SAS® Studio 3.5:

Writing Your First Custom Task

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2016. *SAS® Studio 3.5: Writing Your First Custom Task*. Cary, NC: SAS Institute Inc.

SAS® Studio 3.5: Writing Your First Custom Task

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

April 2016

Contents

Using This Book

Audience

SAS Studio: Writing Your First Custom Task is intended for developers who need to create custom tasks. The purpose of this book is to show you how to convert an existing SAS program to a SAS Studio task. The benefit of using a SAS Studio task is that you can share this point-and-click interface with other SAS Studio users. Other users do not need to understand SAS programming to run the task. They simply enter the parameter values and run the task to get their output.

Prerequisites

Requirements

To complete the examples in this document, you must have access to SAS Studio 3.4 or later. This document assumes you are running SAS Studio 3.5.

Create the Example Data Set

To follow the steps in this document, you should create a SAS data set called Character.

To create this data set:

1 Open SAS Studio. In the banner, click  and select **New SAS Program**.

An empty **Program** tab opens in the SAS Studio workspace.

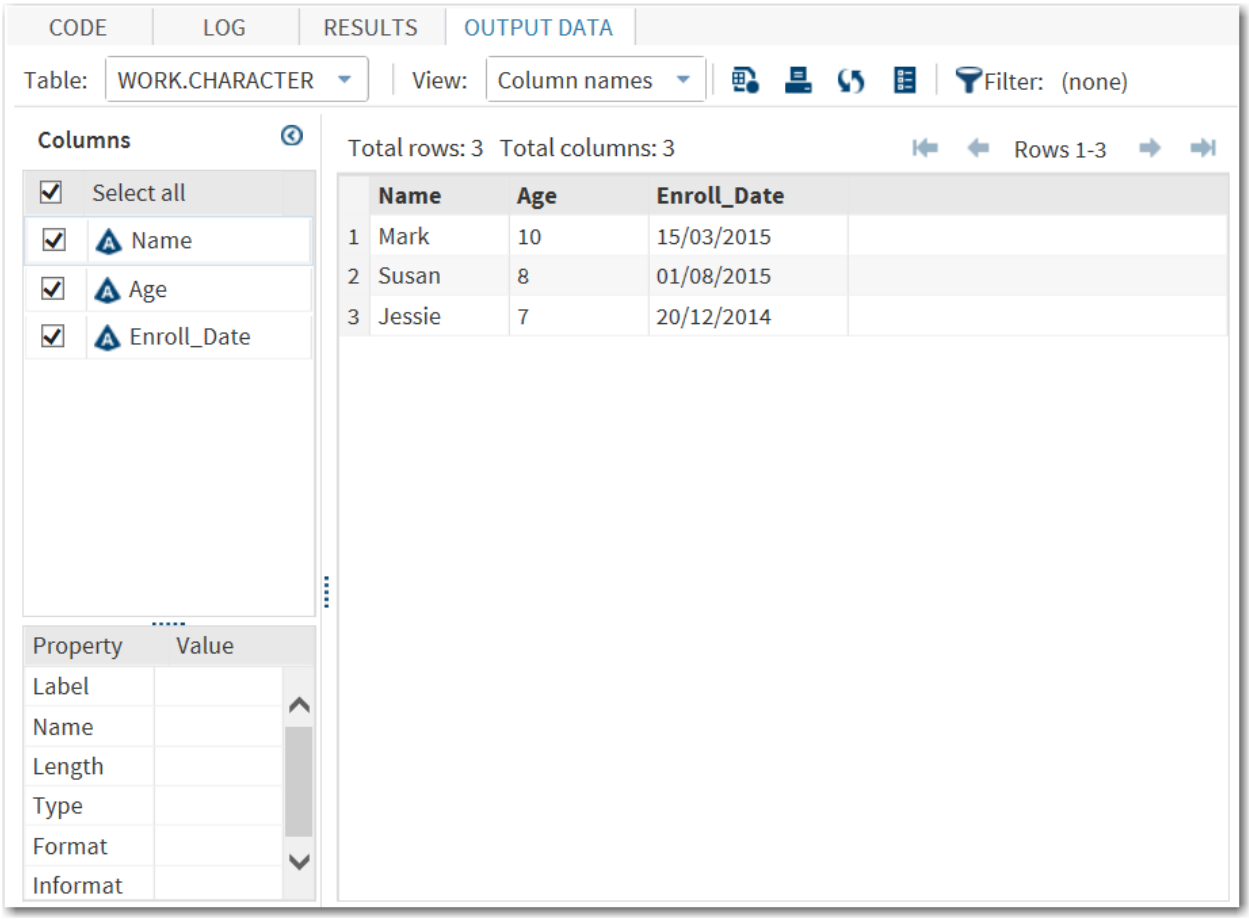
2 Enter this code on the **Program** tab:

```
data work.character;
   input Name $ Age $ Enroll_Date $10.;
datalines;
Mark 10 15/03/2015
Susan 8 01/08/2015
Jessie 7 20/12/2014
run;
```

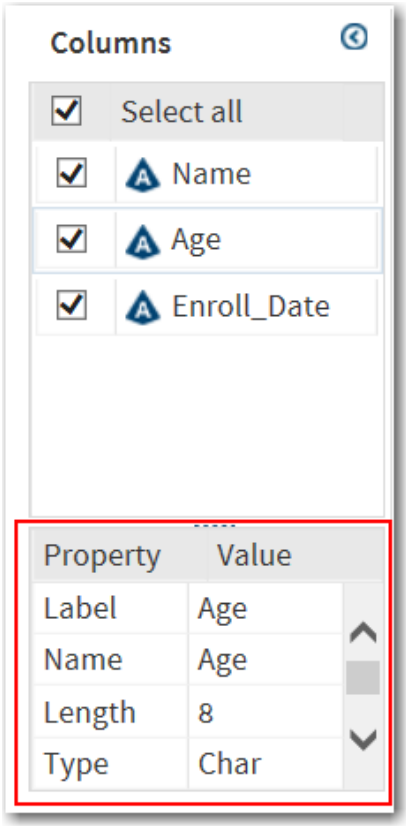
Note: This data set is saved in the Work library, which exists only for the SAS Studio session. If you start a new SAS Studio session, you need to re-create this data set.

3 Click  .

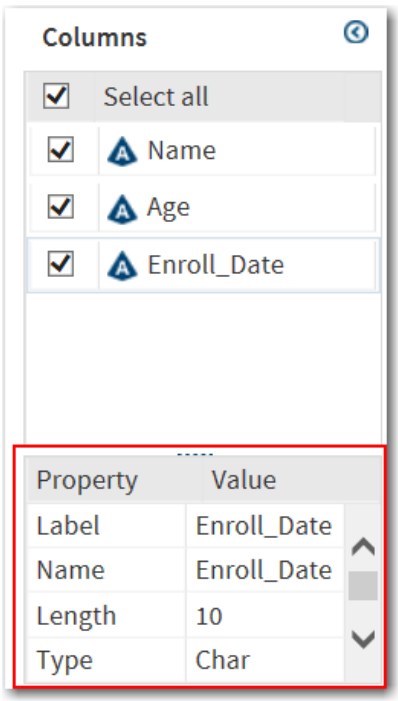
The Work.Character data set opens on the **OUTPUT DATA** tab.



To view the properties for an individual variable, select the variable name from the **Columns** pane. If you select the **Age** variable, the properties pane shows that Age is a character variable.



If you select the **Enroll_Date** variable, the properties pane shows that Enroll_Date is a character variable with a length of 10.



When you are finished, close the **Program** tab.

1

Introduction to SAS Studio Tasks

Overview of SAS Studio Tasks

SAS Studio is shipped with several predefined tasks, which are point-and-click user interfaces. These user interfaces 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. For more information about these predefined tasks, see *SAS Studio: User's Guide*.

Because of the flexibility of the task framework, you can take your existing SAS code and turn it into a SAS Studio task. In SAS Studio, all tasks use the same common task model, which is based on XML and the Velocity Template Language. No Java programming or ActionScript programming is required to build a task.

The common task model (CTM) defines the template for the task. In the CTM file, you define the user interface for the task and specify the code that is needed to run the task. In addition, the task has metadata so that it is recognized by SAS Studio.

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

Registration
: The `Registration` element identifies the type of task. In this element, you define the task name, description, and other task properties.

Metadata
: The `Metadata` element can specify whether an input data source is required to run the task. In the metadata, you also specify any role assignments and the options in the task.

  - The `Roles` element specifies the types of variables that are required by the task. Here is the information that you would specify in this element:
    - type of variable that the user can assign to this role (for example, numeric or character)

    - the minimum or maximum number of variables that you can assign to a role
    - the label or description of the role that appears in the user interface
  - The `Options` element specifies how to display the options in the user interface.

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

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

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

Code Template
: The `CodeTemplate` element determines the final output of the task. For most tasks, the output is SAS code.

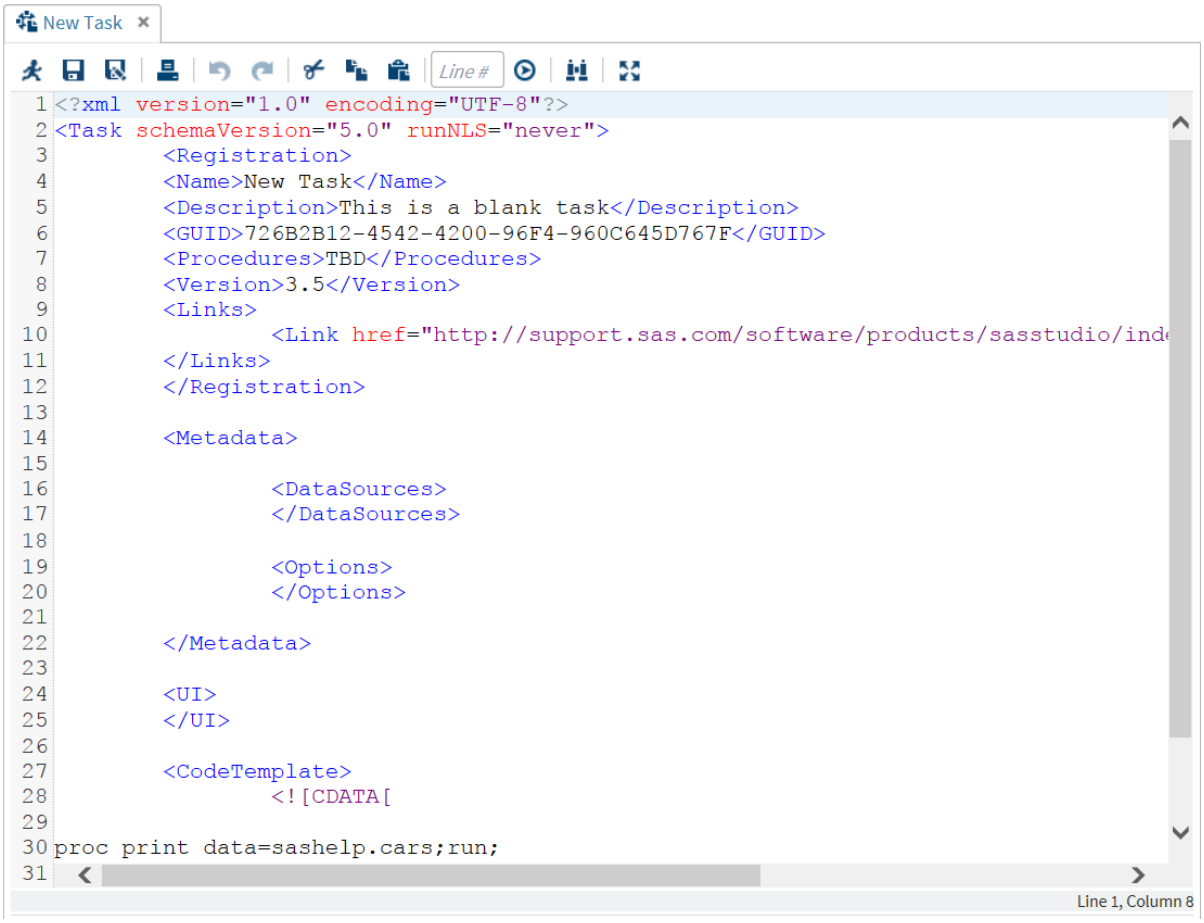
General Steps for Creating a New Task

SAS Studio is shipped with a blank task template that you can use to create your own tasks.

Here is a high-level overview of how to create a new task in SAS Studio.

