Combobox</Option>

...


<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="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"/>
   <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>

<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="hide" conditionResult="true"
option="sliderControl"/>

   <Target action="show" conditionResult="true" option="labelShowSet"/>
   <Target action="show" conditionResult="true"
option="comboSetChange"/>
   <Target action="show" conditionResult="true"
option="checkboxCheckUncheck"/>
</Dependency>
```

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.

```
<Option name="radioShowHide" variable="radioChoice" defaultValue="1"
   inputType="radio">Show / Hide Options</Option>
```

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

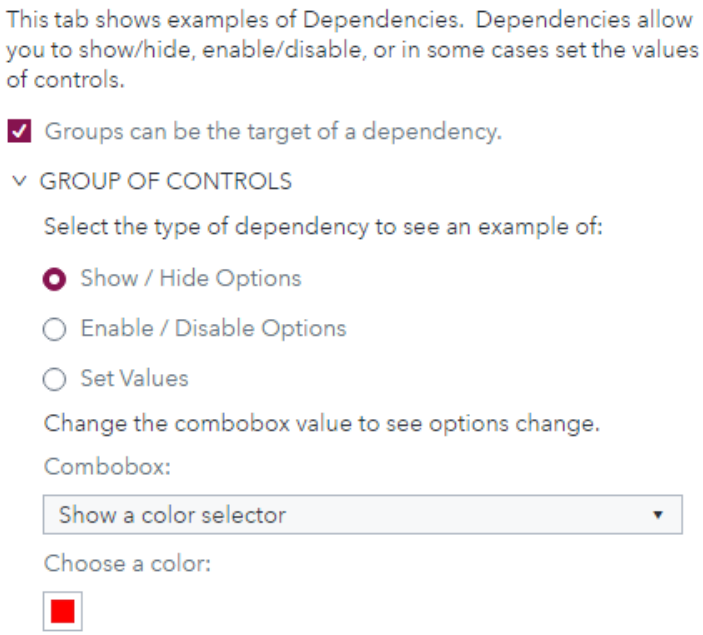
```
<Dependency condition="$radioChoice == 'radioShowHide'">
   <Target action="show" conditionResult="true"
option="labelShowChange"/>
   <Target action="show" conditionResult="true"
option="comboShowChange"/>
   <Target action="hide" conditionResult="true"
option="labelEnableChange"/>
   <Target action="hide"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

   <Target action="hide" conditionResult="true"
option="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>
```

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

```
<Option name="labelShowChange" inputType="string">Change the combobox
value
   to see options change.</Option>
<Option name="comboShowChange" defaultValue="valueShowColor"
inputType="combobox"
   width="100%">Combobox:</Option>
```

Here are the options that are available when the **Show/Hide Options** radio button is selected:



Selecting the Enable/Disable Options Button

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

```
<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:

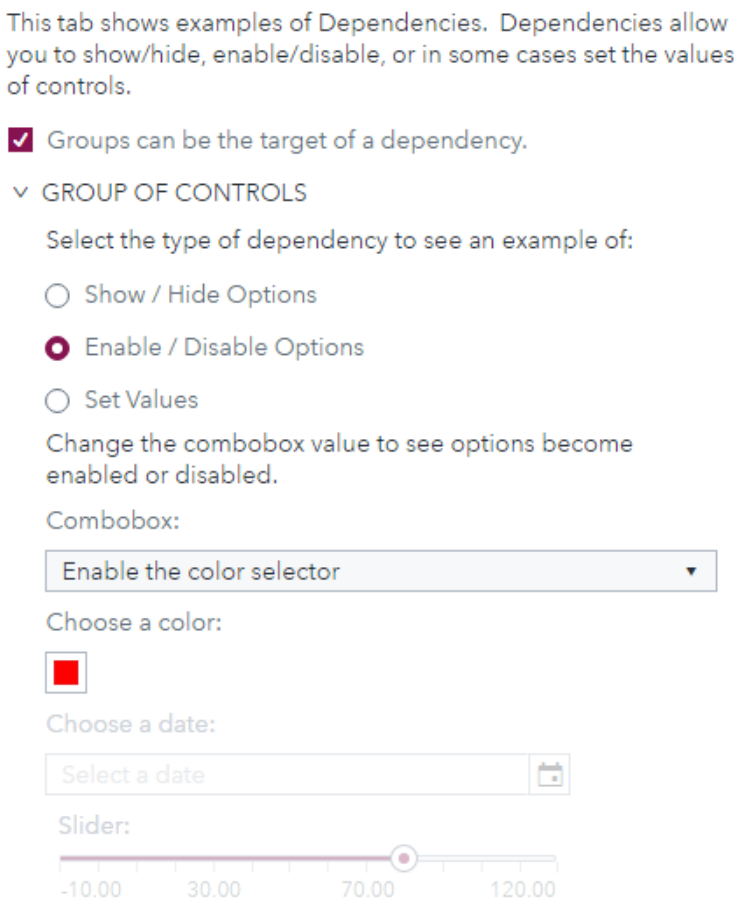
```
<Dependency condition="$radioChoice == 'radioEnableDisable'">
   <Target action="show" conditionResult="true"
option="labelEnableChange"/>
   <Target action="show" conditionResult="true"
option="comboEnableChange"/>
   <Target action="hide" conditionResult="true"
option="labelShowChange"/>
```

```
   <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>
```

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

```
<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:



Selecting the Set Values Button

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

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

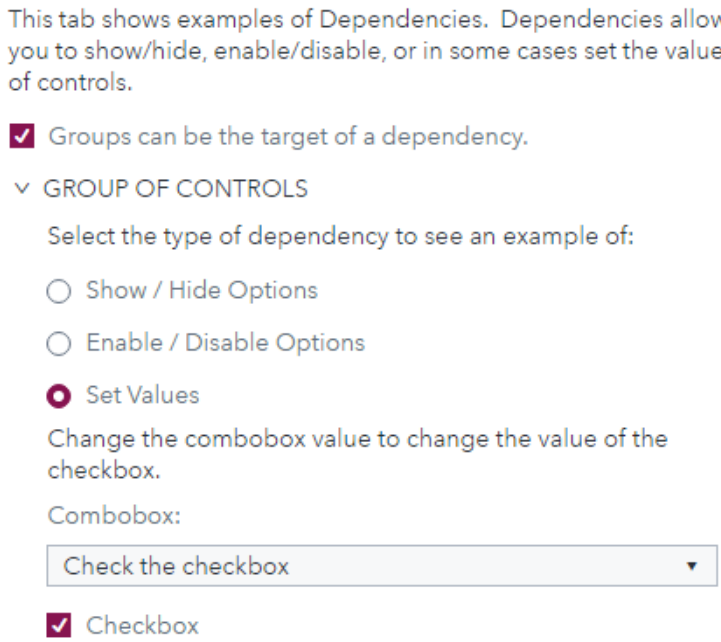
When the **Set Values** button is selected, the conditions for this dependency are met:

