UNICOS Internal Note

LOGIC TEMPLATE USAGE FOR UNICOS-CPC APPLICATIONS

This document explains how to use logic templates using UAB generator for Schneider and Siemens applications.

DOCUMENT PREPARED BY: Adrien Chiron [EN-ICE-PLC]
DOCUMENT CHECKED BY: Benjamin Bradu [EN-ICE-PLC]
DOCUMENT APPROVED BY: [Checkers] [Approvers]
<table>
<thead>
<tr>
<th>REV. NO.</th>
<th>DATE</th>
<th>PAGES</th>
<th>DESCRIPTION OF THE CHANGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>03.04.2012</td>
<td>All</td>
<td>Explain template mechanism (A. Chiron)</td>
</tr>
<tr>
<td>1.0</td>
<td>04.04.2012</td>
<td>All</td>
<td>Document completion (B. Bradu)</td>
</tr>
<tr>
<td>1.1</td>
<td>17.04.2012</td>
<td>All</td>
<td>Modif after Adrien comments (B. Bradu)</td>
</tr>
<tr>
<td>1.2</td>
<td>15.11.2012</td>
<td></td>
<td>Remove restriction on &lt; and &gt; characters for Schneider templates, see UCPC-1017. Need RP 1.3.2 or later. (W.Booth)</td>
</tr>
<tr>
<td>1.3</td>
<td>14.12.2012</td>
<td>section 2.4</td>
<td>Added explanation of findMatchingInstances compound conditions (A.Chiron)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>NOTE:</strong> 'multiple' conditions is only possible with CPC Wizard version 1.4.0 or later</td>
</tr>
<tr>
<td>1.4</td>
<td>16.01.2013</td>
<td>section 3</td>
<td>Add a caution for multiple alarm (A.Chiron)</td>
</tr>
<tr>
<td>1.5</td>
<td>28.01.2013</td>
<td>sections 3.1, 3.2 and 3.3</td>
<td>Update documentation for LparamX change [UCPC-1094] (A.Chiron)</td>
</tr>
</tbody>
</table>
## TABLE OF CONTENTS

1. User Template organization .............................................................. 4  
   1.1 Template mechanism ................................................................. 4  
   1.2 Template locations ................................................................. 5  
   1.3 Standard templates ................................................................. 7  
2. How to write a user template ............................................................. 8  
   2.1 Creation of user templates ....................................................... 8  
   2.2 Modification of a user templates .............................................. 9  
   2.3 Definition of configured and not configured alarms ..................... 12  
   2.4 findMatchingInstances and examples of 'complex' conditions ....... 13  
3. Example with an OnOff dependent logic Template ............................. 15  
   3.1 Presentation of the use case ..................................................... 15  
   3.2 Siemens Template ................................................................. 17  
   3.3 Schneider Template ............................................................... 19
1. User Template organization

1.1 Template mechanism

The logic templates for Dependent Logic and PCO are split in two files:

- The **main template** where the mandatory logic is done. **This template is always called and must not be modified.**

- The **standard template** where the default logic configurations are done or the **user template** where the application expert will do his own logic.

The main template is called once for each instance of each object. For example, if we have five AnaDO objects in a Siemens project, the main template `S7Logic_AnaDO_Template.py` will be called five times. Then from the main template there are two exclusive possibilities:

1. Calling the **standard template**.
2. Calling the **user template**.

If nothing is specified in the "CustomLogicSections" (see Figure 3) of the specification file, the standard template will be called otherwise the user template specified will be called.

To sum up, here is a schematic of the execution flow of one template (let’s take the example of the AnaDO Siemens template but it is the same behavior for the others):

![Activity diagram of the templates](image)

Figure 1: Activity diagram of the templates.
1.2 Template locations

Templates are python files (.py) located under the UAB project folder in the following directory:

- for S7 Logic generator: Resources/S7LogicGenerator/Rules/
- for Unity logic generator: Resources/UnityLogicGenerator/Rules/

Subfolders are organized as follow in a *S7 project*:

- **CommonTemplates**: contains main templates common for several objects
- **DependentLogicTemplates**: contains standard templates and main templates for each dependent logic (OnOff, Analog, Anadig, AnaDO, Controller, MFC, PCO)
- **GlobalTemplates**: contains main templates for global functions
- **PCOTemplates**: contains standard templates and main templates for each PCO section (BL, CDOL, CL, GL, IL, INST, SL, TL)
- **UserSpecific**: contain user templates. User can then create subfolders to organize his user templates as he wants under this directory.