1. Open SAS Studio.
2. In the navigation pane, open the **Tasks** section.
3. Click  and select **New Task**.

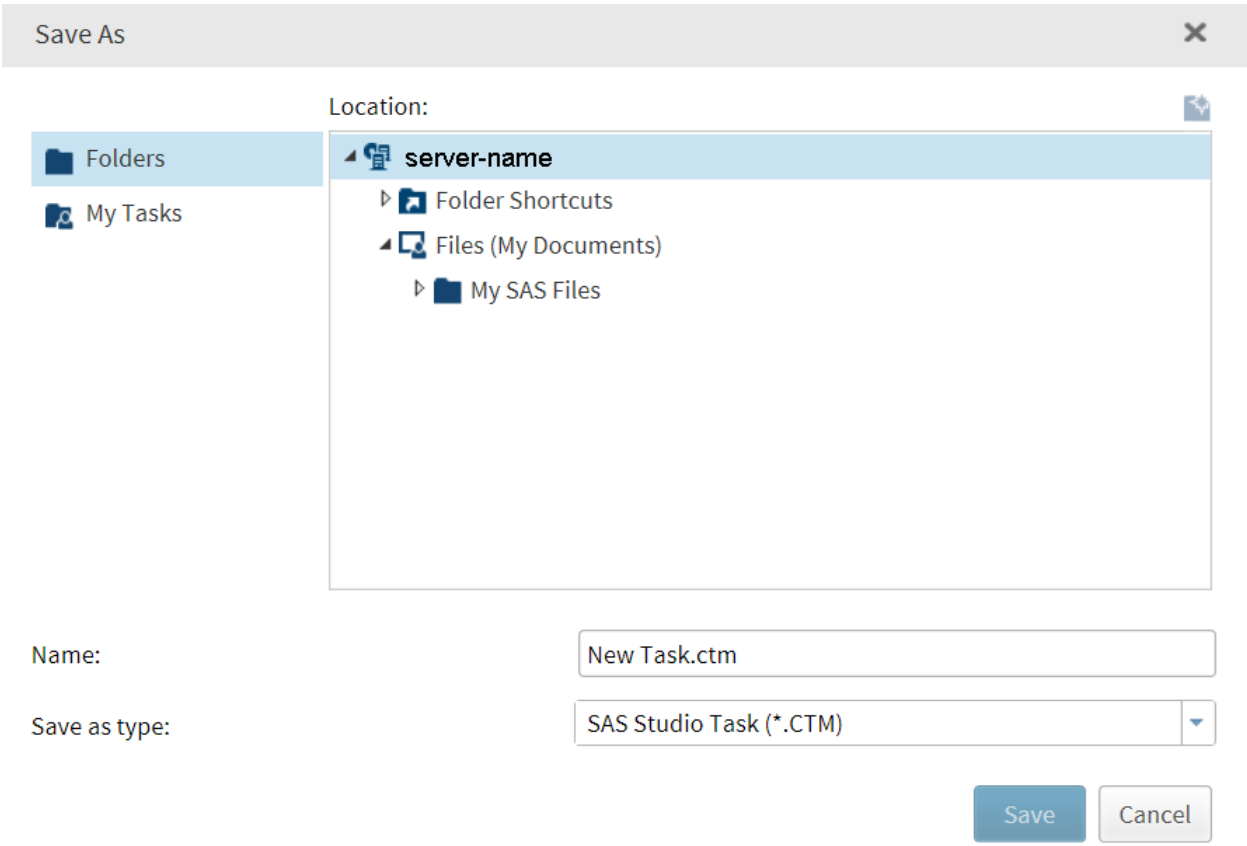
The new task appears in SAS Studio.



4 Revise the code to create your custom task. For more information about this code and the common task model, see *SAS Studio: Developer's Guide to Writing Custom Tasks*.

5 To save the task, click  .

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



About Running a Task

After you write the CTM code for a task, you must run the CTM in SAS Studio to generate the user interface. To run your CTM, click . (Alternatively, you can press F3.) A new tab that contains the user interface for the task appears in your work area. To view the SAS code for this task, click **Code**. The CTM code is still available from the original tab within the task.

2

Creating a Basic Custom Task

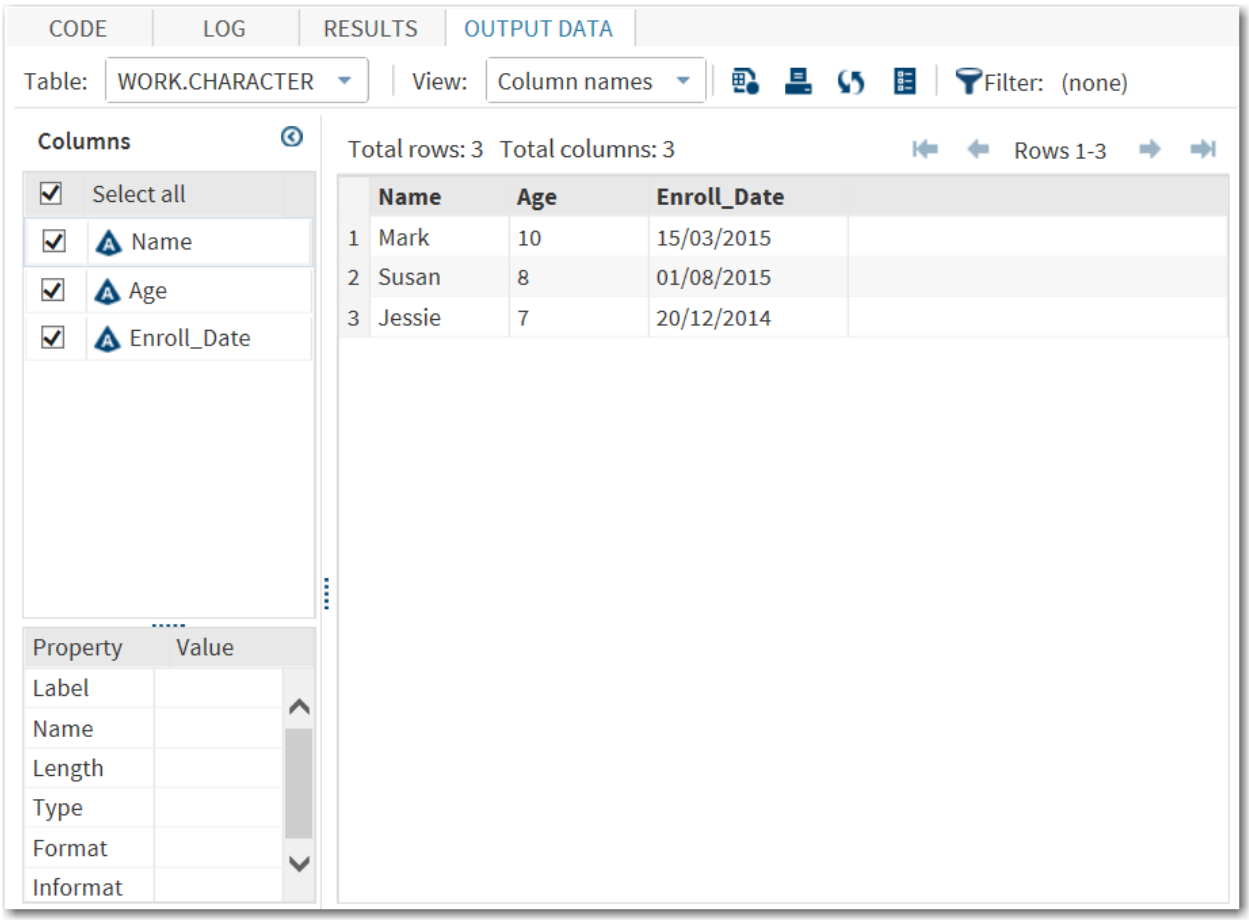
About the SAS Program for This Example

Before performing an analysis, SAS programmers often must clean up the data. Often this cleanup includes converting character variables to numeric variables. For example, the data came from a vendor who used nonstandard character date values, and you need these values to be converted to SAS dates. Another example is when a colleague has a data set with a character variable that contains only digits, and you want to run a numeric analysis on that variable.

It is easy to write a SAS program to convert character variables to numeric variables. However, you want to give your colleague (who is not a SAS programmer) the ability to perform this conversion. You can create a SAS Studio task that enables anyone at your site to change a character variable to a numeric variable in any data set.

This document uses the Work.Character data set to show you how to create this SAS Studio task. Before you can continue, you must create this data set in SAS Studio. (For more information, see “Create the Example Data Set” on page v.)

In the Work.Character data set, both Age and Enroll_Date are listed as character variables. However, you want Age to be a numeric variable and Enroll_Date to be a SAS date. A SAS date is a numeric variable that represents the number of days since January 1, 1960. SAS provides a variety of date formats that you can use to display these date values.



Here is a SAS program that you could use to convert the Enroll_Date variable:

```
data work.character_out;
   set work.character;
   new_enroll_date=input(enroll_date,ddmmyy10.);
run;
```

Note: The informat must match the current structure of the character variable. In this case, the character value 15/03/2015 corresponds to DDMMYY10.

Because you have had multiple requests for this type of conversion, you added macro variables to the SAS program. By using macro variables, you do not have

to rewrite the program code for each user who needs to perform this conversion. Instead, the user can simply specify new values for the macro variables.

Here is the same SAS program using macro variables:

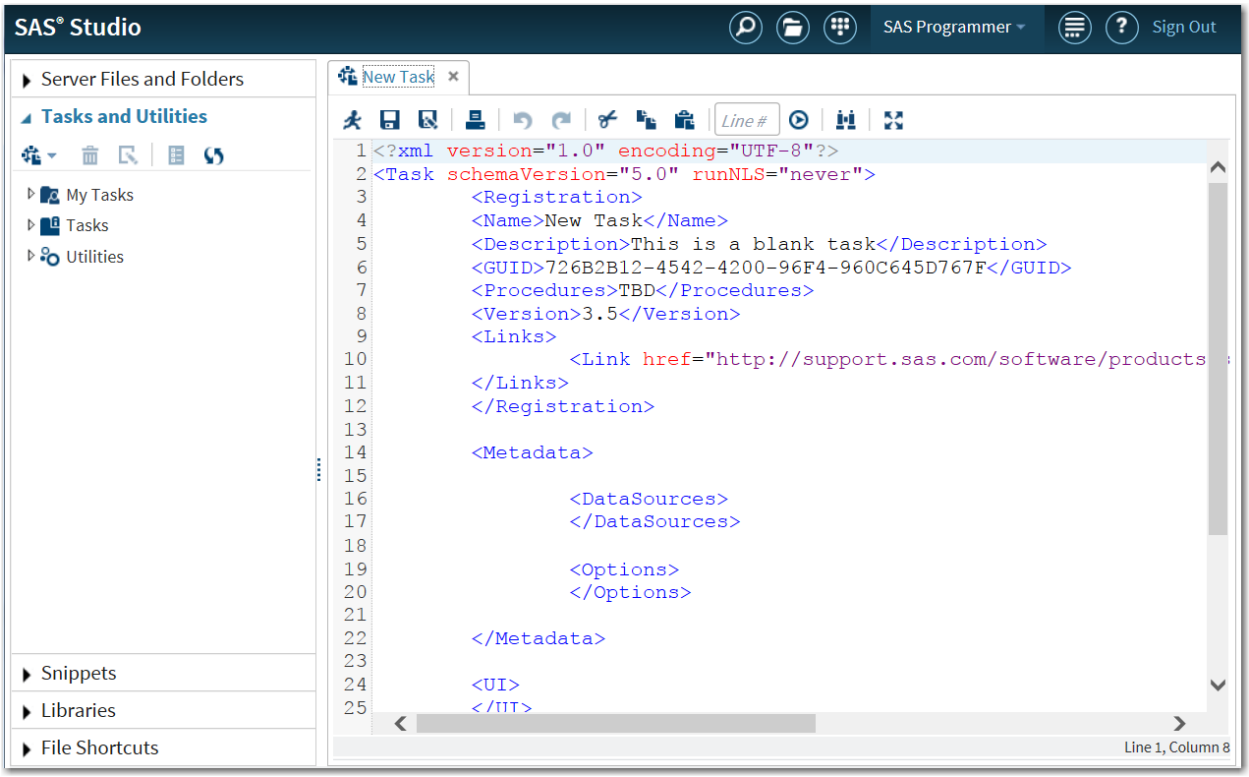
```
%let inlibname = work;
%let indsname = character;
%let invarname = date;
%let outdsname = character_out;
%let informat = ddmmyy10;

data &outlibname.&outdsname;
set &inlibname..&indsname;
&invarname._new = input(&invarname,&informat..);
run;
```

Because changing the values of the macro variables can be time-consuming, the remaining steps in this document show you how to convert this program to a SAS Studio task. The point-and-click interface enables users to quickly and efficiently customize their variables.

Step 1: Open a Blank Task

Open SAS Studio. In the navigation pane, click **Tasks**. To open a blank task, click  and select **New Task**.

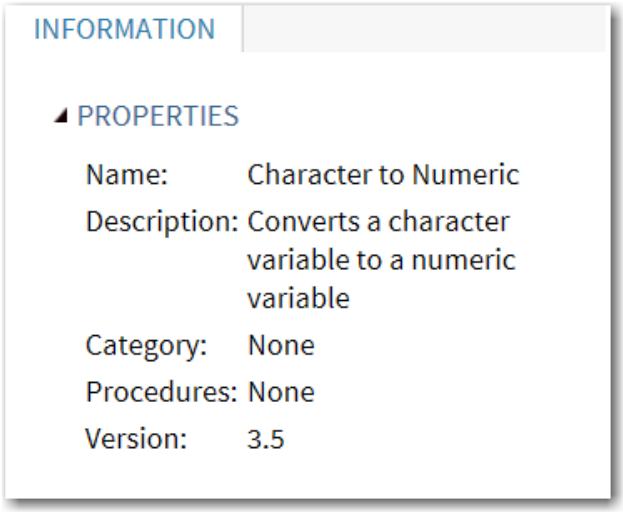
The template for a blank task opens in the SAS Studio workspace.



