TSL Conditional Clauses

Table of Contents

Introduction

When describing a tool, it is often the case that the number of its parameters, their formats, appearance of the setup dialog or a command line dynamically depend on some external conditions or on current values of some tool's parameters.

For example, some parameter A may be reasonable only if some other parameter's B value is V. Otherwise it should not be passed into the command line and, naturally, neither be input in the setup dialog.

Another sample. Many multi-lingual compilers (e.g. C/C++) contain language-dependent options. That is, when a C program is passed on input to such a compiler tool, all C++-specific parameters make no sense.

To support definition of such conditional properties, special syntactic clauses had been introduced in TSL. These clauses are of two kinds: string conditionals and structural conditionals.

Structural Conditional Clauses

Conditionals of this kind serve for conditional (de)activation of entire groups parameters and command lines.

In an XML-file, structural conditional clauses are specified with special XML tags. Their common syntax is:

```
<COND param1=”value1”
      param2=”value2”
      ...
      paramN=”valueN”>
   ...
</COND>
```

Here `COND` is one of the words `″if″`, `″if-not″`, `″if-and″`.

The semantics of the `if`-clause is: its body (the text enclosed within the matching pair of `COND` tags) makes sense if and only if at least one of the `parami` parameters' current values is literally equal to the respective `valuei` (*i*=1..N). This clause may be also thought of as `if-or`-clause.

The semantics of the `if-not`-clause is the opposite: the element makes sense if and only if all current values of `parami` parameters are not equal to their respective `valuei`.

The semantics of the `if-and`-clause is: the element makes sense if and only if all current values of `parami` parameteres are literally equal to their respective `valuei`.

We will write `param=″value″`, meaning «the current value of parameter `param` is literally equal to `value`», and `param≠″value″`, meaning «the current value of parameter `param` is not equal to `value`». Similarly, we will write `param1=param2`, meaning «the current values of parameters `param1` and `param2` are literally equal», and `param1≠param2`, meaning «the current values of parameters `param1` and `param2` are not literally equal». (Note, however, that the last pair of expressions cannot be written as conditions in the structural conditional clause).

Conditional clauses may be nested. For example,

```
<if param1=″value1″>
  <if-and param2=″value2″
          param3=″value3″>
    <if-not param4=″value4″>

      <parameter id=″ParamA″ .../>

    </if-not>
  </if-and>
</if>
```

In this case, `ParamA` will be defined only if conditions `param1=″value1″`, `param2=″value2″`, `param3=″value3″` and `param4≠″value4″` are all true.

Structural conditional clauses may be used in the following sections of a context definition:

- in the parameters sections;
- in the `input` section (setup dialog description);
- in the `output` section (command lines and command files description).

For example, defining a C/C++ compiler tool, in the respective project context we specify the parameter `UsedLanguage` of type `Enum`, which can accept values `″C″` and `″C++″`. Then language-dependent parameters may be defined as follows:

```
<if UsedLanguage=″C″>
```

```
  <parameter id=”ANSI_C_Compliant” .../>
  <parameter id=”Allow_CPP_Comments” .../>
  ...
</if>
<if UsedLanguage=”C++”>
  <parameter id=”ANSI_CPP_Compliant” .../>
  <parameter id=”Enable_RTTI” .../>
  ...
</if>
<!—Common parameters -->
...
```

It is clear that these parameters must be included in or excluded from the setup dialog of this tool context. To do that, we can use conditional clauses in the `input` section, for example:

```
<input>
  ...
  <group ...>
    ...
    <if UsedLanguage=”C”>
      ”ANSI_C_Compliant”
      ”Allow_CPP_Comments”
      ...
    </if>
    <if UsedLanguage=”C++”>
      ”ANSI_CPP_Compliant”
      ”Enable_RTTI”
    </if>
    ...
  </group>
  ...
</input>
```

Or switch on/off the entire groups of parameters:

```
<input>
  ...
  <if UsedLanguage=”C”>
    <group name=”CParams” label=”C specific options”>
      ”ANSI_C_Compliant”
      ”Allow_CPP_Comments”
      ...
    </group>
  </if>
  <if UsedLanguage=”C++”>
    <group name=”CPPParams” label=”C++ specific options”>
      ”ANSI_CPP_Compliant”
      ”Enable_RTTI”
    ...
    </group>
  </if>
  ...
</input>
```

Similarly, conditional parameters can modify the command line of the compiler, for example:

```
<output>
  <line name = "CommandLine">
    <if UsedLanguage="C">
      "%ANSI_C_Compliant"
      "%Allow_CPP_Comments"
      ...
    </if>
    <if UsedLanguage="C++">
      "%ANSI_CPP_Compliant"
      "%Enable_RTTI"
      ...
    </if>
    ...
  </line>
  ...
<output>
```

If necessary, the whole `<line>` section may be put into a conditional clause.

