

NAME

ctg – generate pretty class trees from C++ source files

SYNOPSIS

ctg [**-m**] [**-oname**] [**-p**] [**-rname**] [**-s**] [**-Sname**] [**-v**] name ...

DESCRIPTION

ctg reads a number of C++ source files, extracts class and struct definitions and generates a graphical representation of the resulting class hierarchy as a PostScript source. The output may be constrained to a subset of classes according to the contents of a given configuration file.

OPTIONS

If run without options, **ctg** processes all input files silently and writes the generated class tree to the standard output.

- m** Micro size output. By default **ctg** uses 80x12 boxes and font size 10 for the PostScript output which suffices for trees with depth 6. With this option 60x10 boxes and font size 8 are used. This might suffice for larger class libraries.
- oname** Output file specification. By default **ctg** writes the PostScript source to the standard. With this option the output is written to *name*.
- p** Also include privates. By default **ctg** ignores *private* sections within class definitions. With this option also private classes are included into the hierarchy.
- rname** Root class specification. Only descendants of class *name* are included into the output. This option overrides any **root** specification in a given configuration file. Without this option all classes are sons of the overall root *****.
- s** Also include structs. By default **ctg** ignores struct definitions. With this option also structs are included into the hierarchy.
- Sname** Configuration file specification. By default **ctg** tries to read the configuration from **~/.ctgrc**. This options causes **ctg** to read it from the file *name*.
- v** Verbose option. By default **ctg** processes the input files silently. With this option **ctg** generates messages about all actions and writes them to the standard output. If no output file specification is given, the PostScript source is written to the file *out.ps*.

FILES

If the **-S** option has been omitted, **ctg** tries to read the configuration from **~/.ctgrc**. Note that the argument to **-r** overrides a given **root** specification in the configuration file.

A configuration for **ctg** consists of a number of specification statements which affect the preprocessing stage and the PostScript code generation.

```
Config = { Stmt }.
Stmt   = "preprocess" string
        | "exclude" file_identifier { file_identifier }
        | "root" Node
        | "collapse" class_identifier { class_identifier }.
Node   = (class_identifier | "*") [ "(" Node { "," Node } ")" ].
```

preprocess *string*

In order to enable accurate parsing all sources are preprocessed before they are passed on **ctg**'s parser. The statement specifies a command template used for the invocation of an external preprocessor. The template *string* must contain two **%s** fill-ins where at each invocation the first one is replaced by the source file name and the second one by the preprocessor output file (extension **.i** required). Example: *preprocess "gcc -E %s >%s"*.

exclude *file_identifier* { *file_identifier* }

This statement specifies a number of header files which will be excluded from the final preprocessor output. Typically this list consists of C and C++ standard headers. The *iostream.h* header should always be specified, since it contains classes with multiple inheritance (see diagnostics).

root *Node*

The root of the generated class hierarchy is specified with this statement. Optionally *class_identifier* may be followed by a list of (direct) son nodes enclosed in brackets. In such a case only these sons are included into

CTG(1)

USER COMMANDS

CTG(1)

the output, all other sons do not appear. Each son node may in turn consist of a `class_identifier` and a list of son nodes. The special class node "*" is regarded as the father of all class and struct definitions.

collapse *class_identifier* { *class_identifier* }

This statement specifies a number of class nodes whose sons will not be included in the generated class tree.

EXAMPLE

```
$
preprocess      "gcc      -E      cat      my_ctgrc
exclude        -I/home/gup/rk/misc/include  %s      >%s"
iostream.h iomanip.h fstream.h
assert.h ctype.h errno.h float.h limits.h locale.h math.h
setjmp.h signal.h stdarg.h stddef.h stdio.h stdlib.h
string.h time.h
sys/types.h      sys/param.h      *      sys/time.h      unistd.h      pwd.h
root              TObject      (FuzzyParser,  ClassTree,  Scanner)
                  (Preprocessor,
                  Scanner)

$ ctg -o ctg.ps -S my_ctgrc *.cc
```

DIAGNOSTICS

Error messages generated by the semantic actions of the fuzzy parser:

<i>duplicate</i>		<i>class</i>		<i>definition</i>
	two class definitions with the same identifier			
<i>undefined</i>		<i>base</i>		<i>class</i>
	definition of a derived class where the ancestor is unknown			
<i>multiple</i>		<i>inheritance</i>	<i>not</i>	<i>supported</i>
	class definition with more than one base class (reported for iostream.h classes)			

Error messages generated by the PostScript code generator:

<i>tree</i>	<i>width</i>	<i>exceeds</i>	<i>n</i>
	tree width exceeds paper size		
<i>tree</i>	<i>height</i>	<i>exceeds</i>	<i>n</i>
	tree height exceeds paper size		

Fatal errors that should not occur under normal conditions:

```
file file, line line: unhandled exception: code - exiting
code = -2          out of memory
code = -200       scanner could not open source file
code = -201       hash table error
```

BUGS

ctg does not support multiple inheritance, so only *true* trees are generated.

AUTHOR

Rainer Koppler (rk@gup.uni-linz.ac.at), v1.0, 21/02/95