Step 2: Register the Task

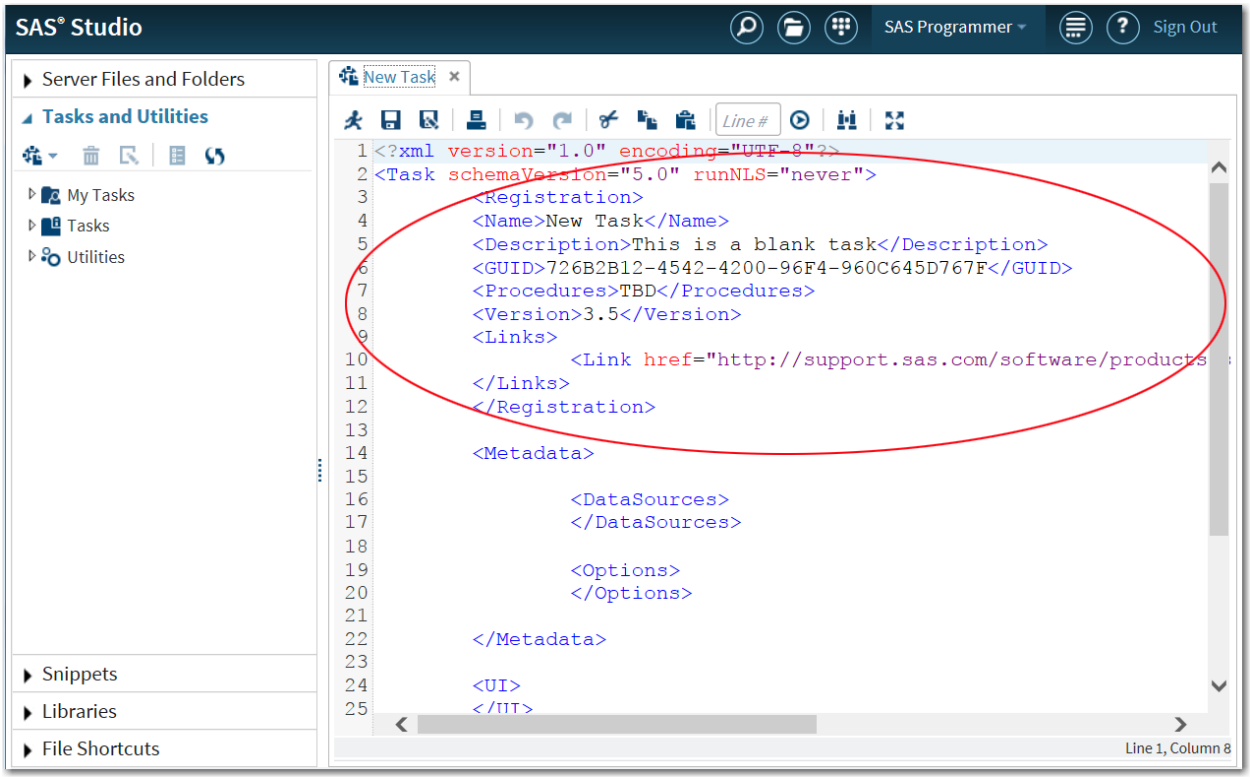
About Registering a Task

In order to run a task in SAS Studio, you must register the task. The registration information is displayed on the **INFORMATION** tab in the user interface. When you complete this step, you will have registered the Convert Character to Numeric task and created the **INFORMATION** tab, as shown in this screen shot.



Edit the Registration Element

In the blank task, here is the `Registration` element:



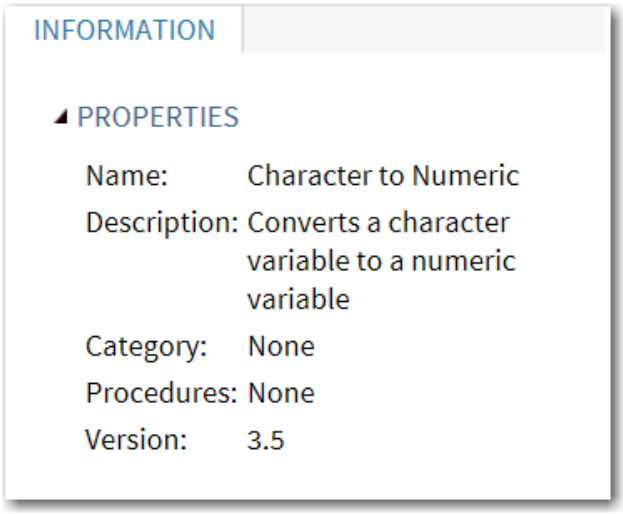
Revise the code in the `Registration` element to match the following code:

```
<Registration>
   <Name>Convert Character to Numeric</Name>
   <Description>Converts a character variable to a numeric variable</Description>
   <GUID>TBD</GUID>
   <Procedures>None</Procedures>
   <Version>3.5</Version>
</Registration>
```

Note: This code removes the `Link` element because it is not needed for this example. The `GUID` and `Procedures` elements are required for the CTM code to run.

View the Information Tab

Click  to generate the user interface for the task. The resulting user interface includes an **INFORMATION** tab that lists the properties that you specified in the `Registration` element.



Close the task and return to the tab that contains the CTM code.

To save your CTM code:

1 Click  . The Save As dialog box appears.

2 In the selection pane, select **My Tasks**.

3 Specify a name for the CTM file and click **Save**.

Step 3: Specify the Input Data Source and Identify Any Roles

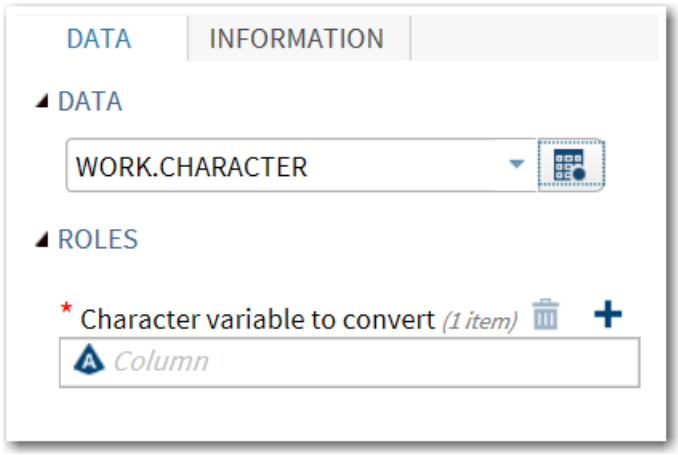
About Input Data Sources and Roles

In the common task model (CTM), you identify the input data source by using the `DataSource` element. You use the `Roles` element to identify how a variable from the input data source is used in the task.

To add any options to a task, you must first define the option in the metadata. Then you must specify how to display the option in the user interface.

When you complete this step, the user interface includes a **DATA** tab with two groups (**DATA** and **ROLES**). Two options (one for selecting the data source and

one for assigning a character variable to the **Character variable to convert** role) are also available.



Define the Input Data Source in the Metadata Element

In the blank task, find the `Metadata` element:

```
<Metadata>
   <DataSources>
   </DataSources>

   <Options>
   </Options>

</Metadata>
```

Now, add the highlighted code to the `DataSources` element:

```
<DataSources>
   <DataSource name="inlibname">
   </DataSource>
</DataSources>
```

In the `DataSource` element, you are specifying a name for the input data source. You refer to this name later in the task definition. In this example, the name of the data source is `inlibname`.

Define the Character variable to convert Role in the Metadata

Now that you can select the input data source, you need to identify the variable that contains the values that you want to convert. You specify this variable by using the `Roles` element, which is a child of the `DataSource` element.

Add the highlighted code to the `DataSource` element:

```
<DataSources>
   <DataSource name="inlibname">
   <Roles>
      <Role name="invarname" type="C" minVars="1" maxVars="1">
```

```
        Character variable to convert</Role>
    </Roles>
    </DataSource>
</DataSources>
```

Here is an explanation of this code:

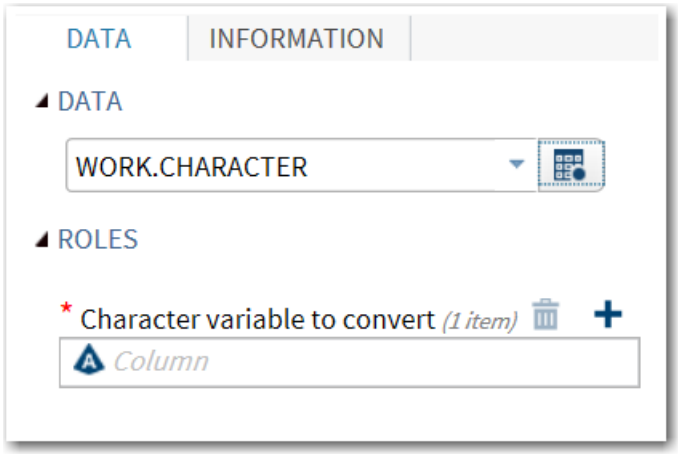
```
<DataSources>
  <DataSource name="inlibname">
  <Roles>
  1 <Role name="invarname"
  2    type="C"
  3    minVars="1"
  4    maxVars="1">
  5    Character variable to convert</Role>
    </Roles>

  </DataSource>
</DataSources>
```

1. The `name` attribute specifies the name for the role. In this example, the name of the role is `invarname`.
2. The `type` attribute specifies the type of variable that can be assigned to this role. In this example, only character variables (represented by `type="C"`) can be assigned to this role. In the user interface for the task, character columns are identified by the  .
3. The `minVars` attribute specifies the minimum number of variables that must be assigned to this role. In this example, a character variable must be assigned to the **Character variable to convert** role in order for the task to run, so `minVars="1"`. In the user interface for the task, a red asterisk appears next to the name of this field to indicate that a variable is required. If no variable is assigned to this role, the task cannot run.
4. The `maxVars` attribute specifies the maximum number of variables that can be assigned to this role. In this example, only one variable can be assigned to this role.
5. `Character variable to convert` is the label for this field in the user interface.

Add the Input Data Source and Roles Fields to the User Interface

Now that the data source and role are defined in the metadata, you need to create the fields for the user interface.

Here is the user interface that you want to create:



In the previous section, you defined the metadata for the data source field and the **Character variable to convert** field. However, the final user interface also shows a **DATA** tab, a **DATA** heading, and a **ROLES** heading. To include these elements in the user interface, you must define these elements in the metadata by using multiple `Option` elements.

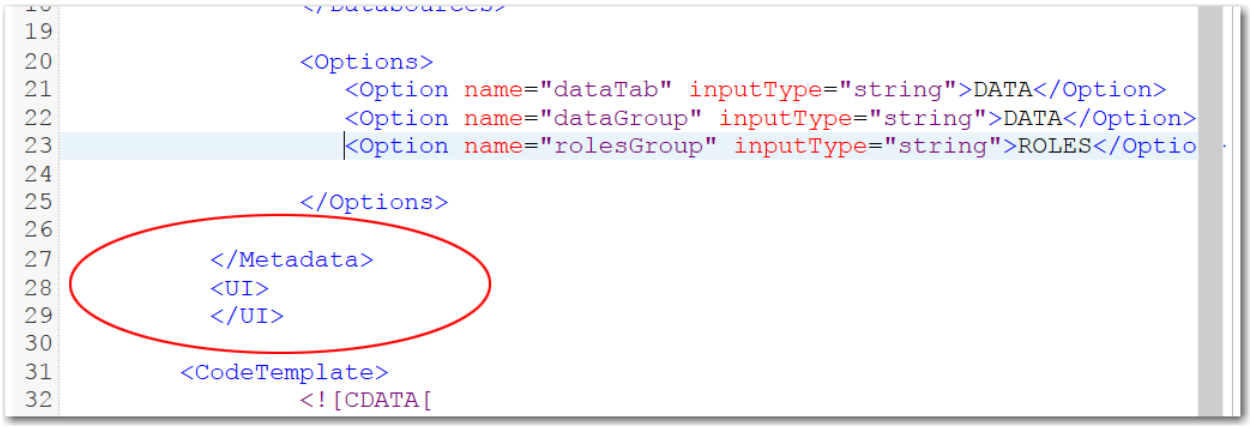
To define these three UI elements in the metadata, add the highlighted code to the `Options` element.

```
    </DataSources>
    ...
    <Options>
       <Option name="dataTab" inputType="string">DATA</Option>
       <Option name="dataGroup" inputType="string">DATA</Option>
       <Option name="rolesGroup" inputType="string">ROLES</Option>
    </Options>

</Metadata>
```

In the `Option` elements, you specified the following information:

- the name for each UI element. For example, the name for the **DATA** tab is `dataTab`. You use this name again in the `UI` element.
- the input type for these options. Because these are labels that appear in the user interface, the input type is a string.

Now that all of the elements that you need (the **DATA** tab, the group headings, the data source option, and the roles option) are defined in the metadata, you can specify how you want these items to appear in the user interface. In your example, here is the location of the `UI` element.