```
<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="hide" conditionResult="true"
option="sliderControl"/>

   <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:

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



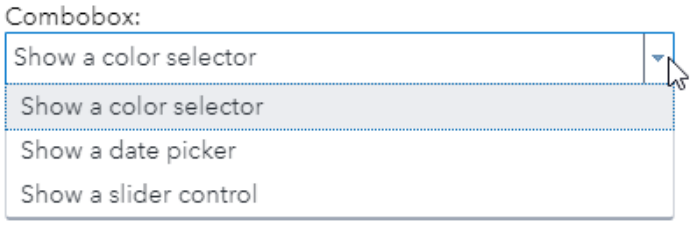
Example 3: Using Combobox Controls

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:

```
<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:

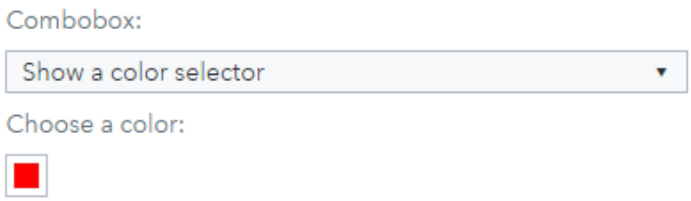


If you select **Show a color selector** from the combobox control, the conditions for this dependency are met:

```
<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.

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

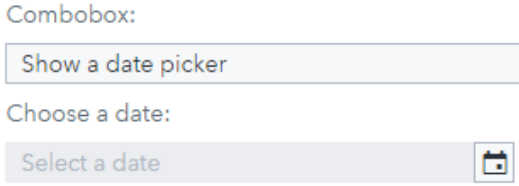


If you select **Show a date picker** from the combobox control, the conditions for this dependency are met:

```
<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.

```
<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:

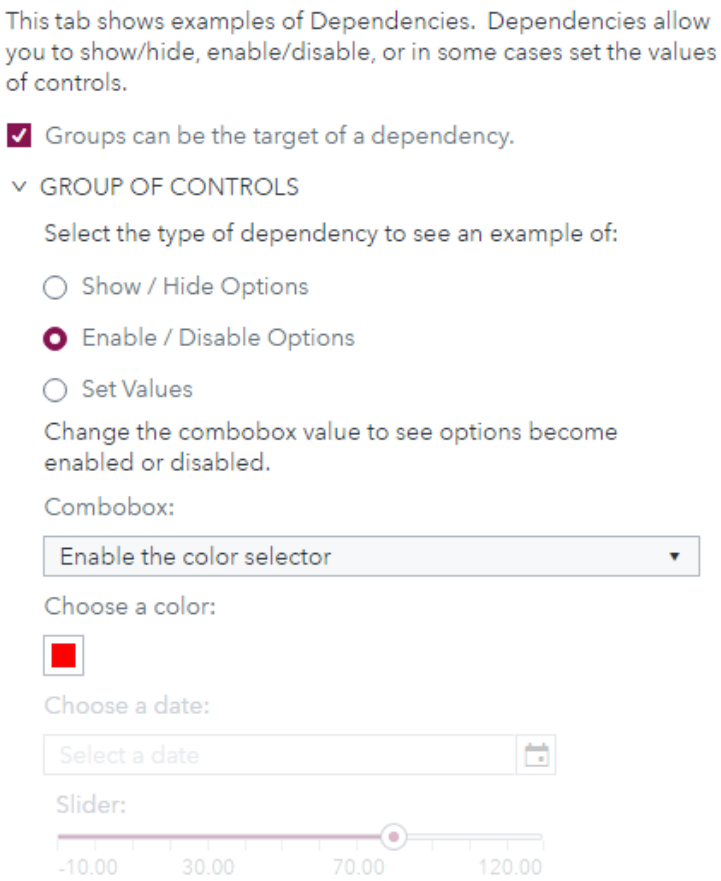
```
<Option name="comboEnableChange" defaultValue="valueEnableColor"
   inputType="combobox" width="100%">Combobox:</Option>
<Option name="valueEnableColor" inputType="string">Enable the color
   selector</Option>
<Option name="valueEnableDate" inputType="string">Enable the date
picker</Option>
<Option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

```
<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:



The user interface shows the colorControl (labeled **Choose a color**), the dateControl (labeled **Choose a date**), and the sliderControl (labeled **Slider**) 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:

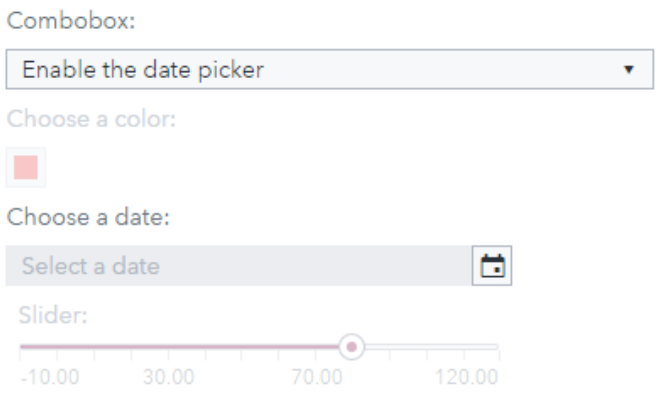
```
<Dependency condition="$comboEnableChange == 'valueEnableColor'">
   <Target sction="enable" conditionResult="true"
option="colorControl"/>
   <Target action="disable" conditionResult="true"
option="dateControl"/>
   <Target action="disable" conditionResult="true"
option="sliderControl"/>
</Dependency>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

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

```
<Dependency condition="$comboShowChange == 'valueShowDate'">
   <Target action="disabel" 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.

```
<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:

```
<Option name="comboSetChange" defaultValue="valueSetCheck"
inputType="combobox"
   width="100%">Combobox:</Option>
<Option name="valueSetCheck" inputType="string">Check the checkbox</
Option>
<Option name="valueSetUncheck" inputType="string">Uncheck the
checkbox</Option>
```

The code also defines the **Checkbox** check box. Because the defaultValue attribute is set to 1 for the checkboxCheckUncheck control, this check box is selected by default.