String Conditional Clauses

In many cases it is necessary to introduce a conditional parameter, whose properties depend on other parameters. Using structural conditionals for that may be unsuitable: they are cumbersome and may require duplication or even numerous replication of definitions. Finally, structure conditionals are not flexible enough to represent arbitrary conditions.

String conditional clauses are free of these disadvantages and are purposed for specification of conditional attributes of parameters, such as default values.

A ***string conditional clause*** is specified by either of two kinds of strings:

```
1. "?strS: str1=res1, … , strN=resN[, resDef]"
2. "?condition: resT, resF"
```

All fields `strX` and `resX` are arbitrary strings written unquoted. The bounds of these strings are determined by expected preceding and subsequent separator characters in the conditional clause pattern; therefore, the terminating separator should better be not used within a string. Besides, leading and final blanks are cut (however, internal blanks are preserved and are significant at string comparison). Thus, non-significant blanks may be inserted for readability only before and after separators. The `"?"` sign (used to recognize the string as a conditional) must be always the first character in it.

The `condition` is written as a conventional logical expression:

- elementary comparisons – pairs of kind `strA=strB` or `strA#strB`;

- a condition is build of elementary comparisons and logical connectives `″|″` (OR) and `″^″` (AND) using parentheses to modify priorities of connectives.

Within string fields `strX` and `resX`, generators `%param` may be used to substitute current values of parameters into the field. It is their usage that brings non-trivial sense to a conditional expression.

Interpretation of a conditional expression works as subsequent expansion of all parameters-generators in strings, performing a series of comparisons of the resulting strings and outputting some string as a *result* of that conditional expression.

The conditional expression of type 1 is interpreted as a conventional select-by-parameter statement. After expanding all generators, the string `strS` is subsequently compared with all *selector* strings `strk`. If comparison with *i*-th selector was *successful* (no distinctions found), the value of the respective result string `resi` becomes the result of the whole expression. If none of comparisons succeeded, but the `resDef` string is present, it becomes the result. If `resDef is` not specified, the result is an empty string.

The conditional expression of type 2 may be used in cases when the result string depends on several parameters in a complex manner. It is interpreted as follows: first, `condition` is evaluated; if it is true, the result is `resT` string, otherwise `resF`.

Some samples:

1) Expression:

```
?%MyParam: %AnotherParam=Fred, MyValue=Wilma, Barney
```

evaluates to`″Fred″` if `MyParam=AnotherParam`. Otherwise, if `MyParam=″MyValue″`, the result is`″Wilma″`, otherwise `″Barney″`.

2) Expression:

```
?(%par1 = %par2 | par3 = foo) ^ %par4 # bar: Fred, Wilma
```

evaluates to `″Fred″`, if either `par1=par2` or `par4≠″bar″`, or `par3=″foo″`, and `par4≠″bar″`. Otherwise the result is `″Wilma″`.

String contitional clauses are allowed in the following attributes of parameters: `default`, `omit`, `visible` and `readonly`.

For example, in our sample C/C++ compiler tool we may need a parameter defining an extension for input file. It can be specified as:

```
<parameter id=″SourceExtension″
           default=″%?UsedLanguage=C: c, cpp″
           .../>
```

Or as:

```
<parameter id=”SourceExtension”
           default=”%?UsedLanguage: C=c, C++=cpp”
           .../>
```

The same parameter could be specified with a structural conditional clause, but longer:

```
<if UsedLanguage=”C”>
  <parameter id=”SourceExtension”
             default=”c”
             .../>
</if>
<if UsedLanguage=”C++”>
  <parameter id=”SourceExtension”
             default=”cpp”
             .../>
</if>
```

and if any other attribute of the parameter ever needs modification, we will have to do it synchronously in both variants.