As you can see, the `UI` element follows the closing `Metadata` element. In the `UI` element, add the following code:

```
<UI>
   <Container option="dataTab">
      <Group option="dataGroup" open="true">
         <DataItem data="inlibname" />
      </Group>
      <Group option="rolesGroup" open="true">
         <RoleItem role="invarname" />
      </Group>
   </Container>
</UI>
```

Here is an explanation of this code:

```
<UI>
 ① <Container option="dataTab">
   ② <Group option="dataGroup" open="true">
        <DataItem data="inlibname" />
     </Group>
   ③ <Group option="rolesGroup" open="true">
        <RoleItem role="invarname" />
     </Group>
   </Container>
</UI>
```

1. `Container` elements enable you to add tabs to your user interface. To create multiple tabs, you need multiple `Container` elements.

   In this code example, you are creating only the **DATA** tab. You specified the name (`dataTab`) and UI label (**DATA**) for this tab in the metadata.

```
<Option name="dataTab" inputType="string">DATA</Option>
```

   Because only one container is defined for this task, all the remaining UI elements appear on the **DATA** tab.

2. In the first `Group` element, `option="dataGroup"` refers to the `dataGroup` name that you defined in the metadata.

```
<Option name="dataGroup" inputType="string">DATA</Option>
```

   In the user interface, the label for this group is **DATA**.

The `DataItem` element creates the field for the input data source. In this example, the name of the input data source is `inlibname`, which was defined in the `DataSource` element in the metadata.

3 In the second `Group` element, `option="rolesGroup"` refers to the `rolesGroup` name that you defined in this code in the metadata.

```
<Option name="rolesGroup" inputType="string">ROLES</Option>
```

In the user interface, the label for this group is **ROLES**.

The `RoleItem` element creates a field for selecting the character variable that you want to convert. You defined this role in the `Roles` element in the metadata.

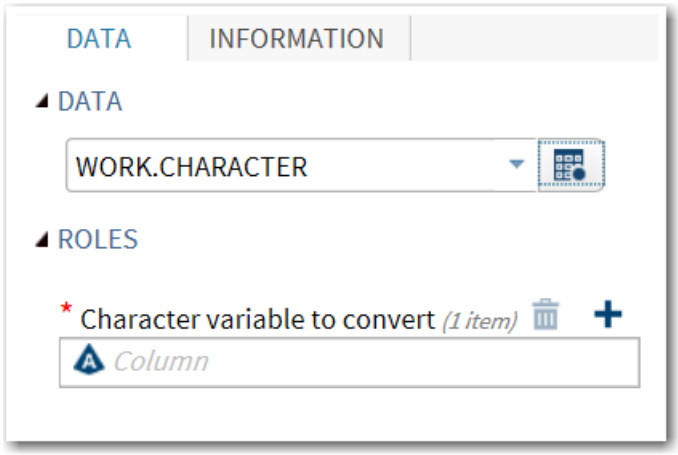
```
<DataSource name="inlibname">
    <Roles>
      <Role name="invarname" type="C" minVars="1" maxVars="1">
         Character variable to convert</Role>
    </Roles>
  </DataSource>
```

The name of this role is `invarname`. When displayed in the user interface, the role is labeled **Character variable to convert**.

View the Data Source and Character variable to convert Options

Click  to generate the user interface for the task.

The user interface includes a **DATA** tab with two groups (**DATA** and **ROLES**). Two options (one for selecting the data source and one for assigning a character variable to the **Character variable to convert** role) also appear.



To create this user interface, you had to complete these steps:

1 Define all elements in the metadata.

- You defined the data source option in “Define the Input Data Source in the Metadata Element” on page 11.

- You defined the **Character variable to convert** role in “Define the Character variable to convert Role in the Metadata” on page 11.
- You defined the **DATA** tab, **DATA** group heading, and **ROLES** group heading at the beginning of “Add the Input Data Source and Roles Fields to the User Interface” on page 12.

2 Use the `UI` element to specify how these items appeared in the user interface. You defined the contents of the `UI` element in “Add the Input Data Source and Roles Fields to the User Interface” on page 12.

Close the task and return to the tab that contains the CTM code. Click  to save your CTM code.

Step 4: Create the Remaining Options

About the Remaining Options

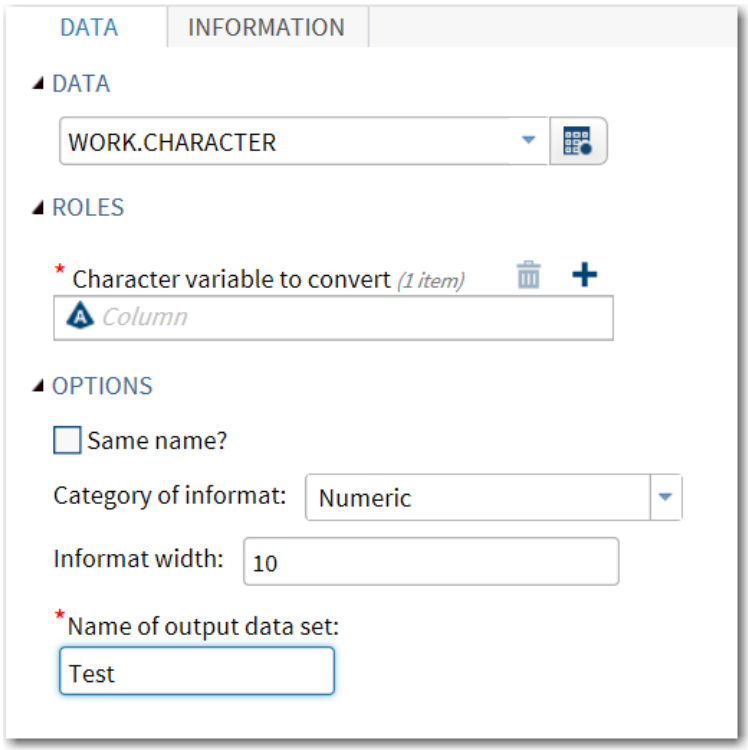
Now that you have defined the data source and the role, you need to create the remaining options for the task. These options represent the macro variables in the original SAS program.

To complete this example, you need to create options that enable you to specify this information:

- whether the resulting numeric variable has the same name as the original character variable
- the informat to use to read the existing variable
- the width of the informat
- the name of the output data source

As with all options, you first must define these options in the `Metadata` element. Then you must specify how the options should appear in the user interface by using the `UI` element.

When you complete this step, the user interface includes a new **OPTIONS** heading and four new options.



Define These Options in the Metadata

To define these new options, add the highlighted code to the `Options` element:

```
<Options>
   <Option name="dataTab" inputType="string">DATA</Option>
   <Option name="dataGroup" inputType="string">DATA</Option>
   <Option name="rolesGroup" inputType="string">ROLES</Option>
   <Option name="optionsGroup" inputType="string">OPTIONS</Option>

   <Option name="samename" inputType="checkbox">Same name?</Option>
   <Option name="informatType" inputType="combobox">Category
      of informat:</Option>
   <Option name="numericChoice" inputType="string" returnValue="BEST">
      Numeric</Option>
   <Option name="dateChoice" inputType="string" returnValue="DDMMYY">
      Date</Option>
   <Option name="informatWidth" inputType="numbertext" minValue="1"
      defaultValue="10">Informat width:</Option>
   <Option name="outputDSName" inputType="outputdata" defaultValue="Test"
      required="true">Name of output data set:</Option>
</Options>
```

Here is an explanation of this code:

```
<Options>
   <Option name="dataTab" inputType="string">DATA</Option>
   <Option name="dataGroup" inputType="string">DATA</Option>
   <Option name="rolesGroup" inputType="string">ROLES</Option>
 ①<Option name="optionsGroup" inputType="string">OPTIONS</Option>

 ②<Option name="samename" inputType="checkbox">Same name?</Option>
 ③<Option name="informatType" inputType="combobox">Category of informat:</Option>
   <Option name="numericChoice" inputType="string" returnValue="BEST">Numeric</Option>
   <Option name="dateChoice" inputType="string" returnValue="DDMMYY">Date</Option>
 ④<Option name="informatWidth" inputType="numbertext" minValue="1"
     defaultValue="10">Informat width:</Option>
 ⑤<Option name="outputDSName" inputType="outputdata" defaultValue="Test"
     required="true">Name of output data set:</Option>
</Options>
```

1. This line defines the **OPTIONS** group in the metadata. The four new options appear under the **OPTIONS** heading on the **DATA** tab.
2. This line defines the `samename` option, which enables you to specify whether the resulting numeric variable has the same name as the original character variable.

Because `inputType=checkbox`, this option appears as a check box labeled **Same name?** in the user interface. (For more information about the available input types, see *SAS Studio: Developer's Guide to Writing Custom Tasks*.)

- If you select the **Same name?** check box, the existing character variable is dropped from the output data set. A new numeric variable with the same name as the original character variable is created.
- If you do not select the **Same name?** check box, a new numeric variable is added to the output data set. The character variable remains and also appears in the output data set.

3. This line defines the `informatType` option, which enables you to specify the category of the informat. In this example, the categories are numeric and date.

Because `inputType=combobox`, this option appears as a combobox control labeled **Category of informat**. From the combobox control, the user can select **Numeric** or **Date**.

- The **Numeric** option is defined in this line of code:

  ```
  <Option name="numericChoice" inputType="string" returnValue="BEST">
     Numeric</Option>
  ```

  The `returnValue` attribute is set to `BEST`, which means that the combobox control returns the string “BEST”.

- The **Date** option is defined in this line of code:

  ```
  <Option name="dateChoice" inputType="string" returnValue="DDMMYY">
     Date</Option>
  ```

  The `returnValue` attribute is set to `DDMMYY`, which means that the combobox control returns the string “ DDMMYYYY”.

4. This line defines the `informatWidth` option, which specifies the width for the informat. Because `inputType=numbertext`, this option appears as a numbertext control labeled **Informat width**. Because `defaultvalue=10`, the default value of the informat width is 10.

5 This line defines the `outputDSName` option, which enables you to specify a name for the output data set. In the user interface, this option appears as an outputdata control, which is simply a text box where the user can specify the name of the output data set. The `defaultValue` attribute is set to `Test`. When you run the task, the output data set appears on the **OUTPUT DATA** tab. The default name that appears in the user interface is Test.

Add These Options to the User Interface

In the `UI` element, add the highlighted code to create the `optionsGroup` and its options:

```
<Container option="dataTab">
   <Group option="dataGroup" open="true">
      <DataItem data="inlibname" />
   </Group>

   <Group option="rolesGroup" open="true">
      <RoleItem role="invarname" />
   </Group>

   <Group option="optionsGroup" open="true">
      <OptionItem option="samename" />
      <OptionChoice option="informatType">
         <OptionItem option="numericChoice" />
         <OptionItem option="dateChoice" />
      </OptionChoice>
      <OptionItem option="informatWidth" />
      <OptionItem option="outputDSName" />
   </Group>
</Container>
```

Here is an explanation of this code:

```
<Container option="dataTab">
   <Group option="dataGroup" open="true">
      <DataItem data="inlibname" />
   </Group>
   <Group option="rolesGroup" open="true">
      <RoleItem role="invarname" />
   </Group>

(1) <Group option="optionsGroup" open="true">
      <OptionItem option="samename" />
   (2)<OptionChoice option="informatType">
         <OptionItem option="numericChoice" />
         <OptionItem option="dateChoice" />
      </OptionChoice>
   (3)<OptionItem option="informatWidth" />
   (4)<OptionItem option="outputDSName" />
   </Group>
</Container>
```

1 This line creates the **OPTIONS** group on the **DATA** tab. Because `open="true"`, this group is expanded by default, so you can see all of the

options in this group. You can use the  arrow in the interface to collapse this group.

The options in the **OPTIONS** group appear in the order in which you specified them in the `UI` element. In this example, the options are in this order: sameName, informatType, informatWidth, and outputDSName.

2 In the `Options` element in the metadata, you specified this code:

```
<Option name="informatType" inputType="combobox">Category of informat:</Option>
```

Because `inputType="combobox"`, the informatType option is a combobox control. You also defined the two options (**Numeric** and **Date**) that are available from the combobox control.

```
<Option name="numericChoice" inputType="string" returnValue="BEST">Numeric</Option>
 <Option name="dateChoice" inputType="string" returnValue="DDMMYY">Date</Option>
```

In the `UI` element, you specify the order in which these two options (**Numeric** and **Date**) appear in the combobox control.

3 This line creates the **Informat width** text box. In the metadata, you defined this option by using this code:

```
 <Option name="informatWidth" inputType="numbertext" minValue="1"
      defaultValue="10">Informat width:</Option>
```

Because `defaultValue="10"`, a default value of 10 appears in the **Informat width** text box when you generate the user interface.

