November 14, 2005

Format For GUI Definition Files

1 Lexical structure

Whitespace is ignored (except in bare strings; see below). Both kinds of C99/C++/Java comment (single-line "`//`" and multiline "`/* */`") are supported.

Apart from single-character punctuation marks, there are three different kinds of tokens in the syntax:

ID An identifier. As usual, this must be a letter or underscore followed by letters, numbers, and/or underscores. (This is a subset of the Java identifiers; in particular, Java also allows dollar signs. We reserve those for constructing unique internal identifiers.)

STRING A string, in double quotation marks. Recognized escapes:

`\"` A literal double quotation mark; does not end the string.

`\\` A literal backslash; will not escape a quotation mark.

`\`⟨*end-of-line*⟩ Skips the end of the line and all leading whitespace in the next line. Allows for long lines to be continued neatly:

```
This is a very long string, which we can continue on the next line \
and not even worry about indentation.
```

Any unrecognized escape sequence will be left alone (including the backslash) to allow the underlying language to process it.

BARESTRING In certain circumstances, the quotes around a string are optional. A bare string always starts on a non-whitespace character, and continues up to the first "`,`" or "`;`". Quotation marks are always necessary for any string that:

- contains any of "`,`", "`;`" and "`"`",
- spans more than one line, or
- begins with whitespace.

2 Grammar

1. ⟨*file*⟩ ::= ⟨*attr*⟩* [⟨*controller*⟩] ⟨*statement*⟩*

   A file is a series of statements. There can also be one controller block, which must appear before all widget statements.

2. ⟨*statement*⟩ ::= ⟨*widget*⟩ | ⟨*accum*⟩ | ⟨*attr*⟩

   There are three kinds of statement:

   Widget statement Defines one or more new widgets, either at the top level or as a child of another widget.

   Accumulation statement Adds more attributes to one or more existing widgets.

   Attribute statement Adds an attribute to a widget or widgets, or, at top level, declares an option or flag.

⟨*file*⟩ ::= ⟨*attr*⟩* [⟨*controller*⟩] ⟨*statement*⟩*

⟨*statement*⟩ ::= ⟨*widget*⟩ | ⟨*accum*⟩ | ⟨*attr*⟩

⟨*widget*⟩ ::= `ID` [⟨*specifier*⟩] ⟨*block*⟩
| `ID` ⟨*specifier*⟩ (“`,`” ⟨*specifier*⟩)+ ⟨*attrblock*⟩

⟨*specifier*⟩ ::= ⟨*declarator*⟩ | `STRING`

⟨*declarator*⟩ ::= `ID` [`STRING`]

⟨*block*⟩ ::= “`;`” | “`{`” ⟨*statement*⟩* “`}`”

⟨*accum*⟩ ::= ⟨*declarator*⟩ ⟨*block*⟩
| ⟨*declarator*⟩ (“`,`” ⟨*declarator*⟩)+ ⟨*attrblock*⟩

⟨*attrblock*⟩ ::= “`;`” | “`{`” ⟨*attr*⟩* “`}`”

⟨*attr*⟩ ::= `ID` “`:`” ⟨*attrvalue*⟩ (“`,`” ⟨*attrvalue*⟩)* “`;`”

⟨*attrvalue*⟩ ::= `STRING` | `BARESTRING`

⟨*controller*⟩ ::= “`%controller`” “`{`” ⟨*handler*⟩+ “`}`”

⟨*handler*⟩ ::= `ID ID` (“`,`” `ID`)* “`;`”

Figure 1: The full formal grammar, in extended Backus-Naur form.

3. ⟨*widget*⟩ ::= `ID` [⟨*specifier*⟩] ⟨*block*⟩
| `ID` ⟨*specifier*⟩ (“`,`” ⟨*specifier*⟩)+ ⟨*attrblock*⟩

   ⟨*specifier*⟩ ::= ⟨*declarator*⟩ | `STRING`

   ⟨*declarator*⟩ ::= `ID` [`STRING`]

   A widget statement is in identifier for the widget type, followed by zero or more specifiers separated by commas, followed by a block. If more than one specifier is given, the block is an attribute block and can therefore contain no widget statements.

   Each specifier may be a string with no identifier or a declarator, which includes an identifier and optionally a string. The string, if present, will be used as the value of the `label` attribute, exactly as if it had been declared as such within the block.

   A widget statement with no declarators is an anonymous widget statement; it creates a widget with no name, which therefore cannot be accessed directly either within the definition file or in user code.

4. ⟨*block*⟩ ::= “`;`” | “`{`” ⟨*statement*⟩* “`}`”

   ⟨*attrblock*⟩ ::= “`;`” | “`{`” ⟨*attr*⟩* “`}`”

   A block can be empty, in which case it is only a semicolon, or it's a series of statements within curly braces. (An empty pair of braces is equivalent to a bare semicolon.)

   An attribute block is a block containing only attribute statements.

5. ⟨*accum*⟩ ::= ⟨*declarator*⟩ ⟨*block*⟩
| ⟨*declarator*⟩ (“`,`” ⟨*declarator*⟩)+ ⟨*attrblock*⟩

N.B. Every declarator in an accumulation statement *must* have appeared earlier in the same block. This is the only way to distinguish an anonymous widget statement from an accumulation statement with only one declarator (without knowing what the valid widget types are called), so this restriction is actually part of the grammar, so that the grammar is not ambiguous. (This does mean the grammar is not context-free, but only in this limited instance; the parser is made only somewhat more complicated because of this.)

An accumulation statement specifies additional attributes for the widget(s) with the given identifier(s). If only one declarator is given, additional child widgets can also be specified.

6. ⟨*attr*⟩ ::= `ID` "`:`" ⟨*attrvalue*⟩ ("`,`" ⟨*attrvalue*⟩)* "`;`"

   ⟨*attrvalue*⟩ ::= `STRING` | `BARESTRING`

   Attribute statements have the form "`attribute: value, value, value;`". The attribute name is an identifier, the values are optionally-quoted strings, and the statement ends in a semicolon.

7. ⟨*controller*⟩ ::= "`%controller`" "`{`" ⟨*handler*⟩+ "`}`"

   ⟨*handler*⟩ ::= `ID ID` ("`,`" `ID`)* "`;`"

   The event handlers are declared in the controller block, which begins with the keyword `%controller`. Each handler statement is simply the name of the handler type (such as `Action` or `KeyPress`), followed by one or more comma-separated identifiers and a terminating semicolon.