CALDoc Crib Sheet

Joseph Wong

Last modified: April 9, 2007

Synopsis

Common tags	Other tags
• `@arg` *argument-name argument-description* • `@return` *return-value-description* • `@see` *context-keyword = reference (, reference)** `@see` *reference (, reference)** • `{@em` *text*`@}` • `{@strong` *text*`@}` • `{@sup` *text*`@}` • `{@sub` *text*`@}` • `{@url` *url*`@}` • `{@code` *code-text*`@}` • `{@link` *context-keyword = reference*`@}` `{@link` *reference*`@}`	• `@author` *author-name* • `@version` *version-info* • `@deprecated` *deprecation-notice* • `{@unorderedList ...@}` `{@orderedList ...@}` o `{@item` *list-item-text*`@}` • `{@summary` *summary-text*`@}`

Common tags

Block tags

@arg and @return tags

An illustrative example:

```
/**
 * @arg firstArg
 * @arg secondArg
 *         it is permissible to leave the description of an argument empty (as in the above).
 * @arg thirdArg
 *        this argument is not a declared parameter of the function,
 *        but its existence is inferred from the type signature.
 * @return
 *    {@link Prelude.True@} if the function succeeds, {@link Prelude.False@} otherwise.
 */
demo :: a -> b -> c -> Prelude.Boolean;
public demo firstArg secondArg = …
```

Note that the return value described by the `@return` tag corresponds to the evaluation of `(demo firstArg secondArg thirdArg)` and not to the partial evaluation `(demo firstArg secondArg)`, even though the third argument is not declared as a parameter.

"See also" cross-references – @see tag

In many circumstances, the short `@see` syntax is sufficient for specifying cross references. For example:

```
/**
 * @see List.map, id, "Cal.Collections.Array.map", Prelude.Nothing, Right,
 * Prelude.Either, Maybe, Prelude.Outputable, Bounded, Prelude
 */
```

If the context of a cross reference is ambiguous, then you will need to use the long `@see` syntax, e.g.:

`/** @see typeConstructor = String */` versus `/** @see module = String */`

The context keywords that can be used are: `module`, `function`, `typeClass`, `typeConstructor`, and `dataConstructor`.

Style Guideline

It is recommended that references to type constructors, type classes and modules always appear with the appropriate context keywords. Also, if one `@see` block in a comment uses a context keyword, then all `@see` blocks in the comment should do so as well. For example:

```
/**
 * @see typeClass = Ord
 * @see function = compare
 */
```

By default, the cross references are checked by the compiler to make sure that the definitions they reference do indeed exist and are found either in the current module or in its imported modules. To refer to entities found in another module that is not imported by the current module, surround the references with double quotes (e.g. `"Cal.Collections.Array.map"`) – such references will not be checked during compilation.

Inline tags

Simple formatting tags

`{@em This is italicized.@}`	Emphasized text (*italics*)
`{@strong This is bolded.@}`	Strongly emphasized text (**bold**)
`2{@sup 5@}`	Superscript, e.g. 2^5
`log{@sub 2@} 16`	Subscript, e.g. $\log_2 16$
`{@url http://www.businessobjects.com@}`	Creates a hyperlink, e.g. http://www.businessobjects.com

Code fragments and inline cross-references in CALDoc – @code and @link tags

Style Guideline

In a CALDoc comment, all CAL fragments should appear in either a `{@code}` or a `{@link}` inline tag.

For example (taken from List.map):

```
/**
 * {@code map mapFunction list@} applies the function {@code mapFunction@}
 * to each element of the list and returns the resulting list.
 *
 * @arg mapFunction a function to be applied to each element of the list.
 * @arg list the list.
 * @return the list obtained by applying {@code mapFunction@} to each element of the list.
 */
```

Whitespace and newlines are preserved in `{@code}` blocks, so a multi-line fragment will appear in the generated documentation as it is written in the comment. For example:

```
/**
 * This is a let expression in CAL:
 * {@code
 *     let
 *         f = 3.0;
 *     in
 *         f
 * @}
 */
```

Style Guideline

Names of top-level functions, class methods, type and data constructors, and type classes should be hyperlinked with the `{@link}` tag.

Like the `@see` tag, the `{@link}` tag can be used with double-quoted (unchecked) references, and the context keywords `module`, `function`, `typeClass`, `typeConstructor`, and `dataConstructor`.

For example:

```
/**
 * This is a cross reference to Prelude.Right: {@link Right@}
 *
 * This is how you would write "Maybe a": {@code {@link Maybe@} a@}.
 *
 * This is a nice example fragment:
 * {@code {@link List.map@} {@link truncate@} [2.6, -2.7] == [2, 2]@}
 *
 * To disambiguate, you would write {@link typeConstructor = String@}
 * or {@link module = String@}.
 */
```

Note that with *Maybe a*, the `{@link}` tag goes inside the `{@code}` – this is as it should be in cases like this, where the hyperlink forms a part of a larger code fragment. However, one should simply write `{@link Right@}` instead of `{@code {@link Right@}@}`.

Other tags

Block tags

@author tag

Style Guideline

It is recommended to put only one author name per `@author` block, and let the documentation generator handle the generation of commas. For example:

```
/**
 * @author Bo Ilic
 * @author Joseph Wong
 */
```

@deprecated tag

To mark that a definition is no longer recommended for use, put in a notice with a `@deprecated` block. It is helpful to include a hyperlinked cross-reference to the recommended replacement in the deprecation notice.

```
/**
 * @deprecated This functionality is no longer supported. Use
 * {@link "Cal.Collections.List.map"@} instead.
 */
```

@version tag

The version string can be an arbitrary block of text, and is not verified against any predefined syntax.

```
/** @version 37.2.1-beta2 */
```

Lists in CALDoc

CALDoc supports both ordered and unordered lists, and the nesting of lists within other lists

```
/**
 * Here's an ordered list:
 * {@orderedList
 *    {@item alpha@}
 *    {@item beta - paragraph 1
 *
 *    beta - paragraph 2
 *    @}
 * @}
 *
 * Here's an unordered list:
 * {@unorderedList {@item A@} {@item B@} @}
 */
```

{@summary} tag

The `{@summary}` tag facilitates the overriding of the default summary extraction behaviour, which uses the first sentence of the first paragraph as the summary. Use this feature when your summary needs to span more than one sentence, or appear in a location other than the start of the comment.

```
/**
 * {@summary This is a demo a multi-sentence summary.
 * This inline tag facilitates the overriding of the default
 * summary extraction behaviour (i.e. first sentence of first
 * paragraph).@}
 */
```

Escaping the '@' character

If you want to put the '`@`' character into a CALDoc comment, you may do so using the escape sequence '`\@`'. It is permissible to simply write the '`@`' character without escaping it if it is 1) not the first non-whitespace character on the line and 2) it does not follow immediately after an open or close inline tag (i.e. `{@tagName` and `@}`).

For example:

```
/**
 * \@ character - this one needs to be escaped.
 *
 * Email address: joseph.wong@businessobjects.com
 * (no escaping necessary in the above)
 */
```

Style suggestions

- Do not repeat the name of the documented entity as a title for your comment
  For example, do not do this:

```
/**
 * demoRedundantTitle
 *
 * This is a demo of a redundant title.
 */
public demoRedundantTitle = Prelude.undefined;
```