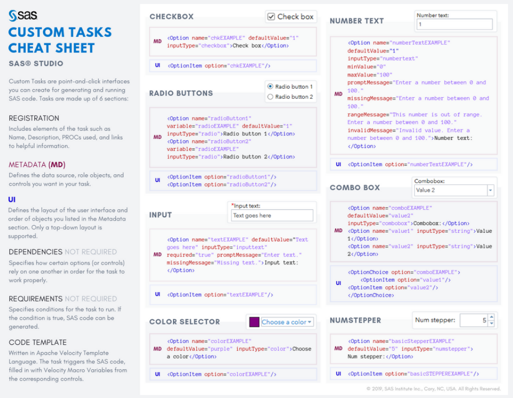
ADVANCED CUSTOM TASKS CHEAT SHEET

SAS® STUDIO

Custom Tasks are point-and-click interfaces you can create for generating and running SAS code.

THE ORIGINAL



CUSTOM TASKS CHEAT SHEET

SAS® STUDIO

Custom Tasks are point-and-click interfaces you can create for generating and running SAS code. Tasks are made up of 6 sections:

REGISTRATION
Includes elements of the task such as Name, Description, PROCs used, and links to helpful information.

METADATA **(MD)**
Defines the data source, role objects, and controls you want in your task.

UI
Defines the layout of the user interface and order of objects you listed in the Metadata section. Only a top-down layout is supported.

DEPENDENCIES NOT REQUIRED
Specifies how certain options (or controls) rely on one another in order for the task to work properly.

REQUIREMENTS NOT REQUIRED
Specifies conditions for the task to run. If the condition is true, SAS code can be generated.

CODE TEMPLATE
Written in Apache Velocity Template Language. The task triggers the SAS code, filled in with Velocity Macro Variables from the corresponding controls.

CHECKBOX

MD
```
<Option name="chkEXAMPLE" defaultValue="1"
inputType="checkbox">Check box</Option>
```

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

RADIO BUTTONS

MD
```
<Option name="radioButton1"
variable="radioEXAMPLE" defaultValue="1"
inputType="radio">Radio button 1</Option>
<Option name="radioButton2"
variable="radioEXAMPLE"
inputType="radio">Radio button 2</Option>
```

UI
```
<OptionItem option="radioButton1"/>
<OptionItem option="radioButton2"/>
```

INPUT

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

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

COLOR SELECTOR

MD
```
<Option name="colorEXAMPLE"
defaultValue="purple" inputType="color">Choose
a color</Option>
```

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

NUMBER TEXT

MD
```
<Option name="numberTextEXAMPLE"
defaultValue="1"
inputType="numbertext"
minValue="0"
maxValue="100"
promptMessage="Enter a number between 0 and
100."
missingMessage="Enter a number between 0 and
100."
rangeMessage="This number is out of range.
Enter a number between 0 and 100."
invalidMessage="Invalid value. Enter a
number between 0 and 100.">Number text:
</Option>
```

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

COMBO BOX

MD
```
<Option name="comboEXAMPLE"
defaultValue="value2"
inputType="combobox">Combobox:</Option>
<Option name="value1" inputType="string">Value
1</Option>
<Option name="value2" inputType="string">Value
2</Option>
```

UI
```
<OptionChoice option="comboEXAMPLE">
    <OptionItem option="value1"/>
<OptionItem option="value2"/>
</OptionChoice>
```

NUMSTEPPER

MD
```
<Option name="basicStepperEXAMPLE"
defaultValue="5" inputType="numstepper">
Num stepper:</Option>
```

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

Before diving in to the Advanced Custom Tasks Cheat Sheet, make sure you've checked out the original.

ADVANCED TOPICS

- Optional Task Sections
  - Dependencies
  - Requirements
- Advanced Apache Velocity Template Language
- Working with CAS tables
- Using the Markdown control (new in SAS® Studio 5.2)

DEPENDENCIES SECTION

The Dependencies section specifies how certain options (or controls) rely on one another in order for the task to work properly. A dependency is when one option or role relies on another option or role to initiate an action on it.

To create a **dependency**:

- Specify the **dependency condition**
- Specify the **target** control that you want to change if the condition is met
- Specify the **action** that you want to take on the control (options include *"show," "hide," "enable," "disable,"* and *"set"*)
- Specify the **conditionResult,** to take action when the condition is *"true" or "false"*

CODE EXAMPLE

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

Not all dependencies are evaluated each time code is generated. When a task is first opened, all of the dependencies are run to establish the initial values. When a user interacts with a particular UI element, only conditions that contain the name of that element will be evaluated and any/all valid actions will be executed. Dependencies that aren't associated with the current interaction aren't evaluated because their values haven't changed, so there is no need to re-evaluate them.