For Unity projects, user can decide to generate its code in FBD or in ST. The user templates can be used ONLY for ST templates, not for FBD. Hence, the split of templates is done only for ST templates. Subfolders are organized as follow in a *Unity project*:

- **CommonTemplates**: contains FBD templates for several objects and templates common for FBD and ST parts (BL, CDOL, INST, SL).
- **FBDTemplates**: contains FBD templates for dependent logic (OnOff, Analog, Anadig, AnaDO, Controller, MFC, PCO) and PCO sections (CL, GL, IL, TL).
- **STTemplates**: contains ST standard and main templates for dependent logic (OnOff, Analog, Anadig, AnaDO, Controller, MFC, PCO) and PCO sections (CL, GL, IL, TL).
- **UserSpecific**: contain user templates. User can then create subfolders to organize his user templates as he wants under this directory.
Figure 2: Template locations under the UAB project folder for S7 and Unity projects

Then, in your UNICOS XML spec file, you can call user templates from the “UserSpecific” directory. You can also create subfolders under “UserSpecific” as shown in Figure 2 where subfolders “DL” and “PCO” are user subfolders. For instance, if you want to link a Dependent Logic (DL) template for a field object located in “Resources/S7LogicGenerator/Rules/UserSpecific/DL/UAPN_TC.py”, you should write in the corresponding column “DL/UAPN_TC.py”, see Figure 3.

Figure 3: Call of user templates from UNICOS XML spec file
1.3 Standard templates

The standard templates for PCO are the following:

- S7Logic_BL_Standard_Template.py
- S7Logic_CDOL_Standard_Template.py
- S7Logic_CL_Standard_Template.py
- S7Logic_GL_Standard_Template.py
- S7Logic_IL_Standard_Template.py
- S7Logic_INST_Standard_Template.py
- S7Logic_SL_Standard_Template.py
- S7Logic_TL_Standard_Template.py

The standard templates for the Dependent Logic are the following:

- S7Logic_AnaDO_Standard_Template.py
- S7Logic_Analog_Standard_Template.py
- S7Logic_AnalogDigital_Standard_Template.py
- S7Logic_Controller_Standard_Template.py
- S7Logic_MassFlowController_Standard_Template.py
- S7Logic_OnOff_Standard_Template.py
- S7Logic_ProcessControlObject_Standard_Template.py
2. How to write a user template

2.1 Creation of user templates

If you need something different of what is configured in the default standard templates, you can choose to create your own templates. Let’s take the example with the OnOff dependent logic template.

In order to do this an easy technique is to copy and paste the standard template (for instance, S7Logic_OnOff_Standard_Template.py) in the UserSpecific directory of your application, see Figure 4.

![Figure 4. Copy/Paste of a standard template to UserSpecific directory](image)

Then, rename your new UserSpecificTemplate as you wish, see Figure 5.
2.2 Modification of a user templates

The templates are decomposed in four main parts, see Figures 6 and 7 where the default template is represented for S7 and Unity:

1. **Header**: Declaration of functions: **must not be modified**

2. **Variables declaration**: You can declare here the PLC variable if you need (temporary variables in S7 and global variables in Unity).

3. **Output Function**: This part contains the function to be generated in the output file. This part has to be modified according to your application.

4. **Alarm Management**: If the template embeds alarm definitions (Dependent Logic or Interlock Logic), this part allows user to define alarm inputs in 2 ways:
   
   a) Keep the standard functions calling the not configured alarms (WriteNotConfiguredDAParameters WriteNotConfiguredAAParameters) and you will have then to complete the alarm conditions in the generated file afterward.

   b) Remove the call of the standard functions for the not configured alarms (WriteNotConfiguredDAParameters WriteNotConfiguredAAParameters) and write your alarm conditions directly in the user template.