```
<Option name="checkboxCheckUncheck" inputType="checkbox"
defaultValue="1">
   Checkbox</Option>
```

When the **Check the checkbox** option is selected for the combobox control, this dependency is met:

```
<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.

Using the OptionsDependencies Element

About the OptionsDependencies Element

The `OptionsDependencies` element is an alternate way to define dependencies. Although this element can be more efficient to use, it is not evaluated by the Velocity engine.

When using the `OptionsDependencies` element, remember these items:

- The `OptionsDependencies` element is intended to be simple. Only the `ShowTarget` and `EnableTarget` elements are available. There are no show or hide attributes. When the value of the `TriggerOption` matches the value of `TriggerValue`, everything is displayed. All other trigger values hide or disable the controls.
- If two different `TriggerOption` values match, the result is the union of both conditions.
- You can target groups with option dependencies instead of targeting a large number of options with a single option dependency. However, you should use either target groups or target individual elements. Mixing these two behaviors can lead to unexpected results.
- Use either `Dependencies` elements or `OptionsDependencies` elements. If the task has any `Dependencies` elements, the `OptionDependencies` are ignored.

OptionDependency Element

The `OptionsDependencies` element can have 0 to n `OptionDependency` elements. Each `OptionDependency` element defines the relationship between the trigger option and any targets for the trigger. The `OptionDependency` element has no attributes. It must contain one `TriggerOption` element and 1 to n `TriggerValue` elements.

Table 5.2 *Elements for the OptionDependency Element*

Element	Description
`TriggerOption`	The `TriggerOption` element has one required attribute, `option`. Use this element to specify the option whose host value is compared to the `TriggerValue` elements to determine whether to show or hide or enable and disable an option. The `TriggerOption` element has no children.

Element	Description
TriggerValue	The TriggerValue element has one attribute, value. This value is compared to the value of the option in the TriggerOption element. If the value in the TriggerOption is equal to the value of the TriggerValue element, the target children are shown or enabled. The TriggerValue element can have 1 to n children that are either ShowTarget or EnableTarget elements. ■ The ShowTarget element has one required attribute, option. Use this attribute to specify the name of the option that should be shown if the value matches the TriggerOption value. The ShowTarget element has no children. ■ The EnableTarget element has one required attribute, option. This attribute is used to specify the name of the option that should be enabled if the value matches the TriggerOption value. The EnableTarget element has no children.

Example: OptionDependencies Element

In this example, when the value of checkbox is 1, the inputtext control is shown. If the value of the checkbox is not 1, the inputtext control is hidden.

```
<Metadata>
   <Options>
      <Option name="cbxValue" inputType="checkbox" >Specify value</
Option>
      <Option name="txtValue" inputType="inputtext">Enter a value</
Option>
   </Options>
</Metadata>
<UI>
          <OptionItem option="cbxValue"/>
          <OptionItem option="txtValue"/>
</UI>
<OptionDependencies>

          <OptionDependency>
                    <TriggerOption option="cbxValue"/>
                    <TriggerValue value="true">
                              <ShowTarget option="txtValue"/>
                    </TriggerValue>
          </OptionDependency>

</OptionDependencies>
```

6

Working with the Requirements Element

About the Requirements Element 107
Example: Using a Requirements Element for Roles 108

About the Requirements Element

The `Requirements` element specifies a condition that must be met in order for code to be generated for the entire task.

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 2.0 User's Guide*.

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

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

Example: Using a Requirements Element for Roles

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

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

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

7

Understanding the Code Template

About the Code Template . . . 110

Using Predefined Velocity Variables . . . 110
- Predefined Velocity Variables . . . 110
- Floating Point Math . . . 111

Working with the DataSource Element in Velocity . . . 112
- About the DataSource Element . . . 112
- columnExists Method . . . 112
- get Method . . . 113
- getEngine Method . . . 113
- getRowsCount Method . . . 114
- getDistinctCount Method . . . 114
- getDistinctValues Method . . . 115
- getLibrary Method . . . 115
- getTable Method . . . 116
- getWhereClause Method . . . 116
- getDataType Method . . . 117

Working with Role Elements in the Velocity Code . . . 117
- How Role Elements Appear in the Velocity Code . . . 117
- get Method . . . 118
- getDistinctCount Method . . . 119
- getDistinctValues Method . . . 119

How the Options Elements Appear in the Velocity Code . . . 120
- checkbox . . . 120
- color . . . 121
- combobox . . . 121
- datepicker . . . 122
- daterange . . . 122
- datetimepicker . . . 123
- distinct . . . 124
- dualselector . . . 124
- inputtext . . . 125
- markdown . . . 126

mixedeffects . . . 126
monthpicker . . . 133
multientry . . . 134
numbertext . . . 135
numericrange . . . 135
numstepper . . . 135
outputdata . . . 136
optiontable . . . 136
passwordtext . . . 137
quarterpicker . . . 138
radio . . . 138
sasserverpath . . . 139
select . . . 140
slider . . . 141
string . . . 141
textbox . . . 141
timepicker . . . 142
validationtext . . . 143
weekpicker . . . 143

Working with the OutputData Control in Velocity . . . **144**
About the OutputData Control . . . 144
get Method . . . 145
getLibrary Method . . . 145
getTable Method . . . 146

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 2.0 scripting language. The string output is produced using this scripting language.