4 This line creates the **Name of output data set** box. In the metadata, you defined this option by using this code:

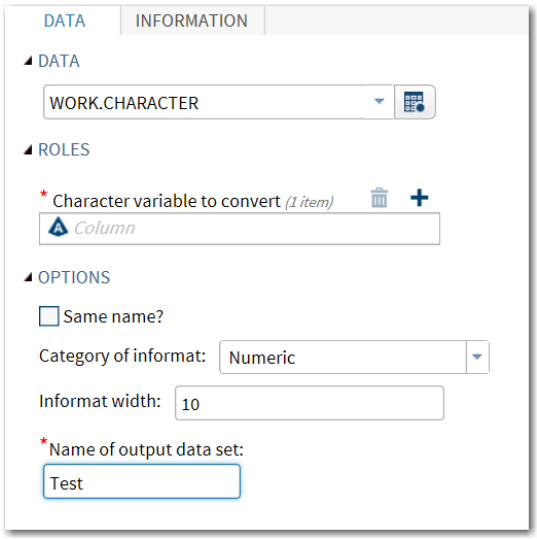
```
<Option name="outputDSName" inputType="outputdata" defaultValue="Test"
      required="true">Name of output data set:</Option>
```

Because `defaultValue="Test"`, a default name of Test appears in the **Name of output data set** box when you generate the user interface.

View the New Options Heading

Click  to generate the user interface for the task.

Now, the **OPTIONS** heading and its contents are visible in the user interface.



Close the task and return to the tab that contains the CTM code. Click  to save your CTM code.

Step 5: Add the Apache Velocity Code

About the Apache Velocity Code

The `CodeTemplate` element contains the Apache Velocity code, which is used to generate the SAS code for the task. Apache Velocity code is a template engine based on Java. In the code, keywords that are identified with a dollar sign ($) are Velocity variables. Keywords with a number sign (#) are directives. For more information about this code, see *Apache Velocity User's Guide*.

Until this point, you have focused on creating the user interface for the task. In your example task, the following code is in the `CodeTemplate` element by default:

```
<CodeTemplate>
<![CDATA[

   proc print=sashelp.cars;
   run;
]]>
</CodeTemplate>
```

If you ran the task (in other words, selected a variable to convert and clicked ), you might have noticed that the contents of the Sashelp.Cars data set appeared on the **RESULTS** tab. The task ran successfully because the content in the `CodeTemplate` element is valid SAS code.

In this step, you replace the default Velocity code with the Velocity code for the Convert Character to Numeric task.

Add the Velocity Code for the Convert Character to Numeric Task

In the task definition, find the `CodeTemplate` element and remove the highlighted code:

```
<CodeTemplate>
<![CDATA[

   proc print=sashelp.cars;
   run;
]]>
</CodeTemplate>
```

Now, add the highlighted code to the `CodeTemplate` element:

```
   <CodeTemplate>
      <![CDATA[

      #set($charvar=$invarname.get(0))

      #set($informat = "${informatType}${informatWidth}.")

     data $outputDSName;
     #if ($samename=='1')
         #set($newCol = "${charvar}_old")
         set $inlibname(rename=$charvar=$newCol);
         $charvar = input($newCol,$informat);
         drop $newCol;
     #else
         #set($newCol = "${charvar}_new")
         set $inlibname;
         $newCol = input($charvar,$informat);
      #end

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

Here is an explanation of this code:

```
<CodeTemplate>
        <![CDATA[

              ① #set($charvar=$invarname.get(0))

              ② #set($informat = "${informatType}${informatWidth}.")

              ③ data $outputDSName;
              ④ #if ($samename=='1')
                    #set($newCol = "${charvar}_old")
                    set $inlibname(rename=$charvar=$newCol);
                    $charvar = input($newCol,$informat);
                    drop $newCol;
               ⑤ #else
                    #set($newCol = "${charvar}_new")
                    set $inlibname;
                    $newCol = input($charvar,$informat);
                 #end

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

1. $invarname is the Velocity variable for the role control. In this code, $charvar is set to the value of the invarname array. Because only one role is allowed, there can be only one item in the array, and you always fetch the role from the first location in the array.
2. $informatType is the Velocity variable that holds the return value for the option that you selected in the **Category of informat** option. $informatWidth is the contents of the **Informat width** option.

   In this line of code, you set a new Velocity variable `$informat` to be the concatenation of the `$informatType` variable, the `$informatWidth` variable, and a '.' character. The period is required because all SAS informats end with a period.
3. This line is the beginning of the SAS code and is the first line in the DATA step. In this code, the $outputDSName Velocity variable represents the name of the output data set. The name of the output data set is determined by the value of the `outputDSName` option, which you defined in this code in the metadata:

   ```
   <Option name="outputDSName" inputType="outputdata" defaultValue="Test"
        required="true">Name of output data set:</Option>
   ```

   In the user interface, this text box is labeled **Name of output data set**.
4. If you want the name of the variable that you are converting to remain the same, you select the **Same name?** check box in the user interface. Selecting this check box sets the $samename Velocity variable to 1. When `$samename == 1`, these steps occur:

   1. In the #SET directive, the *variable* (represented by the $charvar Velocity variable) is renamed *variable*_old (represented by the $newCol Velocity variable).
   2. In the INPUT function in the SET statement, the $charvar Velocity variable is set to the value of the $newCol Velocity variable, which has been converted using the informat specified by the $informat Velocity

variable. The value of the $informat Velocity variable is determined by the **Category of informat** and **Informat width** options in the user interface.

3. In the DROP statement, the variable named variable_old is dropped from the output data set.

5. If you want to create a new variable, do not select the **Same name?** check box in the user interface. When the $samename Velocity variable is not equal to 1, these steps occur:
   1. In the #SET directive, the $newCol Velocity variable is created. $newCol is set to the name of the variable that you assigned to the **Character variable to convert** role (represented by $charvar) and appended by _new. For example, $newCol could be set to Date_new.
   2. The SET statement reads the input data set (represented by the $inlibname Velocity variable).
   3. In the INPUT statement, the new variable (represented by $newCol and called charvar_new) is set to the value of the variable that you assigned to the **Character variable to convert** role and the informat that you specified using the **Category of informat** and **Informat width** options.

Click  to save your CTM code.

Step 6: Run the Task to View the Generated SAS Code and Resulting Output Data Set

Now, you compare the results and generated SAS code when the **Same name?** check box is selected and when this check box is not selected.

The Same Name Check Box Is Selected

If you select the **Same name?** check box in the user interface and then run the task, the resulting output data set contains the same number of variables as the input data set. Also, the names of the variables in the output data set are the same as the names in the input data set. In this example, the character variable in the input data set has been dropped, and a new numeric variable with the same name has been added to the output data set.

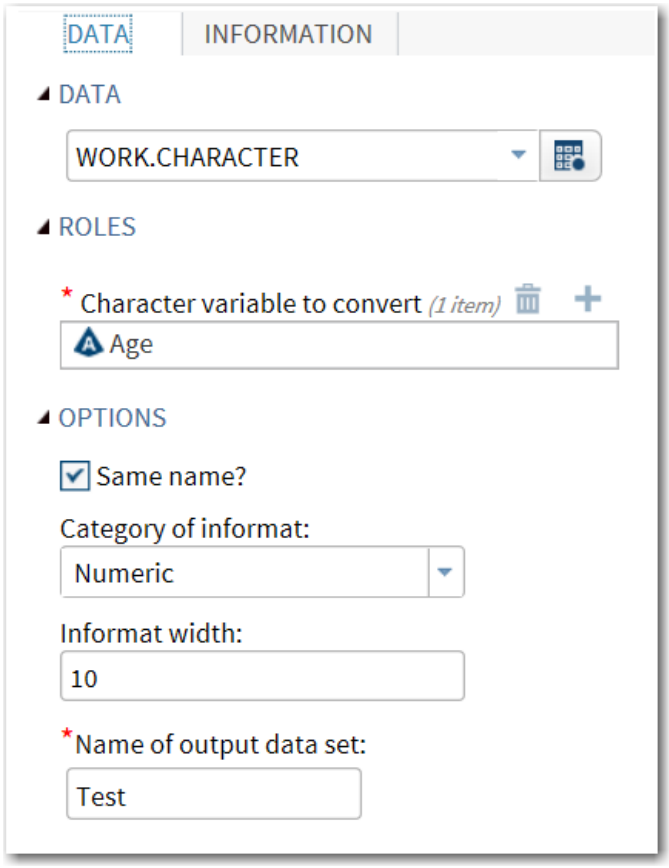
To view this result:

1. On the tab that contains your CTM code, click  to generate the user interface for the task.
2. In the user interface, select these options:
   - For the input data source, select **WORK.CHARACTER**.

     Note: If your SAS Studio session has timed out since you created this data set, this data set is no longer available from the temporary Work library. You must re-create this data set to continue. To re-create the data set, see “Create the Example Data Set” on page v.

- For the character variable to convert, select **Age**.
- Select the **Same name?** check box.
- Verify that `10` is in the **Informat width** box.

Here is the updated user interface:



The generated SAS code appears on the **Code** tab.

```
data WORK.Test;
   set WORK.CHARACTER(rename=Age=Age_old);
   Age=input (Age_old, BEST10.);
   drop Age_old;
run;
```

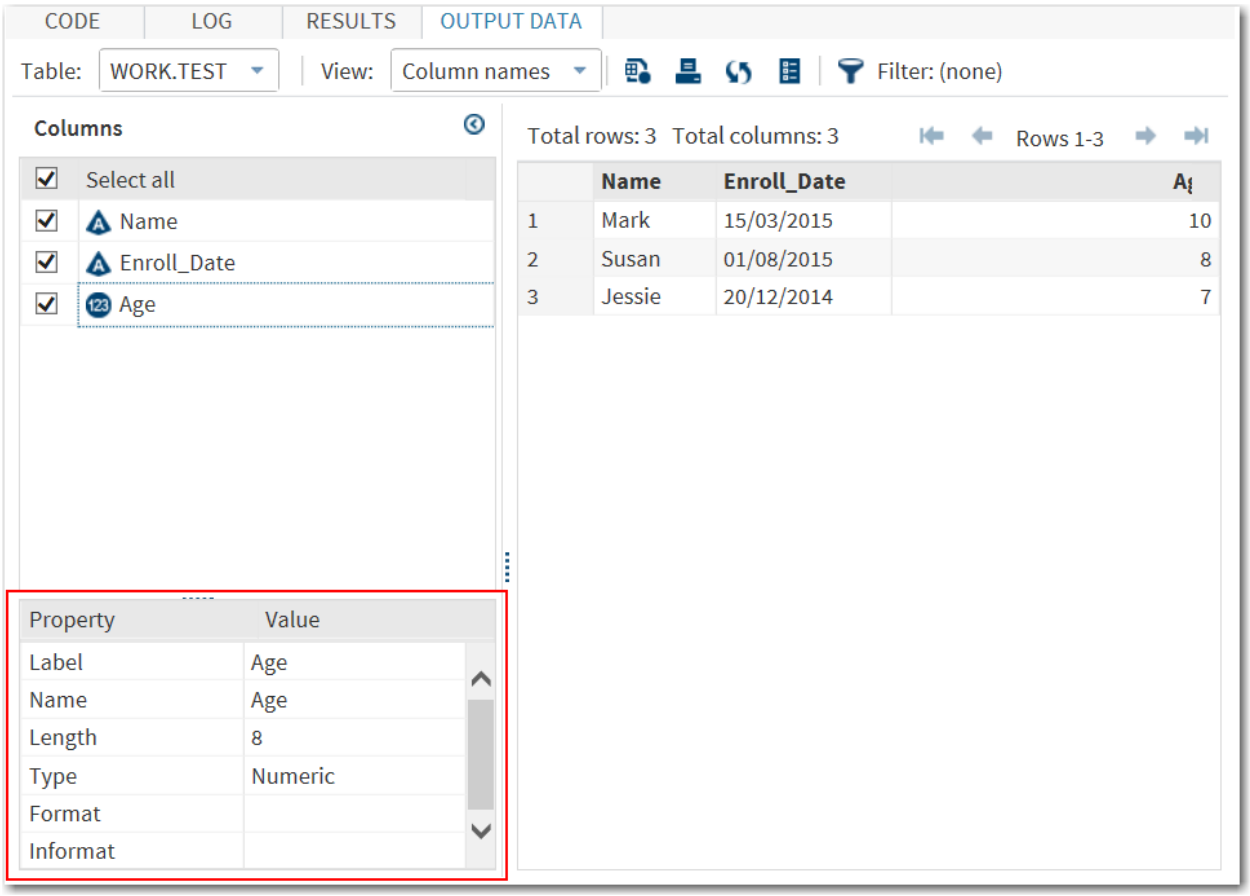
Note: The BEST10 informat was defined using the `returnValue` attribute and the `defaultValue` attribute in these two lines of code in the metadata:

```
<Option name="numericChoice" inputType="string"
   returnValue="BEST">Numeric</Option>
<Option name="informatLength" inputType="numbertext" minValue="1"
      defaultValue="10">Informat width:</Option>
```

3 Click  to run the task.

The results appear on the **OUTPUT DATA** tab. The output data set is called Work.Test. It contains the same three variables (Name, Enroll_Date, and Age)

as the input data set. However, Age is now a numeric variable instead of a character variable.



