ref - Display a C function header

SYNOPSIS

ref [*options*]... [*restrictions*]...

DESCRIPTION

This page describes the *ref* program that is distributed with **elvis(1)**.

ref quickly locates and displays the header of a function. To do this, *ref* looks in the "tags" file for the line that describes the function, and then scans the source file for the function. When it locates the function, it displays an introductory comment (if there is one), the function's declaration, and the declarations of all arguments.

OPTIONS

-t Output tag info, instead of the function header. The tag info consists of the three standard fields from each tag. The fields will be separated by tab characters, just like records from the traditional tags file format. Consequently, you can use "ref -t -a >oldtags" to convert a new-style tags file back to an old-style tags file.

-v Output verbose tag info, instead of the function header. The verbose tag info shows the names and values of all attributes for each matching tag. Each name/value pair is shown on a separate line. It also shows the "match" factor that is used for sorting tags which have the same tagname.

-h Output HTML browser info, instead of the function header. This is an HTML table with hypertext links into the source files where the tags are defined. You can use Netscape or another browser to use this, but they won't move the cursor to the correct line within the source file; only *elvis* knows how to do that.

This resembles *elvis*' **:browse** command.

-c Don't output introductory comments before each tag definition line.

-d Don't output other lines of the definition. The line where the tag is defined is shown but any preceding or following lines which are part of the definition will be omitted.

-a List all matching tags. (Without this option, *ref* would only output the single most likely tag.) would stop searching after processing the first tags file which contained any tags which met the restrictions.

-p *tagpath*
List of directories or tags files to search through. By default, *ref* would use the value from the TAGPATH environment variable or a hardcoded default value for each operating system.

-l *taglength*
Only check the first *taglength* characters of tag names. The default behavior is to check all characters.

RESTRICTIONS AND HINTS

Other than options, any argument on *ref*'s command line is interpreted as a restriction or sorting hint. Elvis parses all of the restrictions and sorting hints, and then scans the tags files (i.e., every file listed in the tag path, or a file named "tags" in every directory listed in the tag path). Tags which fail to meet any restriction are discarded.

Other tags are inserted into a list. The list is sorted primarily by each tag's tagname. If multiple tags have the same overloaded name, then those tags will be sorted according to the sorting hints. In the absence of hints, the tags will be added in the same order in which they appear in the tags file.

The restrictions can be given in any of the following forms:

name:*value*[,*value*...]
: Reject tags which have an attribute named *name*, but that attribute's value isn't in the list of acceptable values. E.g., "class:Foo" rejects tags from a different class, but accepts tags which have no class.

name:**=***value*[,*value*...]
: Reject tags which have an attribute named *name*, but that attribute's value isn't in the list of acceptable values. Also reject tags which don't have an attribute named *name*. E.g., "class:=Foo" only accepts tags which have class "Foo".

name:**/***value*[,*value*...]
: Like "name:value" except that the **tagaddress** field is required to contain *value* as a substring. So "class:/Foo" would find tags in class "Foo" PLUS

value[,*value*...]
: Short for **tagname**:*value*[,*value*...]

The sorting hints follow a similar form:

name:**+***value*[,*value*...]
: Pretend that recent successful searches had attributes named "name" with the given values. This causes any similar tags in the new search to appear near the top of the list.

name:**-***value*[,*value*...]
: Pretend that recent failed searches had attribute named "name" with the given values. This causes any similar tags in the new search to appear near the bottom of the list.

A null value string matches anything. So "struct:=" would accept any tag with a "struct" attribute, and reject those without it. This would be handy when you're trying to do tag lookup for a word which follows a '.' character - you know it is a field name, but you don't know from which struct type.

Note that if you invoke *ref* without giving any restrictions, then **all** tags will match and will (if invoked with the **-a** flag) be output.

A REAL-WORLD EXAMPLE

While converting some code from K&R C to ANSI C, I needed to generate extern declarations for all the functions. To find the global function headers, I used the command...

```
ref -a kind:f file:dummy
```

The "-a" causes *ref* to output all headers, instead of just the first one that it finds. "kind:f" causes it to exclude any non-functions. "file:dummy" is tricky -- it causes *ref* to exclude static tags from all files except "dummy", and since there were no C functions defined in any file named "dummy", all statics were excluded. I only got globals.

Once I had a list of all global functions, I still had to do some editing to convert them into ANSI declarations (*ref* couldn't help me there) but at least I could be confident that my list of functions was complete and accurate.

For each source file, I also needed to find the static functions defined there, so for each "file.c" I used the

This is very similar to the earlier command.
The main difference is that we're using "file:="
(with an '=', to exclude globals)
and a real file name (instead of "dummy") so we do include the st
atic

tags from that particular file.