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

Using Predefined Velocity Variables

Predefined Velocity Variables

Here are the predefined Velocity variables:

Variable	Description
$sasOS	The operating system for the SAS server.
$sasVersion	The version of the SAS server.
$MathTool	The Java object for the Apache Velocity MathTool. For more information, see “Floating Point Math” on page 111.
$CTMUtil	This tool holds a Java object that provides common utility methods for the common task models.
$CTMMathUtil	This tool provides access to basic math utilities.

Floating Point Math

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

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

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

Working with the DataSource Element in Velocity

About the DataSource Element

You can specify multiple `DataSource` elements 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 following methods to get more information about the data source.

columnExists Method

Short Description	Determines whether the specified value already exists as the name of a column in the data source.
Parameter	input the input string that you want to check to see whether it exists.
Return Value	Returns a Boolean value that specifies whether the column already exists.

Example	`<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

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

getEngine Method

Short Description	Returns the value of the engine for the table's library.
Return Value	Returns an engine value such as `CAS` or `V9`.

Example	`<DataSource name="DATASOURCE"></DataSource>`
	
	`$DATASOURCE.getEngine()  /* if dataset is Sashelp.Cars, return value would be V9 */`

getRowsCount Method

Short Description	Returns the number of rows in the currently selected data source.
Return Value	Returns a number. In many cases, –1 is returned when the number of rows is unavailable.

Example	`<DataSource name="DATASOURCE">`
	`  <Roles>`
	`  <Role name="analysisVariables" type="A" maxVars="0"`
	`  minVars="0">Analysis variables</Role>`
	`  </Roles>`
	`</DataSource>`
	
	`#if ($DATASOURCE.getRowsCount() > 0) ... #end /* if dataset is Sashelp.Class,`
	`  return value would be 19*/`

getDistinctCount Method

To use this method, specify `fetchDistinct = "true"` in the `Role` element. For more information, see "Working with the Roles Element" on page 13.

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.
Return Value	Returns the number of distinct values. If there are no distinct values or the distinct values are not available, the return value is –1.

Example	`<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() > 0 && $DATASOURCE.getDistinctCount($VAR[0]) > 0  ... #end
```

getDistinctValues Method

To use this method, specify `fetchDistinct = "true"` in the `Role` element. For more information, see "Working with the Roles Element" on page 13.

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.
Return Value	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.

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

getLibrary Method

Short Description	Returns the name of the library for the data source.
Return Value	Returns a string that contains the name of the library for the data source.
Example	`<DataSource name="DATASOURCE">` `  <Roles>`

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

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

getTable Method

Short Description	Returns the table name for the data source.
Return Value	Returns a string that contains the table name for the data source.

Example	`<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.

Short Description	Returns the filter of the currently assigned data source
Return Value	Returns a string that contains the filter of the currently assigned data source

Example	`<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

Short Description	Returns the type of data set. This value corresponds to the 'typemem' value in Sashelp.Vtable.
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>`
	
	`$DATASOURCE.getDataType()`

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]`. Indexing starts at 0. 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:

Attribute	Description
`format`	Specifies the SAS format that is assigned to the variable.
`informat`	Specifies the SAS informat that is assigned to the variable.
`length`	Specifies the length that is assigned to the variable.
`type`	Specifies the type of variable. Valid values are `Numeric` or `Char`.
`value`	Specifies the name of the variable.

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>
   <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;
RUN;
#end
]]>
</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.

getDistinctCount Method

To use this method, specify `fetchDistinct = "true"` in the `Role` element. For more information, see "Working with the Roles Element" on page 13.

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.
Return Value	Returns the number of distinct values. If there are no distinct values or the distinct values are not available, the return value is –1.

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

getDistinctValues Method

To use this method, specify `fetchDistinct = "true"` in the `Role` element. For more information, see "Working with the Roles Element" on page 13.

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.
Return Value	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.

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

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 names in the `Option` element. 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>
<Code Template>
  <![CDATA[
#if ($PRINTNUMROWS == '1')
    N
```

```
#end]]>
</CodeTemplate>
```

color

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

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

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

Note: When using the `combobox` input type, remember these items:

- If the `returnValue` attribute is defined, the Velocity variable holds the `returnValue` instead of `name`.
- If the all values item is selected, the Velocity value is '-ALL-'.
- If the missing values item is selected, the Velocity value is '_BLANK_'.

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

```
<Options>
   <Option name="HEADING" defaultValue="default"
      inputType="combobox">Direction of heading:</Option>
   <Option name="default" inputType="string">Default</Option>
   <Option name="horizontal" inputType="string">Horizontal</Option>
   <Option name="vertical" inputType="string">Vertical</Option>
</Options>
<UI>
   <Container option="OPTIONSTAB">
      <OptionChoice option="HEADING">
         <OptionItem option="default"/>
         <OptionItem option="horizontal"/>
         <OptionItem option="vertical"/>
```

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

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. The date is in the ISO format yyyy-MM-dd. To convert this value to a SAS date value, use the E8601DA. informat.

This example generates SAS code to read in the date value into the macro variable, mydate. The date is written to the log in both its raw date form and the formatted MONYY7. form.

```
<Options>
   <Option name="oDateLong" inputType="datepicker" defaultValue="2018-09-05"
       displayFormat="long" minDate="2018-09-01"
       maxDate="2018-09-30" helpMessageRef="datePickerLabel">
       Choose a date:
   </Option>

</Options>
<CodeTemplate>
   <![CDATA[
%let mydate=%SYSFUNC(InputN(%QUOTE($oDateLong), %QUOTE(e8601da.)));

data _null_;
rawdate=&mydate;

frmdate = PUTN(rawdate, 'monyy7.');
put rawdate= frmdate= ;

run;
#end
]]>
</CodeTemplate>
```

daterange

The Velocity variable for the `daterange` variable holds two values: `fromValue` and `toValue`. If these range values are not set, `fromValue` and `toValue` are empty

strings. When valid values are selected, this variable contains the values in formats that are consistent with each date type.

This table lists the input and output format for each date type and the informat to use to convert the return values to SAS date values.