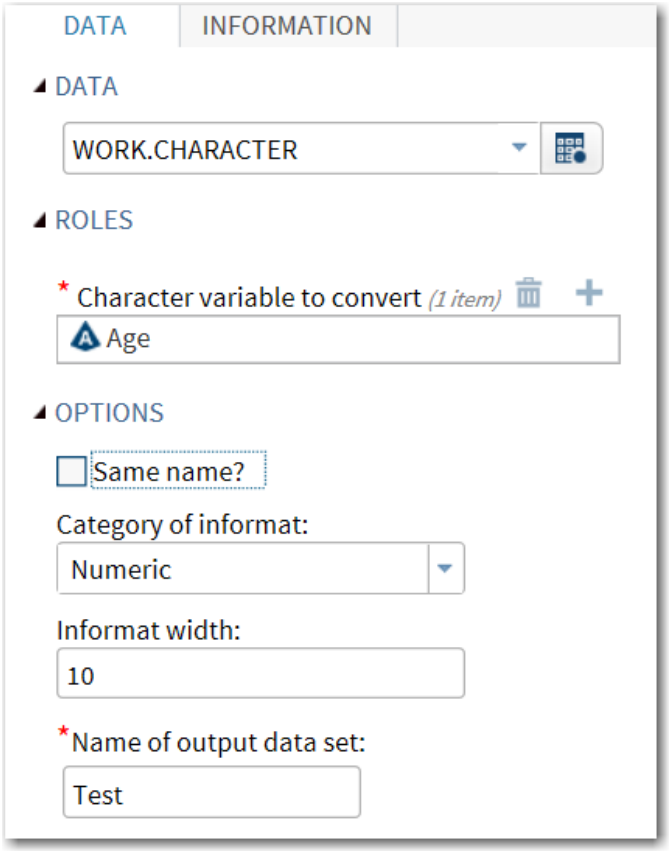
The Same Name Check Box Is Not Selected

When you run the task and the **Same name?** check box is not selected, the resulting data set includes the existing character variable and a new numeric variable called *variable-name*_new.

To view this result:

1 Return to the **DATA** tab in the user interface and clear the **Same name?** check box.

Here is the updated user interface:

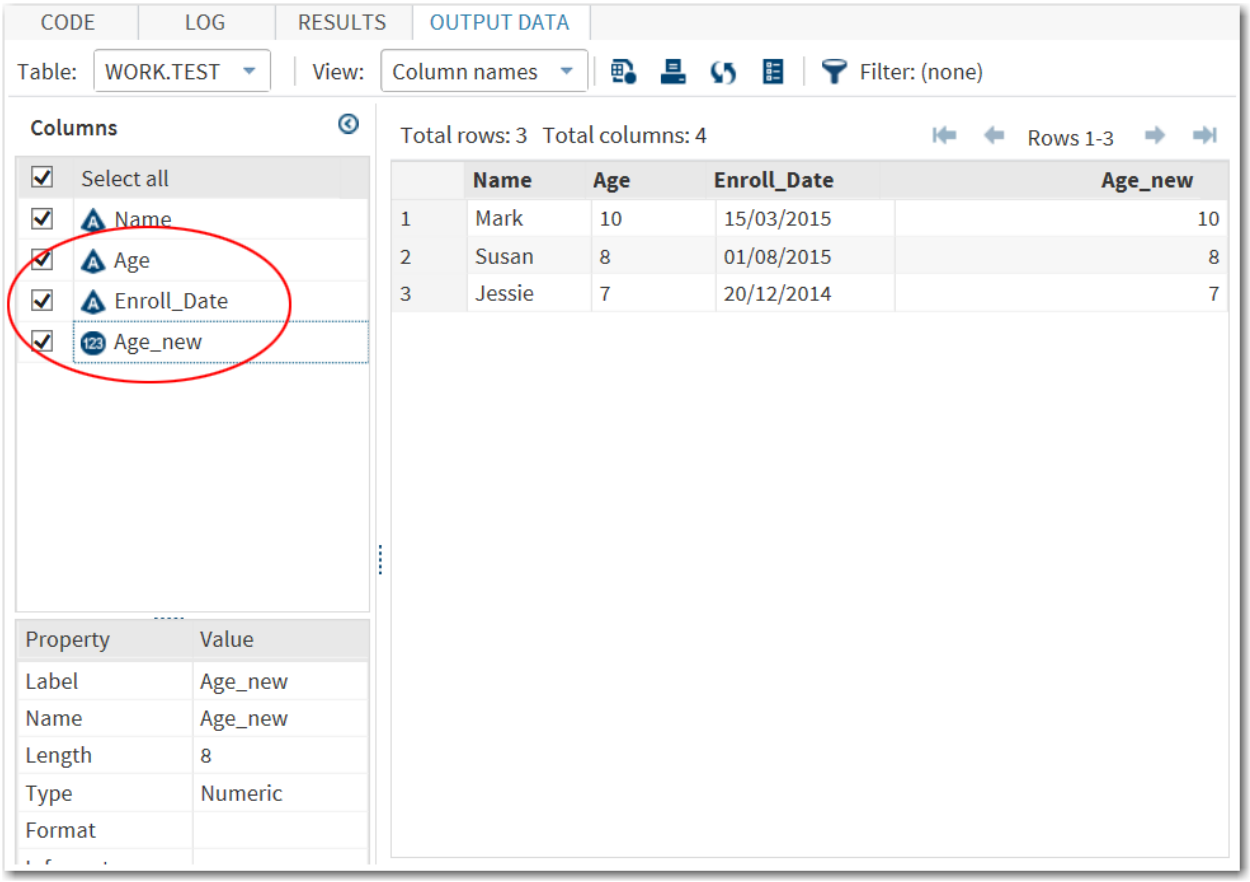


The generated SAS code appears on the **Code** tab.

```
data WORK.TEST;
   set WORK.CHARACTER;
   Age_new=input(Age, BEST10.);
run;
```

2 Click  to run the task.

The results appear on the **OUTPUT DATA** tab. The resulting output data set includes all of the variables from the input data set and a new variable called *variable-name*_new. In this example, the results contain both the Age and Age_new variables. Age is a character variable, and Age_new is a numeric variable.



You can now close the task.

Next Steps

Congratulations! You have created your first SAS Studio task. Next, you learn how to make this task more robust by using dependencies. For more information, see Chapter 3, “Going Further: Adding a Dependency,” on page 29.

3

Going Further: Adding a Dependency

About Dependencies

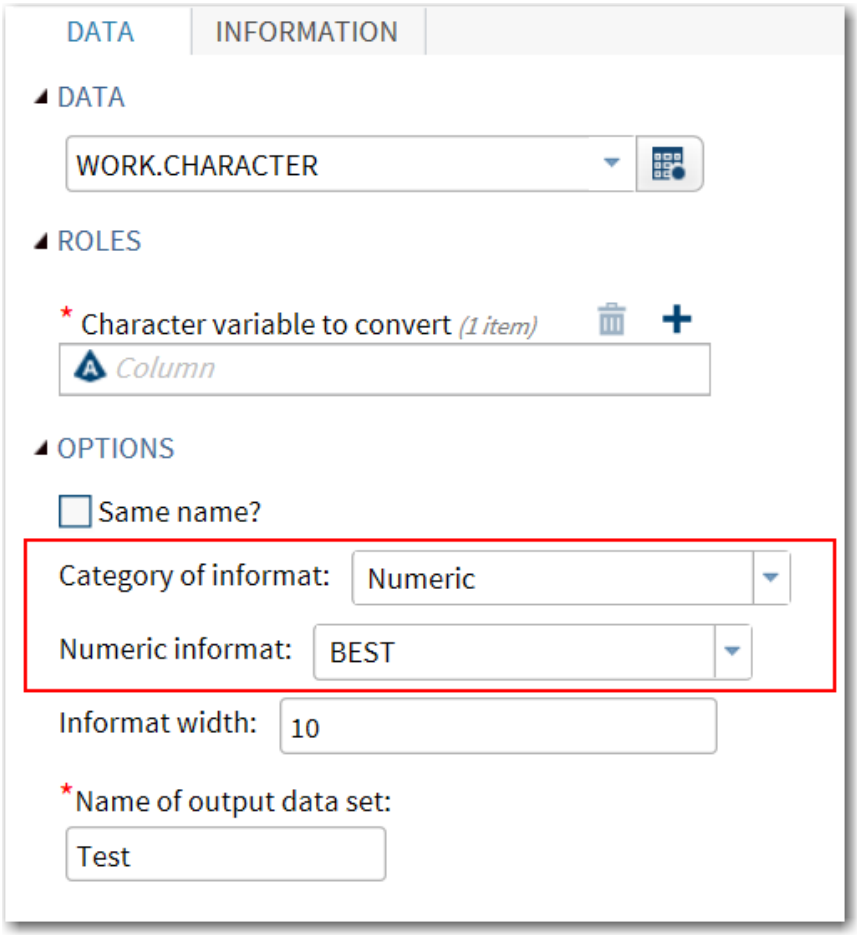
Chapter 2, “Creating a Basic Custom Task,” on page 5 shows how to convert a SAS program to a basic SAS Studio task.. As you become more comfortable creating SAS Studio tasks, you might want to use some of the more advanced features of the common task model, such as dependencies. The `Dependencies` element specifies how certain options or roles rely on one another in order for the task to work properly.

Here is an example of a dependency that you can create. In the previous chapter, you created the **Informat** drop-down list with two values: **Numeric** and **Date**.

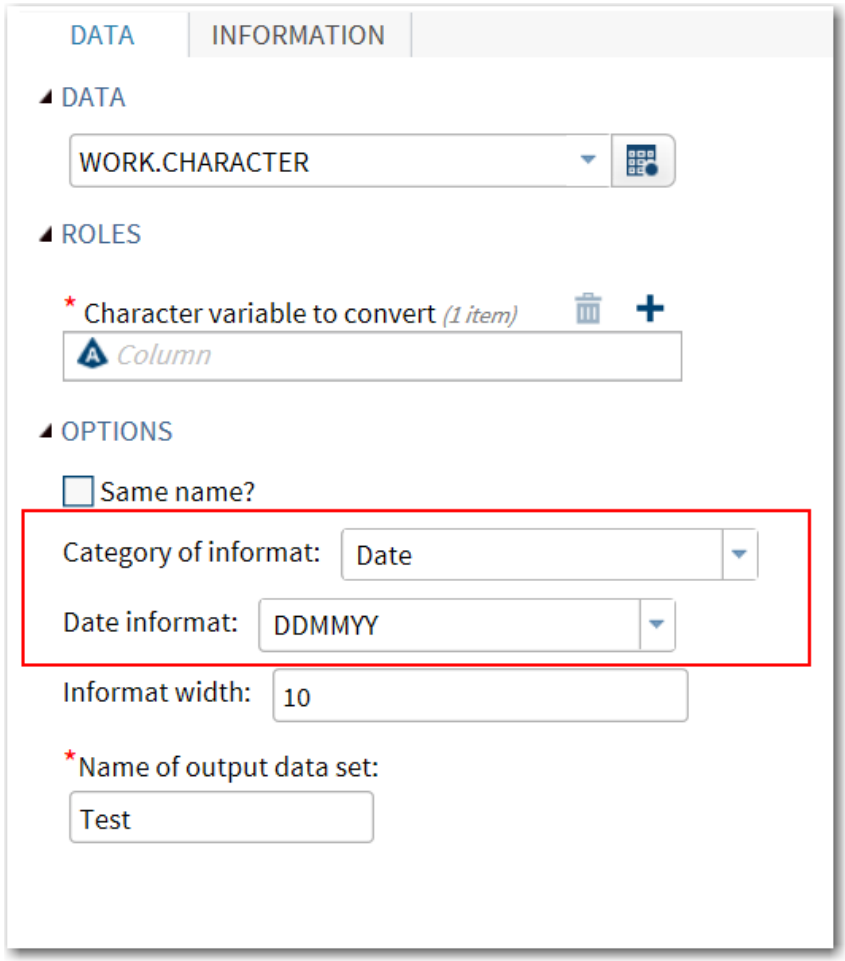
- If you selected **Numeric**, the BEST informat was used to create the new numeric variable.
- If you selected **Date**, the DDMMYY informat was used to create the new numeric variable.

However, SAS supports additional numeric and date informats. You want to give task users the ability to choose a specific numeric or date informat. After you complete this section, you see the following behavior in the user interface:

- When you select **Numeric** from the **Informat** drop-down list, the **Numeric informat** drop-down list appears.



- When you select **Date** from the **Informat** drop-down list, the **Date informat** option appears, and the **Numeric informat** option is hidden.



Step 7: Create the Numeric Informat and Date Informat Options

Define the Numeric Informat and Date Informat Options in the Metadata

Currently, here is the code in the `Options` element:

```
<Options>
   <Option name="dataTab" inputType="string">DATA</Option>
   <Option name="dataGroup" inputType="string">DATA</Option>
   <Option name="rolesGroup" inputType="string">ROLES</Option>
   <Option name="optionsGroup" inputType="string">OPTIONS</Option>

   <Option name="samename" inputType="checkbox">Same name?</Option>
   <Option name="informatType" inputType="combobox">Category
      of informat:</Option>
   <Option name="numericChoice" inputType="string" returnValue="BEST">
      Numeric</Option>
   <Option name="dateChoice" inputType="string" returnValue="DDMMYY">
      Date</Option>
```

```
   <Option name="informatLength" inputType="numbertext" minValue="1"
      defaultValue="10">Informat width:</Option>
   <Option name="outputDSName" inputType="outputdata" defaultValue="Test"
      required="true">Name of output data set</Option>
</Options>
```

First remove the highlighted sections from your code.

```
<Options>
   ...

   <Option name="samename" inputType="checkbox">Same name?</Option>
   <Option name="informatType" inputType="combobox">Category
      of informat</Option>
   <Option name="numericChoice" inputType="string" returnValue="BEST">
      Numeric</Option>
   <Option name="dateChoice" inputType="string" returnValue="DDMMYY">
      Date</Option>

   ...
</Options>
```

You remove the `returnValue` attributes from the `numericChoice` and `dateChoice` options because you are going to use the `returnValue` attribute from the new **Numeric informat** and **Date informat** options.