REQUIREMENTS SECTION

The Requirements section specifies conditions for the task to run. If the condition is true, SAS code can be generated.

To create a **requirement**:

- Specify the **requirement condition**
  - In most cases, it is easier to specify the condition that would fail the requirement and then negate it (by using the standard "!") rather than specify the condition that would pass the requirement.
- Specify the **message** to be displayed if the requirement is not met.

CODE EXAMPLE

```
<Requirements>
  <Requirement condition="$AVAR.size() %gt; $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>
```

The timing of Requirements being evaluated is significant. Since Dependencies can affect the state of the UI as well as the state of the velocity variables, Requirements are always evaluated after Dependencies are evaluated. This way, changes due to any specified dependencies are part of the evaluation of whether requirements have been satisfied.

ADVANCED APACHE VELOCITY CODE

Velocity Terminology:

VARIABLES: $variables are Velocity references that usually refer to a specific control in your task. All references are preceded by the "$" symbol.

DIRECTIVES: #directives are Velocity statements that perform some action and allow for code manipulation. These are preceded by the "#" symbol.

METHODS: $variable.methods() are Velocity references that refer to Java methods that perform a useful action on the variable.

PREDEFINED VELOCITY VARIABLES

PREDEFINED VARIABLE	METHOD	DESCRIPTION
`$CTMUtil`	`quoteString()`	Wraps a string in single quotes
	`doubleQuoteString()`	Wraps a string in double quotes
	`isProductLicensed()`	Checks to see if a specific product is installed
	`toSASName()`	Transforms a string into SAS naming conventions
`$CTMMathUtil`	`getMin()`	Returns the smallest value of an array of doubles passed in
	`getMax()`	Returns the largest value of an array of doubles passed in
	`getSum()`	Returns the sum of all the values of an array of doubles passed in

USEFUL VELOCITY DIRECTIVES

DIRECTIVE	DESCRIPTION
`#if #elseif #else #end`	Allows conditional logic
`#foreach #end`	Loops through items in a list
`#set`	Sets a value for a velocity variable (similar to a SAS %LET statement)
`#break`	Stops a loop in a #foreach

USEFUL JAVA.UTIL.LIST METHODS

METHOD	DESCRIPTION
`.isEmpty()`	Returns true(1) if the list is empty
`.size()`	Returns the size of the list
`.get(#)`	Gets a certain value in the list Helpful when: You have a role selector that you have restricted to allow only one variable. You can call $var.get(0) to get that variable without having to loop through the list

WORKING WITH CAS TABLES

Specify libraryEngineInclude="CAS" to restrict data set selector to only look at CAS libraries:

```
<DataSources>
   <DataSource libraryEngineInclude="CAS" name="dataset" where="true">
   </DataSource>
</DataSources>
```

Parse two-level data set name for use in CAS Actions (which separate table name and library name):

```
#if($proc == 'procCas')
#set ($datasetCASLibref = $dataset.getLibrary())
#set ($datasetCASLib = "%sysfunc(getlcaslib($datasetCASLibref))")
#set ($datasetCASTable = $dataset.get("table"))
#end
proc cas;
action sampling.srs /
 table={caslib="%sysfunc(getlcaslib($datasetCASLibref))",
name="$datasetCASTable"};
quit;
```

MARKDOWN

Available in SAS Studio 5.2, the markdownText object allows you to create formatted text using the Markdown language.

The Sample Task built into SAS Studio 5.2 shows the full list of capabilities of the markdown object.

```
<Option inputType="markdown"
name="markdownTextEXAMPLE">
# Heading
## Subheading
Text styling:  _italic_, **bold**, `monospace`.
&gt; Indented, block quote
Bulleted list:
  * Apples
  * Oranges
Link: The [SAS website](http://www.sas.com/).
Image: ![SAS](http://www.....png)
</Option>
```

HELPFUL LINKS

SGF Paper: "Developing Your Own SAS® Studio Custom Tasks for Advanced Analytics"

SAS® Studio 3.8 Developer's Guide to Writing Custom Tasks

Custom Task Tuesday article series on SAS Communities

FREE e-Learning on SAS® Studio Custom Tasks

SGF Paper: "Teach Them to Fish—How to Use Tasks in SAS® Studio to Enable CoWorkers to Run Your Reports Themselves"

Custom Task Tuesday GitHub Page

Follow author of #CustomTaskTuesday @OliviaJWright on Twitter

Apache Velocity Website with Resources