Table 7.1 *Formats for Each Data Type*

dateType	Input and Output Format	Example	SAS Informat
Date	yyyy-MM-dd	2019–09–05	
Time	HH:mm:ss	19:15:22	ANYDTTME.
DateTime	yyyy-MM-ddTHH:mm:ss	2019–09–05T19:15:22	E8601DA.
Week	yyyy-Www	2018–W45	WEEKV.
Month	yyyy-MM	2018–09	ANYDTDTE7.
Quarter	yyyy-Qq	2018Q3	YYQ7.
Year	yyyy	2019	

datetimepicker

The Velocity variable for the `datetimepicker` input type holds the information for the datetimepicker control. By default, this variable is an empty string. When a user selects a date and time or if a default value is supplied, the variable holds the date and time in the ISO format yyyy-MM-ddTHH:mm:ss. To convert this to a SAS datetime value, use the E8601DA. format.

This example generates SAS code to read the value into the macro variable, mydate. The value is then written to the log in both the raw datetime value and the formatted MONYY7. value.

```
<Options>
    <Option name="oDateTimeLong" inputType="datetimepicker"
required="true"
   defaultValue="2018-09-05T19:15:22" displayFormat="long"
   minDate="2018-09-01" maxDate="2018-09-30" use24HourTime="true"
   helpMessageRef="helpMessage">
        Select a Date and Time:
    </Option>
</Options>
<CodeTemplate>
   <![CDATA[
#if( $oDateTimeLong)
%let mydate=%SYSFUNC(InputN(%QUOTE($oDateTimeLong), %QUOTE(e8601da.)));

data _null_;
```

```
rawdate=&mydate;

frmdate = PUTN(rawdate, 'monyy7.');
put rawdate= frmdate= ;

run;
#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)`.

```
<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.

Note: When working with the `dualselector` input type, remember these items:

- If the `returnValue` attribute is defined, the Velocity variable holds the return value instead of the name.
- If the data type is numeric or date, the Velocity variable contains the date in the ISO format or the numeric value.
- If the data type is set to date, any static values or return values should be in the ISO format.
- If the all values item is selected, the Velocity variable is '-ALL-'.

- If the missing values item is selected, the Velocity variable is '_BLANK_'.

This example is for a dualselector control that contains three values: Choice1, Choice2, and Choice3.. Any or all of these values can be selected. Only the values that are selected in the dualselector control appear in the Velocity code.

```
<Options>
   <Option inputType="string" name="Choice1">First Choice</Option>
   <Option inputType="string" name="Choice2">Second Choice</Option>
   <Option inputType="string" name="Choice3">Third Choice</Option>
   <Option inputType="string" name="FirstHundredDayOf2019"
      returnValue="2019-04-10">100th day of 2019</Option>
</Options>
   <OptionChoice name="ANOTHERLIST" inputType="dualselector"
       defaultValue="Choice1,FirstHundredDayOf2019"
       editable="true" dataType="date" allowAllValues="true"
          allowMissingValues="true" >
       <OptionItem option="Choice1"/>
       <OptionItem option="Choice2"/>
       <OptionItem option="Choice3"/>
      <OptionItem option="FirstHundredDayOf2019"/>
   </OptionChoice>
...

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

inputtext

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 label** text box. If the user enters **`Student Number`** into the **Column label** text box, the output is `OBS="Student Number"`.

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

markdown

No Velocity variable is created for the markdown input type.

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:

Member	Description
`randomEffectsSetCount`	Specifies the number of random effects model sets that were created.
`repeatedEffectsSetCount`	Specifies the number of repeated effects model sets that were created.
`fixedEffectsCount`	Specifies the number of fixed effects that were created.
`fixedContinuousMainEffectsCount`	Specifies the number of main fixed effects that were created for a continuous variable.
`fixedClassificationMainEffectsCount`	Specifies the number of main fixed effects that were created for a classification variable.
`fixedInterceptValue`	Specifies the value of the intercept of the fixed effects model set. Valid values are `true`, `false`, or null.
`fixedModelsetInvalidStateCount`	Specifies the number of fixed effects model sets with an invalid context.
`randomModelsetInvalidStateCount`	Specifies the number of random effects model sets with an invalid context.

Member	Description
repeatedModelsetInvalidStateCount	Specifies the number of repeated effects model sets with an invalid context.
meansModelsetInvalidStateCount	Specifies the number of means effects model sets with an invalid context.
zeroInflatedEffectsCount	Specifies the number of zero-inflated effects that were created.
zeroInflatedEffectsSetCount	Specifies the number of zero-inflated effects model sets that were created.
zeroInflatedModelsetInvalidStateCount	Specifies the number of zero-inflated effects model sets with an invalid context.

The 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.

Member	Description
emtype	Specifies the type of model.
intercept	Specifies whether the intercept is visible to the user. Valid values are true, false, or null.
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 ■ hierarchyTerms: an array in which each element contains the variables used for the term in that level of the hierarchy. The first element corresponds to the topmost parent term in the hierarchy. For example, if you create the hierarchy State > District > School, the hierarchyTerms are [[State],[District],[School]]. ■ levelInclusionIndices: an array of indices that correspond to each level of the hierarchy. These indices indicate whether the nested effect term for that hierarchy level should be in included in the model.

Member	Description
	For example, your hierarchy is State > District > School > Teacher. The model terms are District(State), School (State District), and Teacher (State District School). When you include all of these terms in your model, `levelInclusionIndices=[1,2,3]`. If you choose to exclude the School(State District) term from the model, `levelInclusionIndices=[1,3]`.
Additional Members for Random and Repeated Effects	
`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 ■ `hierarchyTerms`: an array in which each element contains the variables used for the term in that level of the hierarchy. The first element corresponds to the topmost parent term in the hierarchy. For example, if you create the hierarchy State > District > School, the hierarchyTerms are [[State],[District],[School]]. ■ `levelInclusionIndices`: an array of indices that correspond to each level of the hierarchy. These indices indicate whether the nested effect term for that hierarchy level should be in included in the model. For example, your hierarchy is State > District > School > Teacher. The model terms are District(State), School (State District), and Teacher (State District School). When you include all of these terms in your model, `levelInclusionIndices=[1,2,3]`. If you choose to exclude the School(State District) term from the model, `levelInclusionIndices=[1,3]`.
`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 ■ `hierarchyTerms`: an array in which each element contains the variables used for the term in that level of the hierarchy. The first element corresponds to the topmost parent term in the hierarchy. For example, if you create the hierarchy State > District > School, the hierarchyTerms are [[State],[District],[School]].

Member	Description
	■ `levelInclusionIndices`: an array of indices that correspond to each level of the hierarchy. These indices indicate whether the nested effect term for that hierarchy level should be in included in the model. For example, your hierarchy is State > District > School > Teacher. The model terms are District(State), School (State District), and Teacher (State District School). When you include all of these terms in your model, `levelInclusionIndices=[1,2,3]`. If you choose to exclude the School(State District) term from the model, `levelInclusionIndices=[1,3]`.
`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.

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

