



Syntax_Tools

Copyright © 2006-2019 Ericsson AB. All Rights Reserved.
Syntax_Tools 2.2.1
November 21, 2019

Copyright © 2006-2019 Ericsson AB. All Rights Reserved.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0> Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License. Ericsson AB. All Rights Reserved..

November 21, 2019

2 Reference Manual

Syntax_Tools contains modules for handling abstract Erlang syntax trees, in a way that is compatible with the "external format" parse trees of the `STDLIB` module `erl_parse`, together with utilities for reading source files, pretty-printing syntax trees, merging and renaming modules, cleaning up obsolete constructs, and doing metaprogramming in Erlang.


```
scan_lines(Text::string()) -> [CommentLine]
```

Types:

```
CommentLine = {Line, Column, Indent, Text}  
Line = integer()  
Column = integer()  
Indent = integer()  
Text = string()
```

Extracts individual comment lines from a source code string. Returns a list of comment lines found in the text, listed in order of **decreasing** line-numbers, i.e., the last comment line in the input is first in the resulting list. `Text` is a single string, containing all characters following (but not including) the first comment-introducing `%` character on the line, up to (but not including) the line-terminating newline. For details on `Line`, `Column` and `Indent`, see *file/1*.

```
string(Text::string()) -> [Comment]
```

Types:

```
Comment = {Line, Column, Indentation, Text}  
Line = integer()  
Column = integer()  
Indentation = integer()  
Text = [string()]
```

Extracts comments from a string containing Erlang source code. Except for reading directly from a string, the behaviour is the same as for *file/1*.

See also: *file/1*.

`warning_marker(Warning::term()) -> syntaxTree()`

Creates an abstract warning marker. The result represents an occurrence of a possible problem in the source code, with an associated Erlang I/O `ErrorInfo` structure given by `Error` (see module *io(3)* for details). Warning markers are regarded as source code forms, but have no defined lexical form.

Note: this is supported only for backwards compatibility with existing parsers and tools.

See also: *eof_marker/0*, *error_marker/1*, *is_form/1*, *warning_marker_info/1*.

`warning_marker_info(Node::syntaxTree()) -> term()`

Returns the `ErrorInfo` structure of a `warning_marker` node.

See also: *warning_marker/1*.

merl_transform

Erlang module

Parse transform for merl. Enables the use of automatic metavariables and using quasi-quotes in matches and case switches. Also optimizes calls to functions in merl by partially evaluating them, turning strings to templates, etc., at compile-time.

Using `-include_lib("syntax_tools/include/merl.hrl")` enables this transform, unless the macro `MERL_NO_TRANSFORM` is defined first.

Exports

`parse_transform(Forms, Options) -> term()`