Add the highlighted code to create the new **Numeric informat** and **Date informat** options.

```
<Options>
   ...
   <Option name="informatType" inputType="combobox">Category
      of informat:</Option>
   <Option name="numericChoice" inputType="string">Numeric</Option>
   <Option name="dateChoice" inputType="string">Date</Option>

   <Option name="numericInformat" inputType="combobox">
      Numeric informat:</Option>
   <Option name="numeric1" inputType="string" returnValue="BEST">
      BEST</Option>
   <Option name="numeric2" inputType="string" returnValue="COMMA">
      COMMA</Option>

   <Option name="dateInformat" inputType="combobox">Date informat:
      </Option>
   <Option name="date1" inputType="string" returnValue="DDMMYY">
      DDMMYY</Option>
   <Option name="date2" inputType="string" returnValue="MMDDYY">
      MMDDYY</Option>

   <Option name="informatWidth" inputType="numbertext" minValue="1"
      defaultValue="10">Informat width:</Option>
   ...
</Options>
```

Here is an explanation of the new lines of code:

```
①<Option name="numericInformat" inputType="combobox">Numeric informat:</Option>
  <Option name="numeric1" inputType="string" returnValue="BEST">BEST</Option>
  <Option name="numeric2" inputType="string" returnValue="COMMA">COMMA</Option>

②<Option name="dateInformat" inputType="combobox">Date informat:</Option>
  <Option name="date1" inputType="string" returnValue="DDMMYY">DDMMYY</Option>
  <Option name="date2" inputType="string" returnValue="MMDDYY">MMDDYY</Option>
```

1. This code defines the **Numeric informat** option in the metadata. From the **Numeric informat** drop-down list, you can choose from these informats: BEST and COMMA.
2. This code defines the **Date informat** option in the metadata. From the **Date informat** drop-down list, you can choose from these informats: DDMMYY and MMDDYY.

Add the Numeric Informat and Date Informat Options to the User Interface

Now add the highlighted code for the new `numericInformat` and `dateInformat` options to the `UI` element.

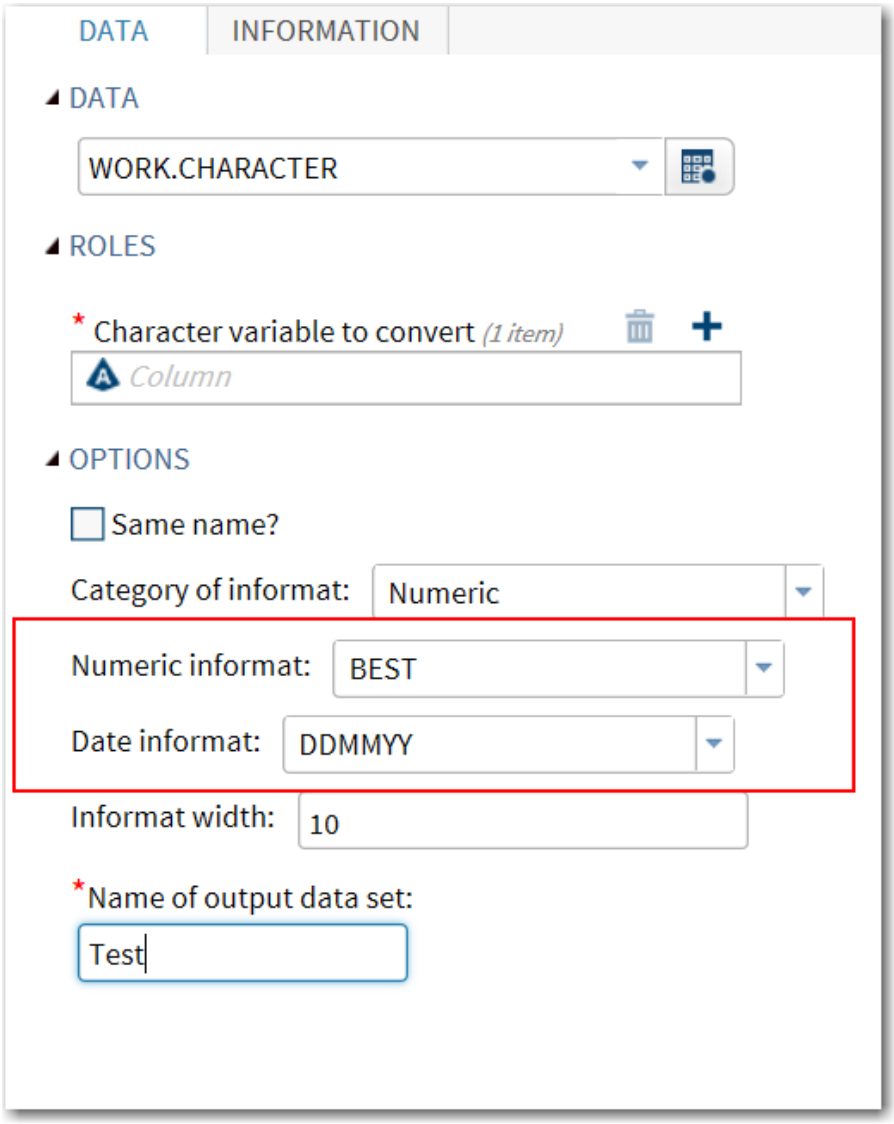
```
<Group option="optionsGroup" open="true">
  <OptionItem option="samename" />
  <OptionChoice option="informatType">
     <OptionItem option="numericChoice" />
     <OptionItem option="dateChoice" />
  </OptionChoice>

  <OptionChoice option="numericInformat">
       <OptionItem option="numeric1" />
     <OptionItem option="numeric2" />
  </OptionChoice>

    <OptionChoice option="dateInformat">
     <OptionItem option="date1" />
     <OptionItem option="date2" />
  </OptionChoice>

  <OptionItem option="informatWidth" />
  <OptionItem option="outputDSName" />
</Group>
```

```
    </OptionChoice>

(1) <OptionChoice option="numericInformat">
      <OptionItem option="numeric1" />
      <OptionItem option="numeric2" />
    </OptionChoice>
(2) <OptionChoice option="dateInformat">
      <OptionItem option="date1" />
      <OptionItem option="date2" />
    </OptionChoice>

    <OptionItem option="informatWidth" />
    <OptionItem option="outputDSName" />
```

1 This code creates the **Numeric informat** drop-down list in the user interface.

2 This code creates the **Date informat** drop-down list in the user interface.

When you run this code to generate the user interface, both the **Numeric Informat** and **Date Informat** options appear under the **OPTIONS** heading.



Close the task interface and return to the CTM code. Click  to save your CTM code.

Step 8: Create the Dependency

Use the `Dependencies` element to specify when each informat option should appear.

Add this code immediately after the closing `UI` element (`</UI>`).

```
<Dependencies>
<Dependency condition="$informatType == 'numericChoice'">
      <Target action="show" conditionResult="true" option="numericInformat"/>
      <Target action="hide" conditionResult="true" option="dateInformat"/>
```

```
    </Dependency>
    <Dependency condition="$informatType == 'dateChoice'">
       <Target action="hide" conditionResult="true" option="numericInformat"/>
       <Target action="show" conditionResult="true" option="dateInformat"/>
    </Dependency>
</Dependencies>
```

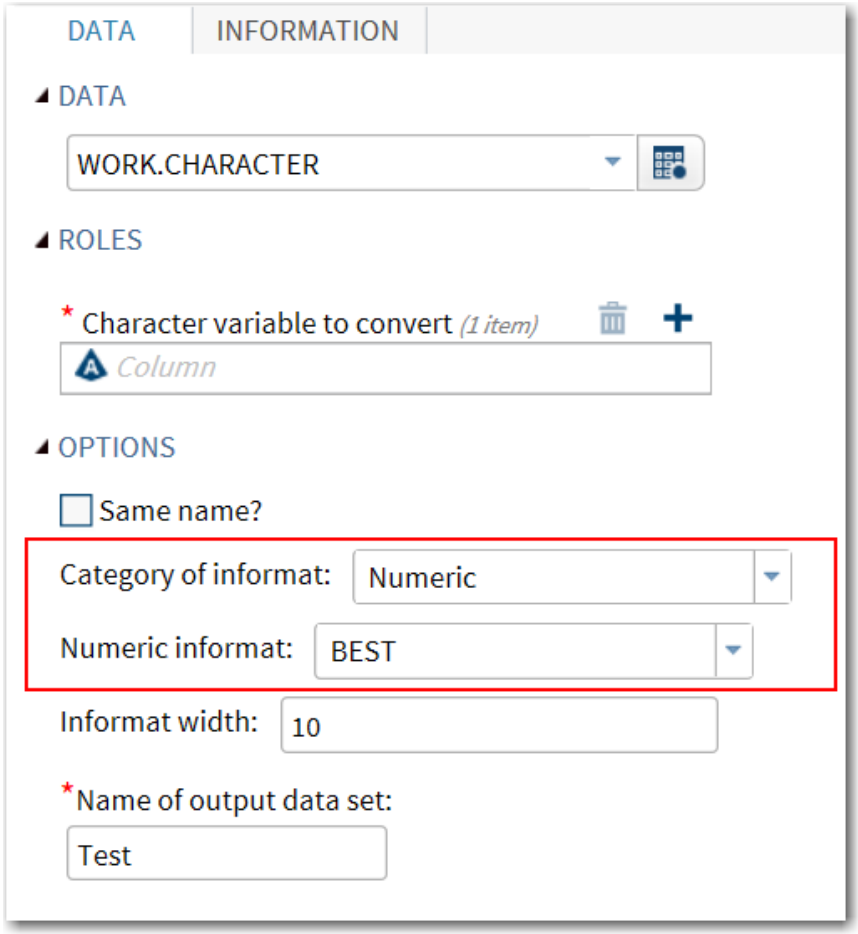
Here is an explanation of the code:

```
<Dependencies>
① <Dependency condition="$informatType == 'numericChoice'">
      <Target action="show" conditionResult="true" option="numericInformat"/>
      <Target action="hide" conditionResult="true" option="dateInformat"/>
  </Dependency>

②<Dependency condition="$informatType == 'dateChoice'">
      <Target action="hide" conditionResult="true" option="numericInformat"/>
      <Target action="show" conditionResult="true" option="dateInformat"/>
  </Dependency>
</Dependencies>
```

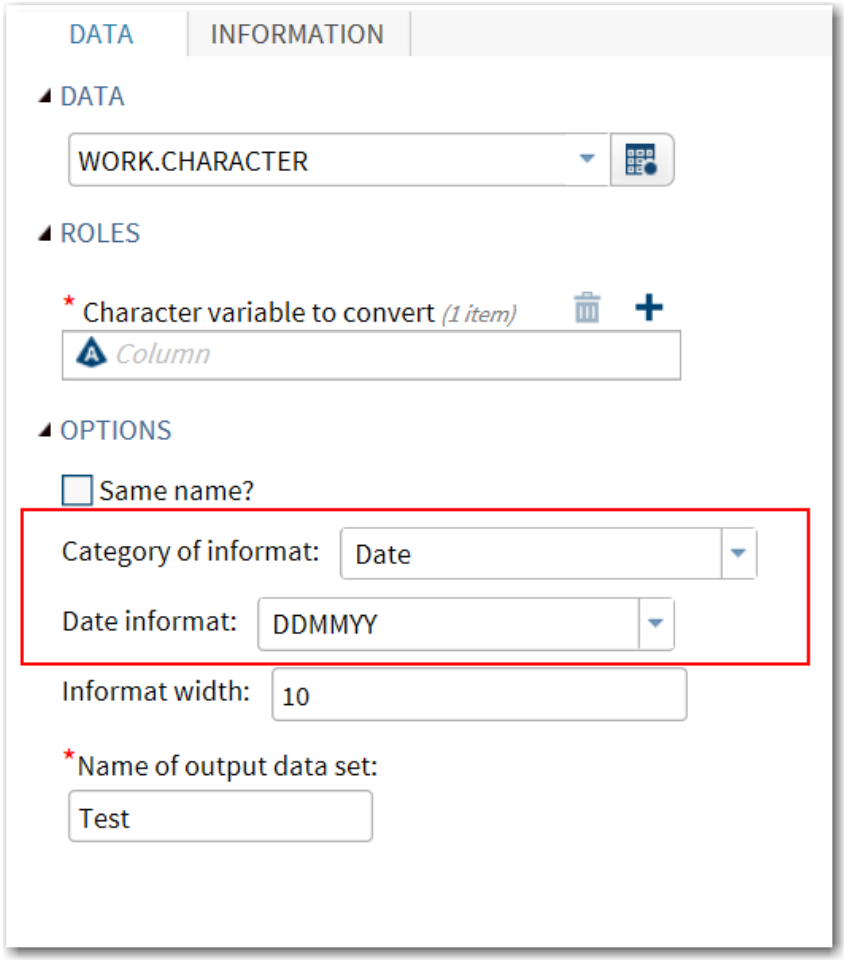
1. According to this dependency condition, when you select the **Numeric** option (named `numericChoice`) from the **Informat** drop-down list (named `$informatType`), the **Numeric informat** option (named `numericInformat`) is available in the user interface. The **Date informat** option (named `dateInformat`) is hidden.
2. According to this dependency condition, when you select the **Date** option (named `dateChoice`) from the **Informat** drop-down list (named `$informatType`), the **Date informat** option (named `dateInformat`) is available in the user interface. The **Numeric informat** option (named `numericInformat`) is hidden.