*Figure 5. Rename the standard template according to your application*
Figure 6: Default User template for OnOff dependent logic (S7 template)
from java.util import ArrayList
import SchLogio_DefaultAlarms_Template
reload(SchLogio_DefaultAlarms_Template)

def OnOffLogic(self, PluginInstance, Master, Name, program_DL, lParamVector):
    --- Lparam1, Lparam2, Lparam3, Lparam4, Lparam5, Lparam6, Lparam7, Lparam8, Lparam9, Lparam10
    --- program_DL = program_DL + '"
    --- Master: ---#Master\n    --- Name: ---#Name\n    --- Lparam1: ---#lParam1\n    --- Lparam2: ---#lParam2\n    --- Lparam3: ---#lParam3\n    --- Lparam4: ---#lParam4\n    --- Lparam5: ---#lParam5\n    --- Lparam6: ---#lParam6\n    --- Lparam7: ---#lParam7\n    --- Lparam8: ---#lParam8\n    --- Lparam9: ---#lParam9\n    --- Lparam10: ---#lParam10"
    
    """--- Declaration of Variables in Memory ---
    thePlugin.writeVariable("""Variables name=\"dp\_FieldName\" type=\"REAL\"\""")
    """--- Process Input ---
    program_DL = program_DL + '"
    # Position Management
    $Name\_AuxOut := #Master\_S.\$RunCnt AND $lParam5.\$Posr \& \&; 1.0 ;
    $Name\_AuxOff := NOT \#Master\_S.\$RunCnt OR $lParam1.\$Posr \& \&; 2.0 ;
    """
    program_DL = program_DL + '"
    $Name\_IOErr := 0;
    $Name\_IOIn := 0;

    """--- Get Single and multiple FS/TS/SL/AL Digital alarms that are child of the current
    # theFAD, theFAD, theFAD, theSLA, theSLA, theSLA, theSLA, theSLA, theSLA, theSLA
    # Get Single and multiple FS/TS/SL/AL Analog alarms that are child of the current
    # theFAD, theFAD, theFAD, theSLA, theSLA, theSLA, theSLA, theSLA, theSLA, theSLA
    program_DL = program_DL + '"(** Full Stop Interlock Logic **)"
    --- LParam2\_FS_AuEEL := NOT $lParam2.\$IOErrW ;
    --- LParam2\_FS_AuEL := TRUE ;
    """--- $program_DL = SchLogio_DefaultAlarms_Template.writeNotConfiguredDAParameters(thePlugin)
    """
    program_DL = program_DL + '"
    """--- Temporary Stop Interlock Logic : to complete ---
    ""
    program_DL = program_DL + '"
    """--- Start Interlock Logic : to complete***
    """
    $d\_FieldName := $lParam2.\$Posn-\$lParam1.\$Posn ;
    $Name\_IOErr := $FieldName.\$IOErr \& \&; 0.0 ;
    $Name\_IOIn := $FieldName.\$Posn-\$lParam2.\$Posn ;

    """--- $program_DL = SchLogio_DefaultAlarms_Template.writeNotConfiguredDAParameters(thePlugin)
    """--- $program_DL = SchLogio_DefaultAlarms_Template.writeNotConfiguredDAParameters(thePlugin)
    """--- $program_DL = SchLogio_DefaultAlarms_Template.writeNotConfiguredDAParameters(thePlugin)
    """--- $program_DL = SchLogio_DefaultAlarms_Template.writeNotConfiguredDAParameters(thePlugin)
    """

    """--- Alarms : to complete ***
    ""
    program_DL = program_DL + '"
    """--- $program_DL = SchLogio_DefaultAlarms_Template.writeNotConfiguredDAParameters(thePlugin)
    """--- $program_DL = SchLogio_DefaultAlarms_Template.writeNotConfiguredDAParameters(thePlugin)
    """--- $program_DL = SchLogio_DefaultAlarms_Template.writeNotConfiguredDAParameters(thePlugin)
    """

    return program_DL

1. Header

2. Variable declaration

3. Unity Section

4. Alarm management

Figure 7: Default User template for OnOff dependent logic (Unity template)
2.3 Definition of configured and not configured alarms

2.3.1 Digital Alarms

The following table shows up in which function the different parameters are affected according to the input chosen in the different columns of the specification file needed to parameterize digital alarms.