```
<?xml version="1.0" encoding="UTF-8"?>
<Task schemaVersion="7.2">

    <Registration>
        <Name>MEC Tester</Name>
        <Description>Example task for testing the Mixed Effects Control
           component.</Description>
        <LongDescription>Example task for testing the Mixed Effects
Control
           component.</LongDescription>
        <Procedures>MIXED</Procedures>
        <Version>5.2</Version>
    </Registration>

    <Metadata>
        <DataSources>
            <DataSource name="DATASOURCE">
                <Roles>
                    <Role type="N" maxVars="0" order="true" minVars="0"
                       name="CONTVARS" exclude="CLASSVARS">Continuous</
Role>
                    <Role type="A" maxVars="0" order="true" minVars="0"
                       name="CLASSVARS"
exclude="CONTVARS">Classification</Role>
                </Roles>
            </DataSource>
```

```
        </DataSources>

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

            <Option name="MODELSTAB" inputType="string">MODEL</Option>
            <Option name="MECTESTER" inputType="string">MEC MODELS</
Option>

            <!-- how to handle the various options for new controls
            (excludeTools etc...) -->
            <Option name="mixedEffects" inputType="mixedeffects"
                    excludeTools="THREEFACT"
effects="fixedrandomrepeated"
                    roleClassification="CLASSVARS"
                    roleContinuous="CONTVARS">Optional label:</Option>
        </Options>
    </
Metadata>
                                                `

    <UI>
        <Container option="DATATAB">
            <Group option="DATAGROUP" open="true">
                <DataItem data="DATASOURCE"/>
            </Group>
            <Group option="ROLESGROUP" open="true">
                <RoleItem role="CONTVARS"/>
                <RoleItem role="CLASSVARS"/>
            </Group>
        </Container>

        <Container option="MODELSTAB">
            <Group option="MECTESTER" open="true">
                <OptionItem option="mixedEffects"/>
            </Group>
        </Container>
    </UI>

    <Dependencies>
    </Dependencies>

    <CodeTemplate>
<![CDATA[

## Macro to display model effects
#macro( displayEffectEntry $effectEntry $prefix )
    #if ( $effectEntry.effectType == 'nested' )
## this is a 'nested' effect
$prefix$effectEntry.effectType effect name is
"$effectEntry.nestedName":
- hierarchyTerms: [#foreach( $term in $effectEntry.hierarchyTerms )
      $term#if($foreach.count < $effectEntry.hierarchyTerms.size()),
#end#end];
```

```
- levelInclusionIndices: [#foreach( $layerNum in
      $effectEntry.levelInclusionIndices )
      $layerNum#if($foreach.count <
      $effectEntry.levelInclusionIndices.size()), #end#end];
- model terms: #foreach( $layerNum in
$effectEntry.levelInclusionIndices )
      #set( $outerTerm = $effectEntry.hierarchyTerms[$layerNum] )
      #foreach( $subitem1 in $outerTerm )$subitem1#if($foreach.count
         < $outerTerm.size())*#end#end
     (#foreach($subterm2 in $effectEntry.hierarchyTerms)
         #if($foreach.index < $layerNum)
      #foreach($subitem2 in $subterm2)$subitem2#if($foreach.count
         < $subterm2.size())*#end#end
      #if($foreach.index < ($layerNum) - 1) #end#end#end) #end;
    #else
## this is a 'main' or 'interaction' effect
$prefix$effectEntry.effectType effect: #foreach( $subitem in
   $effectEntry.memberSet1 )
   $subitem#if($foreach.count <
$effectEntry.memberSet1.size())*#end#end;
#end
#end

/* ======== MEC Summary Values START ======== */
#if( $mixedEffects.modelSummaryValues )

    /* Model effects set counts */
    Random Effects Set Count:
       $mixedEffects.modelSummaryValues.randomEffectsSetCount;
    Repeated Effects Set Count:
       $mixedEffects.modelSummaryValues.repeatedEffectsSetCount;
    Zero-Inflated Effects Set Count:
       $mixedEffects.modelSummaryValues.zeroInflatedEffectsSetCount;
    Zero-Inflated Effects Count:
       $mixedEffects.modelSummaryValues.zeroInflatedEffectsCount;

    /* Fixed effects information */
    Fixed Effects Count:
      $mixedEffects.modelSummaryValues.fixedEffectsCount;
    Fixed Continuous Main Effects Count:
      $mixedEffects.modelSummaryValues.fixedContinuousMainEffectsCount;
    Fixed Classification Main Effects Count:

$mixedEffects.modelSummaryValues.fixedClassificationMainEffectsCount;
    ## Legal values for modelSummaryValues.fixedInterceptValue
    ## are True, False, or null.
    #if ( $mixedEffects.modelSummaryValues.fixedInterceptValue )
        Fixed Intercept Value:
$mixedEffects.modelSummaryValues.fixedInterceptValue;
    #else
        Fixed Intercept Value: null;
    #end

    /* Model set invalid state count */
    Fixed Model Invalid State Count:
      $mixedEffects.modelSummaryValues.fixedModelsetInvalidStateCount;
```

```
    Random Model Invalid State Count:
      $mixedEffects.modelSummaryValues.randomModelsetInvalidStateCount;
    Repeated Model Invalid State Count:

$mixedEffects.modelSummaryValues.repeatedModelsetInvalidStateCount;
    Means Model Invalid State Count:
      $mixedEffects.modelSummaryValues.meansModelsetInvalidStateCount;
    Zero-Inflated Model Invalid State Count:

$mixedEffects.modelSummaryValues.zeroInflatedModelsetInvalidStateCount;

#else
    /* No summary values found. */
#end
/* ======== MEC Summary Values END ======== */


/*
 * ======== MEC Model Effects START ========
 * Number of mixed model effects sets:
$mixedEffects.mixedEffectsModels.size()
 *
 */

#foreach( $model in $mixedEffects.mixedEffectsModels )
    /* -------- Begin mixed model effects set model[$foreach.index]
--------
     *          model effects set type: $model.emtype
     *
    ## Handle Intercept only if not null --------------------------
        ## Legal values for model.intercept are True, False, or null.
    #if( $model.intercept == "True" )
        * --       This model has an intercept       --
    #elseif( $model.intercept == "False" )
        * --       This model has no intercept       --
    #end
    ## Handle Model Effects ---------------------------------------
    * --       This model effects set has $model.modelEffects.size()
model
      effect(s)       -- */
    #if( $model.modelEffects.size() > 0 )
        #foreach( $modelEffect in $model.modelEffects )
            #displayEffectEntry( $modelEffect "$foreach.count " )
        #end
    #end

    ## Handle Subject Effect --------------------------------------
    #if( $model.subjectEffect )
        /* --       This model effects set has a subject effect
-- */
        #displayEffectEntry( $model.subjectEffect "" )
    #end

    ## Handle Covariance Structure ------------------------------
    #if( $model.covarianceStructures &&
( $model.covarianceStructures.size() > 0 ) )
```

```
        /* --       This model effects set has a covariance
structure       -- */
        #set( $covStruct = $model.covarianceStructures[0])
        Type: $covStruct.csType
#if( $covStruct.csParameterValues.size() > 0 )with
           parameters: [#foreach( $subitem in
$covStruct.csParameterValues )
           $subitem#if($foreach.count <
$covStruct.csParameterValues.size()),
           #end#end]#end;
    #end

    ## Handle Group Effect ---------------------------------------
    #if( $model.groupEffect )
        /* --       This model effects set has a group effect       --
*/
        #displayEffectEntry( $model.groupEffect "" )
    #end
    /* -------- End mixed model effects set model[$foreach.index]
--------
     */
#end
/* ======== MEC Model Effects END ======== */

/*
 * ======== MEC Velocity Variable Members ========
 *
 */
/* -------- modelSummaryValues members -------- */
modelSummaryValues: $mixedEffects.modelSummaryValues;
/* -------- mixedEffectsModels members --------
 * Number of mixed model effects sets:
$mixedEffects.mixedEffectsModels.size() */
mixedEffectsModels: $mixedEffects.mixedEffectsModels;


]]>
    </CodeTemplate>

</Task>
```

monthpicker

The Velocity variable for the `monthpicker` input type holds the value for the monthpicker option. If a month is not set, this variable is an empty string. When a month is selected, this variable contains the month value in the ISO format yyyy-MM. To convert this value to a SAS date value, use the ANYDTDTE7. informat.

This example generates SAS code to read the month value into the macro variable, my date. The value is then written to the log in both the raw data form and the formatted MONYY. value.

```
<Options>
   <Option name="oMonth" inputType="monthpicker" defaultValue="2019-05"
```

```
      minValue="2019-02" maxValue="2020-11" displayFormat="long" >
      Choose a Month:
   </Option>

</Options>
<CodeTemplate>
   <![CDATA[
#if( $oMonth )
%let mydate=%SYSFUNC(InputN(%QUOTE($oMonth), %QUOTE(ANYDTDTE7.)));

data _null_;
rawdate=&mydate;

frmdate = PUTN(rawdate, 'monyy.');
put rawdate= frmdate= ;

run;
#end
#end
]]>
</CodeTemplate>
```

multientry

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

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

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

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`.

```
<Options>
   <Option name="AMT" defaultValue="1" minValue="0" maxValue="100"
      inputType="numbertext">Number to order:</Option>
</Options>
<CodeTemplate>
   <![CDATA[
AMOUNT=$AMT]]>
</CodeTemplate>
```

numericrange

The Velocity variable for the `numericrange` input type holds the fromValue and toValue for the control. If the range values are not set, both values are empty strings. When valid values are selected, this variable contains numeric values.

This example generates SAS code to read in the current values for a numeric range option. Their values are then written to the log.

```
<Options>
<Option name="oNumericRange" inputType="numericrange" minValue="-10"
   maxValue="100000" minInclusive="true" maxInclusive="true"
decimalPlaces="0,3"
   defaultFromValue="10" defaultToValue="20"
helpMessageRef="helpMessage"
   required="false" fromLabel="Low:" toLabel="High:">
   Specify the range of numbers
</Option>
</Options>
<CodeTemplate>
   <![CDATA[
#if ($oNumericRange)
%put $oNumericRange.fromValue $oNumericRange.toValue;
#end
]]>
</CodeTemplate>
```

numstepper

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".`

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

outputdata

For more information, see "Working with the OutputData Control in Velocity" on page 144.

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:

Attribute	Description
`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:

Attribute	Description
column name as defined in the `Column` element	Specifies the information specific to that column. This structure has these members: ■ `values` – an array of the current values. ■ `isValid` – a Boolean value (1 or 0) that indicates whether the column is currently valid. ■ `numValues` – the current number of values for this column.

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.

```
<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.

```
<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;
```

quarterpicker

The Velocity variable for the `quarterpicker` input type holds the current value for the quarterpicker control. If the quarter and year are not set, the variable is an empty string. When a quarter and year are selected, this variable contains a value in the yyyyQq format. To convert this value to a SAS date value, use the YYQ7. informat.

