Extended Test Specification Language Format

This document describes our interpretation of the TSL language presented in Ostrand and Balcer's 1988 CACM paper ("The Category-Partition Method for Specifying and Generating Functional Tests", Thomas Ostrand and Marc Balcer, Communications of the ACM, Volume 31, Number 6, June 1988, pp 676-686). Our syntax is almost exactly like that presented in the paper; however, our interpretation of the semantics of that syntax required several choices due to lack of details in the paper.

Comments

The '#' character marks a comment until the next newline.

```
# Comment: [everything] is ignored, even # chars
# The property list below will be ignored
a choice.           # [property Skin]
```

Categories

A string ending with the ':' (colon) character is interpreted as a category.

The "Parameters:" and "Environments:" category divisions will be ignored (because they really only serve to help the tester create the specs -- categories within either division are then treated the same during test frame generation), as will any category that isn't followed by any choices.

```
Parameters:             # this one will be ignored
    Pattern size:       # this one won't
        empty.          [property Empty]
```

Choices

A string ending with the '.' (period) character is interpreted as a choice.

```
this is another choice.     [single]
                      ^
```

Constraints

The '[' character marks a constraint until the next ']' character. Constraints are applied to the choice they follow. They can't be applied to categories.

Individual constraints can't span multiple lines, but the set of constraints for a choice can.

```
simple choice.  [if thisIsAReallyLongProperty]
                    [single]
                [else]
```

```
                    [property short]
```

Constraints are case sensitive, so [ELSE], [Else], or any other variations won't work.

The three types of constraints are property lists, error/single markings, and selector expressions.

Property Lists

```
[property <name1>, <name2>, ... <nameK>]
```

Sets the properties <name1>, <name2>, ... <nameK> to true. They can later be referenced in selector expression. Property names are case sensitive.

Each choice can have several property lists, but not more than 10 properties total.

```
# this choice uses the max number of properties
a choice.      [if ABC] [property 1, 2, 3, 4]
               [else] [property 5, 6, 7, 8, 9, 10]
```

Error & Single Markings

```
[error]
[single]
```

Both markings signal that only one test frame should be generated; the meaning you give each of them is the only difference. The one test frame will be appropriately marked as an error or single.

Selector Expressions

```
[if <expression>] <error/single> <property list>
[else] <error/single> <property list>
```

The [else] is optional, and so are the <error/single> and <property list> for both [if] and [else].

If <expression> evaluates to true then the following three steps are applied to the statements after the [if]. Otherwise, the steps are applied to the statements after the [else] (if there is one).

1. If there's an error or single marking only one frame is generated for this choice. The property list is ignored and this choice isn't combined with any other choices, but its dependencies are shown (if any).
2. Otherwise this choice is selected to be combined with other choices.
3. If there's a property list its properties are set to true.

A choice can have only one selector expression (you can't nest them like [if] ... [else] [if] ...).

How Choices Are Selected

If a choice has no error/single markings or selector expression then it is always selected (it might have a property list).

```
#this choice is always selected
a good choice is this.      [property Goodness]
```

When there is an error/single marking before a selector expression the expression is ignored. This allows you to experiment with marking choices as strictly error/single without having to comment out previous constraints.

```
# only one frame is ever generated for this choice
[single] [if Random] [property RandQuoted]
```

When there is a selector expression any property lists should follow it. If there is a property list before a selector expression the property list is ignored.

```
# Long will never be set to true
[property Long] [if Unquoted] [property Zero]
```

Examples:

1. `[if NonEmpty]`
   Reads: if NonEmpty is true combine this choice with others.
2. `[if Hmmmmm] [single]`
   Reads: if Hmmmmm is true make a single frame with this choice.
3. `[if Radical] [property Cool] [else] [error]`
   Reads: if Radical is true set Cool to true and combine this choice with others,
   else make an error frame with this choice.
4. `[if NoWay] [single] [else]`
   Reads: if NoWay is true make a single frame with this choice,
   else combine this choice with others.
5. `[error]`
   Reads: make an error frame with this choice.
6. `[property IC] [single]`
   Reads: make a single frame with this choice. (the property list is ignored)
7. `[single] [if Random] [property RandQuoted]`
   Reads: make a single frame with this choice. (the selector expression is ignored)
8. `[property Oh, Yeah]`
   Reads: set Oh and Yeah to true and combine this choice with others.
9. `[property Long] [if Unquoted] [error]`
10. `[else] [property Zero]`

Reads: if Unquoted is true make an error frame with this choice,
else set Zero to true and combine this choice with others. (the first property list is ignored)

Expressions

The logical operators '&&' (and), '||' (or), and '!' (not) can be used in a selector expression. The normal order of evaluation is used (not > and > or), and the terms can be grouped using parentheses to change the order of evaluation.

Examples:

```
[if A]
[if !B]
[if A || B]
[if A && B]
[if !(A && B)]
[if A && B || C]
[if A && (B || C) && D]
[if A || !B && !C]
[if !(!A || B) && C || D]
```

The entire expression doesn't need to be enclosed in parentheses. The following example is what NOT to do.

```
[if (A && B || C)]
```