<table>
<thead>
<tr>
<th>Spec Column</th>
<th>User input</th>
<th>WriteConfiguredDAParameters</th>
<th>WriteNotConfiguredDAParameters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><em>(NOTE: user does not see this function, this is written by main template automatically and cannot be changed)</em></td>
<td><em>(NOTE: user can choose to remove this function call from the user template, in which case he will be responsible for writing the code for the appropriate pins himself)</em></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>Empty</th>
<th>IOError and IOSimu and I</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Not Empty</td>
<td>IOError and IOSimu and I</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Delay</th>
<th>&quot;logic&quot;</th>
<th>PAIDt</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reference to an object</td>
<td>PAIDt</td>
<td></td>
</tr>
</tbody>
</table>

2.3.2 Analog Alarms

The following table shows up in which function the different parameters are affected according to the input chosen in the different columns of the specification file needed to parameterize analog alarms.

<table>
<thead>
<tr>
<th>Spec column</th>
<th>User input</th>
<th>WriteConfiguredAAParameters</th>
<th>WriteNotConfiguredAAParameters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><em>(NOTE: user does not see this function, this is written by main template automatically and cannot be changed)</em></td>
<td><em>(NOTE: user can choose to remove this function call from the user template, in which case he will be responsible for writing the code for the appropriate pins himself)</em></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HH Alarm</th>
<th>Empty</th>
<th>AuEHH</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(same for: H Warning</td>
<td>Number</td>
<td>HH</td>
<td>AuEHH and HH</td>
</tr>
<tr>
<td>L Warning</td>
<td>&quot;logic&quot;</td>
<td>AuEHH and HH</td>
<td></td>
</tr>
<tr>
<td>LL Alarm</td>
<td>Reference to an object</td>
<td>HH</td>
<td>AuEHH</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>Empty</th>
<th>IOError and IOSimu and I</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Not Empty</td>
<td>IOError and IOSimu and I</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Delay</th>
<th>&quot;logic&quot;</th>
<th>PaIDt</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reference to an object</td>
<td>PaIDt</td>
<td></td>
</tr>
</tbody>
</table>
2.4 findMatchingInstances and examples of 'complex' conditions

The function findMatchingInstances is used to get an array of instances which match the conditions given in the arguments.

This function is quite powerful because logical operators and, or, not and parentheses () are available and consequently you can combine several conditions. There are also operators like =, !=, startsWith, endsWith, contains and matches which give even more flexibility.

This function is overloaded. Two prototypes of the function are the following:

1. findMatchingInstances(DeviceTypeName[s], Condition[s])
2. findMatchingInstances(DeviceTypeName[s], Master, Condition[s])

where:

- DeviceTypeName[s] is a list of 1 or more device types, comma-separated e.g. "DigitalInput, DigitalOutput"

- Condition[s] is a string defining 1 or more logical conditions to be satisfied, e.g. "'#FEDeviceEnvironmentInputs:Input#' contains '$Name$'"
  
  - NOTE: the column name from the spec must be surrounded by hash symbols ('#FEDeviceEnvironmentInputs:Input#') and variables from the jython template must be surrounded by dollar symbols ('$Name$').
  
  - NOTE2: 'multiple' conditions is only possible with CPC Wizard version 1.4.0 or later (i.e. not available with CPC Wizard 1.3.1).

- Master is a string containing the name of a master in which case the function will only return those devices whose master=Master

Number 1 will return all the instances of the given device type(s) matching given conditions
Number 2 will return all the instances of the given device type(s) whose master = Master and matching the given conditions.

Note: There is also a third form of findMatchingInstances we do not talk about here. For more information see UAB API documentation:

location: TBD

Some examples:

Example with prototype 1

```python
theRawInstances.findMatchingInstances("AnalogAlarm", "("#FEDeviceAlarm:Type#"!='TS') and not('DeviceIdentification:Name#"matches\"Ana[dD].\"')")
```

1st argument: In the AnalogAlarm worksheet,

2nd argument: Get only the objects where (Alarm Type is not TS) and (Name does not begin with Anad or AnaD).
**Note:** In the matches operators you can specify a regular expression. However the $ regular expression keyword is forbidden because of Jython interpreter issues - see [UAB-349](#).