INTERACTION WITH ELVIS

ref is used by *elvis'* shift-K command. If the cursor is located on a word such as "splat", in the file "foo.c", then *elvis* will invoke *ref* with the command "ref splat file:foo.c".

TAGS

A tag is a collection of attributes. Each attribute has a name and a value. Every tag has attributes with the following names:

tagname
The name of the tag; usually the same as the function (or whatever) that the tag refers to.

tagfile
The name of your source code file, in which the tag's definition occurred.

tagaddress
Either a line number, or a "nomagic" regular expression, which allows *elvis* or *ref* to locate the tag's definition within your source file.

In addition, any tag can have additional, optional attributes. These extra tags are meant to serve as hints, describing the contexts in which the tagname is permitted to occur in your source code. The list of additional attribute names is not preset; any tags file can use whatever seem appropriate. The following are typical:

kind This value is a single letter indicating the lexical type of the tag. It can be "f" for functions, "v" for variables, and so on.

file If the tag can only be used within a single source file, then this should be the name of that file. E.g., in C, a "static" function can only be used in the function in which it is defined, so if a function is static then its tag will usually have a **file** attribute, and its value will be the same as that of its **tagfile** attribute.

function

struct For fields of a struct or union. The value is the name of the struct or union. If it has no name (not even a typedef) then "struct=struct" is better than nothing.

enum For values in an enum data type. The value is the name of the enum type. If it has no name (not even a typedef) then "enum=enum" is better than nothing.

class Member functions of a class in C++ could use this to identify which class they're in. The class name itself, however, is global so it doesn't have a

class attribute.

scope Intended mostly for class member functions. It will usually be "public" or "private", so users can restrict tag searches to only public members.

arity For functions. Its value is the number of arguments.

Currently, the hacked-up version of **ctags(1)** (sometimes installed as **elvtags(1)**) included with elvis will only generate **kind**, **file**, and **class** hints, and it doesn't do a very good job on **class** hints.

THE TAGS FILE

The tags file is a text file, in which each line describes a single tag. Each line is divided into fields, delimited by tab characters.

The first 3 fields are implicitly defined to be the values of the tagname, tagfile, and tagaddress attributes, in that order. Note that this is identical to the traditional format of the tags file.

If there are other fields, then semicolon-doublequote will be appended to the tagaddress field; vi ignores anything after that, so the extra fields won't interfere with vi's ability to perform tag searches. Other editors such as *elvis* and *vim* use the extra fields though.

The extra fields are required to have the format "<tab>name:value". I.e., a ':' is required, and everything before the ':' is used as an attribute name, and everything after it is used as this tag's value for that attribute. There are two exceptions:

* If an extra field lacks a colon, then the field is assumed to be the value of an attribute named "kind". (Some versions of ctags generate a single- all tags have this field, omitting "kind:" significantly reduces the size of the tags file, and the time needed to search it.

* Static tags are usually marked with "file:", with no file name after the ":". In this case the file name is understood to be identical to the "tagfile" field. This does more than just reduce the size of the tags file -- "tagfile" values are relative to

the directory containing the tags file, and this rule offers a way to make "file" values be relative, too.

Different tags may have differently named hints. Since each hint includes an explicit name with each value, they can appear in any order, and you can omit any which don't apply to a given tag.

Ref and *elvis* store attribute names are stored in a fixed-size array, which is shared among all tags from a given file. Consequently, the number of distinct attribute names within a tags file is limited. As currently configured, that limit is 10 names - the 3 standard ones plus up to 7 other names for hints.

THE REFS FILE

When *ref* has found a tag entry and is searching for the source of that tag, if it can't read the original source file then it will try to read a file named "refs". The "refs" file should contain a copy of all source code, with the bodies of functions replaced by "{}". Elvis' version of **ctags(1)** can generate a "refs" file.

FILES

The following files can be found in any directory named in the tagpath.

tags List of function names and their locations, generated by *ctags.*

refs Function headers extracted from source files (optional).

ENVIRONMENT

TAGPATH
List of directories or files to be searched. In the case of directories, *ref* looks for a file named "tags" in that directory. The elements in the list are separated by either colons (for Unix) or semicolons (for most other operating systems). For each operating system, *ref* has a built-in default which is probably adequate.

You might want to generate a "tags" file for the directory that contains the source code for standard C library on your system. This will allow *ref* to serve as a quick reference for any library function in addition to your project's functions.

If licensing restrictions prevent you from making the library source readable by everybody, then you can have elvis' version of *ctags* generate a "refs" file, and make "refs" readable by everybody. If your system doesn't come with the library source code, then perhaps you can produce something workable from the **lint(1)** libraries.

SEE ALSO

elvis(1), ctags(1), lint(1)

Note that on some systems, **ctags(1)** is installed as *elvtags(1)***.**

AUTHOR

Steve Kirkendall
kirkenda@cs.pdx.edu