Click  to generate the user interface for the task.

When you run the task, the first value in the combobox control (**Numeric**) is selected by default. As a result, the **Numeric informat** option is displayed.



If you select **Date** from the **Category of Informat** drop-down list, the **Date informat** option appears, and the **Numeric informat** option is hidden.



Close the task interface and return to the CTM code. Click  to save your CTM code.

Step 9: Add the Dependency to the CodeTemplate Element

Now you need to update the `CodeTemplate` element to reflect these dependencies.

First remove the highlighted line of code.

```
<CodeTemplate>
   <![CDATA[

   #set($charvar=$invarname.get(0))

  #set($informat = "${informatType}${informatWidth}.")
```

```
   data $outputDSName;
   #if ($samename=='1')
      #set($newCol = "${charvar}_old")
      set $inlibname(rename=$charvar=$newCol);
      $charvar = input($newCol,$informat);
      drop $newCol;
   #else
      #set($newCol = "${charvar}_new")
      set $inlibname;
      $newCol = input($charvar,$informat);
   #end

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

Then add the highlighted code to account for the new dependency.

```
<CodeTemplate>
   <![CDATA[

   #foreach($item in $invarname) #set($charvar = $item)#end

   #if($informatType == "numericChoice")
      #set($informat = "${numericInformat}${informatWidth}.")
#else
   #set($informat = "${dateInformat}${informatWidth}.")
#end

   data $outputDSName;
   #if ($samename=='1')
      #set($newCol = "${charvar}_old")
      set $inlibname(rename=$charvar=$newCol);
      $charvar = input($newCol,$informat);
      drop $newCol;
   #else
      #set($newCol = "${charvar}_new")
      set $inlibname;
      $newCol = input($charvar,$informat);
   #end

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

Click  to save your CTM code.

Step 10: Run the Task and Review the Results

Now, if **Numeric** is selected from the **Category of informat** drop-down list, the value of the `$informat` Velocity variable is set to the values in the **Numeric informat** and **Numeric width** options.

To verify this behavior:

1 Click  to generate the user interface for the task.

2 Select these options:

- For the input data source, select **WORK.CHARACTER**.

  Note: If your SAS Studio session has timed out since you created this data set, this data set is no longer available from the temporary Work library. You must re-create this data set to continue. To re-create the data set, see “Create the Example Data Set” on page v.

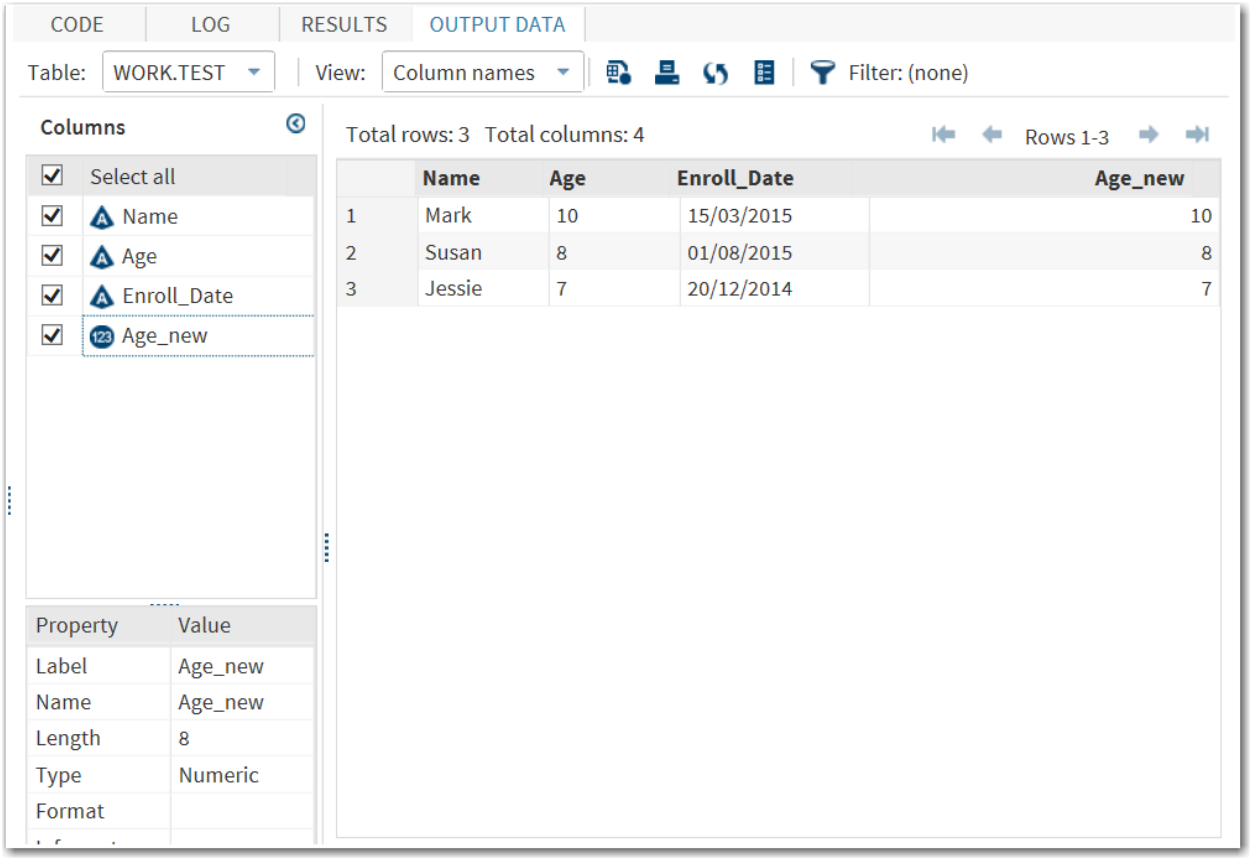
- Assign **Age** to the **Character variable to convert** role.
- From the **Category of informat** drop-down list, select **Numeric**.

On the **Code** tab, you should see the following SAS code:

```
data WORK.Test;
   set WORK.CHARACTER;
   Age_new=input(Age, BEST10.);
run;
```

3 Click  to run the task.

The output data set contains the Age_new variable, which is numeric.



If **Date** is selected from the **Category of informat** drop-down list, the value of the `$informat` Velocity variable is set to the value in the **Date informat** option.

To verify this behavior:

1 Return to the task interface and select these options:

- Assign **Enroll_Date** to the **Character variable to convert** role.
- From the **Category of informat** drop-down list, select **Date**.

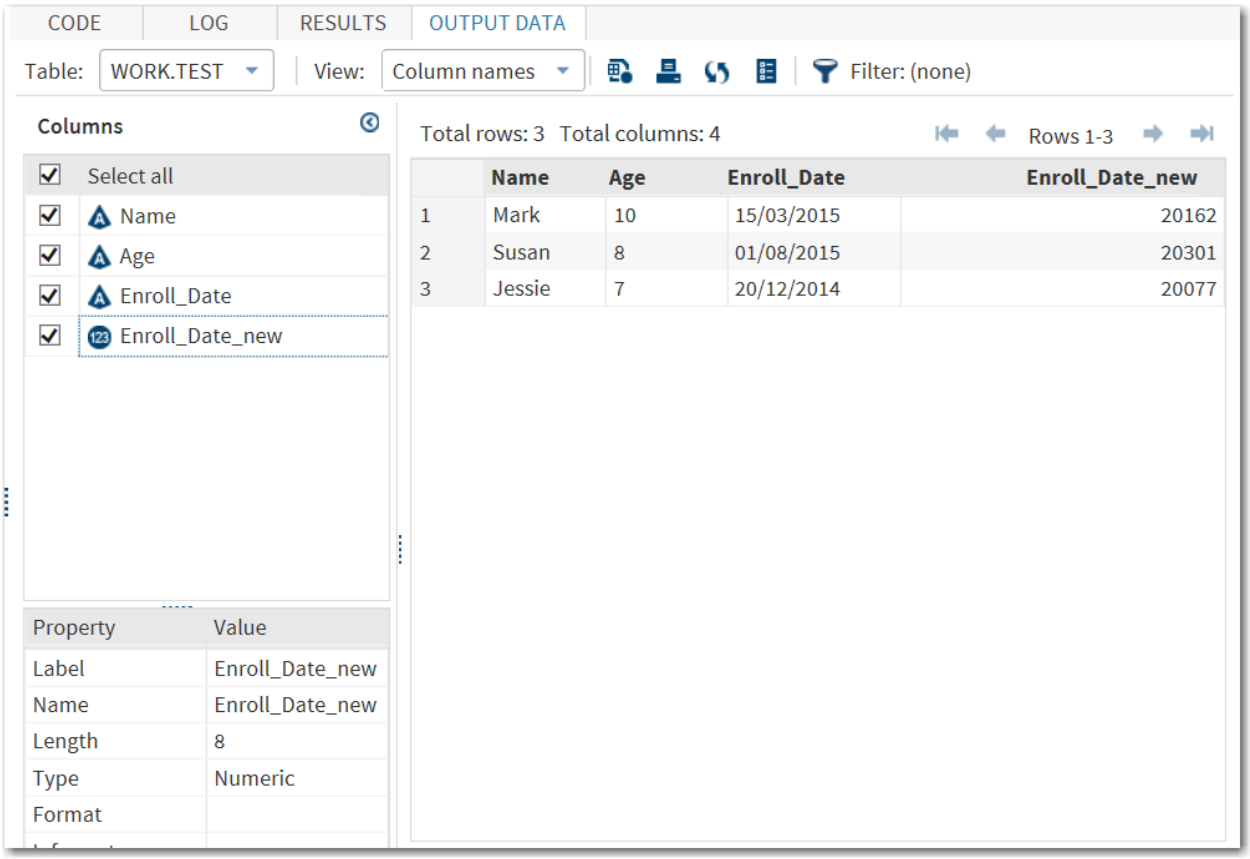
On the **Code** tab, you should see the following SAS code:

```
data WORK.Test;
   set WORK.CHARACTER;
   Enroll_Date_new=input(Enroll_Date, DDMMYY10.);
run;
```

2 Click  to run the task.

When you run the task, the output data set contains the _Enroll_Date_new variable, which is numeric. The value of the Enroll_Date_new variable is

determined using the DDMMYY10. informat. (For more information about informats, see *SAS Formats and Informats: Reference*.)

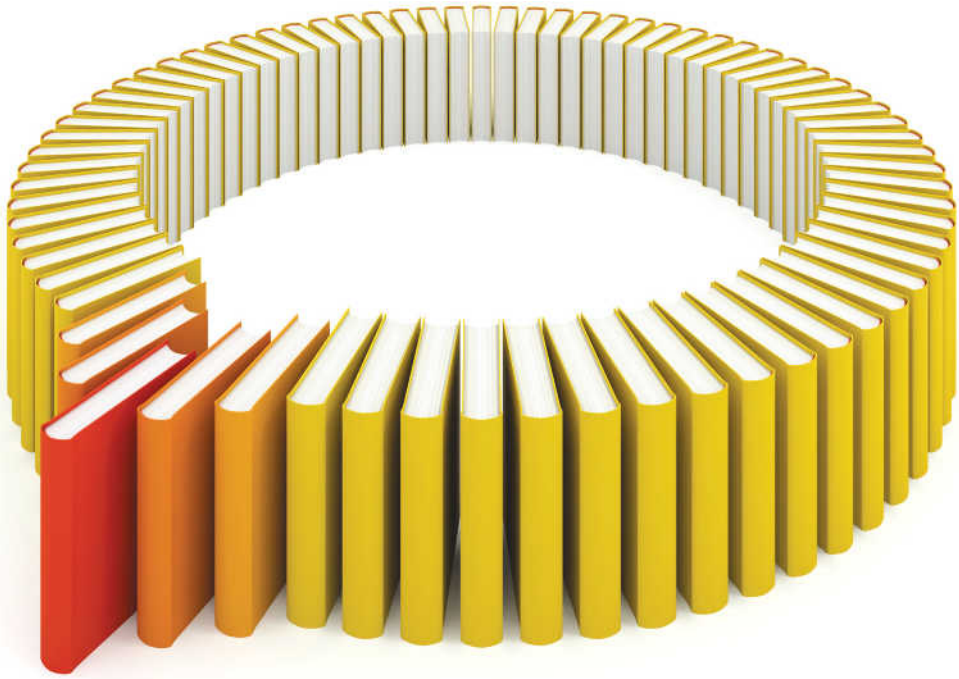


Recommended Reading

- *SAS Studio: Developer's Guide to Writing Custom Tasks*
- *SAS Studio: User's Guide*

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

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



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

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

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