This example generates SAS code to read the quarter value into the macro variable, mydate, using the YYQ7. format. The value is then written to the log in its raw date form and as a formatted YYQ-2 value.

```
<Options>
   <Option name="oQuarter" inputType="quarterpicker"
defaultValue="2018Q2"
      minValue="2018Q1" maxValue="2020Q4" displayFormat="long" >
      Choose a Quarter:
   </Option>

</Options>
<CodeTemplate>
   <![CDATA[
#if( $oQuarter)
%let mydate=%SYSFUNC(InputN(%QUOTE($oQuarter), yyq7.));

data _null_;
rawdate=&mydate;

frmdate = PUTN(rawdate, 'yyq-2.');
put rawdate= frmdate= ;

run;
#end
]]>
</CodeTemplate>
```

radio

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

In this example, there are four radio buttons.

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

- If the third radio button is selected, the string output is `GROUPS="10"`.
- If the fourth radio button is selected, the string output is `GROUPS="4"`.

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

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.

Member	Description
path	Specifies the path of the folder or file selected.
pathLocation	Specifies the location of the chosen file. This value can be either `sascontent` or `filesystem`.
name	Specifies the name of the selected file for a file selector or the name of the selected project for a project selector. **Note:** This member does not exist for a folder selector.
fullPath	Specifies the complete path of the selected file for the file selector or the complete path of the selected project for the project selector. The full path is the combination of the path and name attributes. **Note:** This member does not exist for a folder selector.

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

```
<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>
</Options>
<CodeTemplate>
   <![CDATA[
%let folderPath=$folderSelector.path;

%let filePath=$fileSelector.path;
%let fileName=$fileSelector.name;
]]>
</CodeTemplate>
```

select

The Velocity variable for the `select` input type holds the array of selected values.

Note: When using the `select` input type, remember these items:

- If the return value is defined, the Velocity variable holds the return value instead of name.
- If the all values item is selected, the Velocity value is '-ALL-'.
- If the missing values item is selected, the Velocity value is '_BLANK_'.

This example shows a selection list that contains three options. Any or all of these options can be selected. Only the selected items are displayed in the code.

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

...
   </Container>
</UI>
<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[
%put Text entered: '$text';
]]>
</CodeTemplate>
```

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 generated SAScode:

```
%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.

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

Now if you enter

```
Hello
World
```

in the text box, here is the generated SAS code:

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

timepicker

The Velocity variable for the `timepicker` input type holds the information for the time control. By default, this variable is an empty string. After the user has selected a time or if there is a default value for the timepicker, the variable holds the control's current time. The input and return value of the times are in the ISO format HH:mm:ss.

This example generates SAS code to read the time value into a macro variable, mytime. The value is then written to the log in the raw date form and the formatted TIMEAMPM. value.

```
<Options>
    <Option name="oTime24Hours" inputType="timepicker" required="true"
   defaultValue="23:15:22" use24HourTime="true"
   helpMessageRef="timePickerLabel">
        Select the time:
    </Option>
</Options>
<CodeTemplate>
   <![CDATA[
#if( $oTime24Hours )
%let mytime=%SYSFUNC(InputN(%QUOTE($oTime24Hours), %QUOTE(ANYDTTME.)));

data _null_;
rawtime=&mytime;
```

```
frmtime = PUTN(rawtime, 'timeampm.');
put rawtime= frmtime= ;

run;
#end
]]>
</CodeTemplate>
```

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

```
<Options>
   <Option name="nullRho" indent="1" inputType="validationtext"
      defaultValue="0" required="true"
      promptMessage="Enter a number greater than -1 and less than 1
         for the null hypothesis correlation"
      invalidMessage="Enter a number greater than -1 and less than 1
         for the null hypothesis correlation"
      missingMessage="Enter a number grearter than -1 and less than 1
         for the null hypothesis correlation"
      regExp="[-+]?((0\.\d*)|(\.\d+)|0)">Null hypothesis correlation:</
Option>
 </Options>
<CodeTemplate>
   <![CDATA[
rh0=$nullRho]]>
</CodeTemplate>
```

weekpicker

The Velocity variable for the `weekpicker` input type holds the information for the weekpicker control. By default, this variable is an empty string. When a week is selected, the variable value is in the ISO format yyyy-Wwww. To convert this value to a SAS date value, use the WEEKV. informat.

This example generates SAS code to read the week value into a macro variable, mydate. The value is then written to the log in the raw date form and the formatted MONNAME. value.

```
<Options>
   <Option name="oWeek" inputType="weekpicker" defaultValue="2018-W25"
      minValue="2018-W15" maxValue="2020-W25">
      Choose a Week:
   </Option>

</Options>
<CodeTemplate>
```

```
   <![CDATA[
#if( $oWeek)
%let mydate=%SYSFUNC(InputN(%QUOTE($oWeek), %QUOTE(weekv.)));

data _null_;
rawdate=&mydate;

frmdate = PUTN(rawdate, 'monname.');
put rawdate= frmdate= ;

run;
#end
#end
]]>
</CodeTemplate>
```

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

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

get Method

Short Description	Provides information about the outputdata control. This method takes a string parameter that accepts one of these values: ■ `table` – the name of the table that is used for the data source ■ `value` – the name of the data source in *Library.Table* format.
Return Value	Returns the attributes of the outputdata control. **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	`<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	`<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	`<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 Developers

$CTMMathUtil Variable 147
getMin Method 147
getMax Method 148
getSum Method 148
$CTMUtil Variable 149
doubleQuoteString Method 149
quoteString Method 149
isProductLicensed Method 150
toSASName Method 150

$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

Short Description	Returns the largest value of an array of doubles.
Syntax	`Double getMax(ArrayList<Double> inputArray)`
Parameter	input an array of double values.
Return Value	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.

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

getSum Method

Short Description	Returns the smallest value of an array of doubles.
Syntax	`Double getSum(ArrayList<Double> inputArray)`
Parameter	input an array of double values.
Return Value	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.
Example	`#set($array = [1.0, 2.0, 3.0])` `$CTMMathUtil.getSum($array)`

```
/* double returned: 6.0 */
```

$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( $input="sysvlong")`
	`$CTMUtil.doubleQuoteString($input);`
	
	`/* string returned: "&sysvlong" */`

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

For this method to work as expected, the `fetchProductLicenses` attribute must be set to `true` for the `Task` element.

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`
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, 9.4)` `/* Boolean returned - true if Base SAS 9.4 or earlier is licensed; false if a version greater than SAS 9.4 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 Variables'n */`