**Example with prototype 2**

```python
theRawInstances.findMatchingInstances("AnalogAlarm", "TEST_PCO1", "('#FEDeviceAlarm:Type#='TS' and '#DeviceIdentification:Name#'startsWith'PCO') or ('#FEDeviceAlarm:Type#='AL' and '#DeviceIdentification:Name#'startsWith'PCO1')")
```

1st argument: In the AnalogAlarm worksheet,

2nd argument: Get the objects whose the Master is TEST_PCO1

3rd argument: Get only the objects where (Alarm Type is TS and Name starts with PCO) or (Alarm Type is AL and Name starts with PCO1)

**Note:** For working examples see attached user templates for OnOff:

1. UCPC6_UserTemplate_Siemens_OnOff_Example.py and
2. UCPC6_UserTemplate_Schneider_OnOff_Example.py
3. Example with an OnOff dependent logic Template

NOTE: When you use 'multiple' alarm and user templates - the logic for the Alarm is created in the DL of the first object given in the alarm 'Master' list.

3.1 Presentation of the use case

We have 2 OnOff valves PV001 and PV002 located in 2 identical circuits with the same logic structure.

- PV001:
  - Open If: the master “circuit1” is On AND the pressure PT015 < 1bar
  - Close if: the master “circuit1” is Off OR the pressure PT015 > 2bar
  - A digital Alarm “PV001_TS1” is configured on the valve to provoke a temporary stop if the DI “PV001Def = 0”.
  - A digital Alarm “PV001_SI1” is configured on the valve to provoke a start Interlock if “PT012-PT015 < 0” during 5sec.
  - An Analog Alarm “PT012_FS” is configured on the valve to give a warning if “PT012 < 1.5bar” (Low) and a Full Stop if “PT012 < 1bar” (LowLow). Moreover, the LL threshold is activated only if there is no IOError on “PT012”.

- PV002:
  - Open If: the master “circuit2” is On AND the pressure PT025 < 1bar
  - Close if: the master “circuit2” is Off OR the pressure PT025 > 2bar
A digital Alarm “PV002_TS1” is configured on the valve to provoke a temporary stop if the DI “PV001Def = 0”.

A digital Alarm “PV002_SI1” is configured on the valve to provoke a start Interlock if “PT022-PT025 < 0” during 10sec.

An Analog Alarm “PT022_FS” is configured on the valve to give a warning if “PT022 < 1.5bar” (Low) and a Full Stop if “PT022 < 1bar” (LowLow). Moreover, the LL threshold is activated only if there is no IOError on “PT022”.

The spec lines for the OnOff, DA and AA are represented in Figure 8, 9 and 10. Note that the logic parameters 1 and 2 in the OnOff objects will be used to generate the logic.

In this example, the configured alarm inputs are:
- Digital Alarm input and delay of PV001_TS1 and PV002_TS1
- Digital Alarm delay of PV001_SI1 and PV002_SI1
- Analog Alarm delay, input, L and LL thresholds of PT012_FS and PT022_FS

In this example, the not configured alarm inputs are:
- Digital alarm Input of PV001_SI1 and PV002_SI1
- Analog Alarm activation of thresholds of PT012_FS and PT022_FS
The series of modifications to perform in the OnOff dependent logic to implement this logic is the following:

1- Create a PLC variable "dP_PV00X" to store intermediate data.

2- Fill-in actuator logic of the valve using the parameters.

   **NOTE:** The Logic parameters defined in the specification file are directly available in the variables LparamX.

   Lparam1 corresponds to the
   LogicDeviceDefinitions:CustomLogicParameters:Parameter1 in the spec.

   Lparam2 corresponds to the
   LogicDeviceDefinitions:CustomLogicParameters:Parameter2 in the spec.

   Etc.

   So in the present case we will use Lparam1 and Lparam2.

3- Remove the default not configured alarm inputs

4- Write your own logic for the not configured alarms (the configured alarms as the input of the temporary stop and the input of the analog alarm will be done automatically in the main part).

These modifications are represented in Figures 11 and 12 for Siemens and Schneider.

3.2 Siemens Template
Figure 11: Example of S7 OnOff user template
3.3 Schneider Template

As of RP 1.3.2 you *can* use < and > symbols in Schneider templates (Ref UCPC-1017). Note also that in Schneider, the alarm part is divided in FS, TS, SI and AL parts whereas it is by default a single block in S7 template. You can of course reorganize these parts in the user logic template according to your convenience.
Figure 12: Example of Schneider OnOff user template