14. Tags

Tags provide a fast way to locate specific points in a collection of files. This is done by storing a list of tag names, and their corresponding locations, in a file named "tags". This is particularly handy for programmers who are working on large projects.

Traditionally, each tag has three attributes: its name (generally the name of a function, or some other symbol from your program), the name of the source code file in which that function is defined, and the address of its line within that file. The tag name is used for selecting a particular tag.

Elvis supports more a more sophisticated model. The extensions are intended to allow elvis to handle C++, and similar languages, which allow different functions to have the same name. Since tag names are derived from function names, the tag name alone isn't sufficient to select a single tag. Elvis permits tags to have other attributes, which help it select the correct tag.

All of this is described in more detail below, in the following sections:

14.1 The tags file

Tags are stored in a file named "tags". It is a plain ASCII text file. Each line of the file contains the attributes of a single tag. A tab character is used to delimit the attributes. The traditional tags file contains three attributes for each tag.

The first attribute is the tag's name. It is typically the name of a function, variable, or data type -- a name that you could guess by looking at the source code of your project. Traditionally, this attribute has been the sole means for selecting a tag, so tag names should ideally be unique. The lines of the tags file are sorted by this attribute.

The second attribute is the name of source code file in which the corresponding function (or whatever) is defined. If it isn't an absolute file name (relative to the root directory) then it should be relative to the directory where the tags file resides -- which isn't necessarily the current working directory.

The third attribute is the address of the line within that file,

where the function (or whatever) is defined. This address can either be a line number, or a nomagic style of regular expression. If it is a regular expression, it must be bound by '/' or '?' characters, and it may contain tab characters. Typically, the entire source line is encoded as a regular expression by inserting "/^" onto the front, appending "$/" onto the end, and inserting a backslash character before each / or \ character within the line.

Elvis actually supports a superset of this format (by permitting extra attributes) but we'll start with the basics.

14.2 Creating a tags file

Usually the tags file is created automatically by a program such as ctags. It reads a collection of C or C++ source files, and generates tags for each global function. It can also generate tags for global types and variables, or for static instances of any of these.

You will usually invoke *ctags* on all source files in the current directory via a command similar to this:

```
ctags *.c *.h
```

The *ctags* program can also generate other types of output. Be sure to look at its manual page to see the options.

14.3 Reading the tags file

Tags exist mostly for use with elvis, but for the sake of simplicity we'll start with the ref program.

ref selects tags just like elvis, and then displays information about them. The simplest way to use it is to pass it the name of the tag you're interested in. The following example would display the definition of the "main" function:

```
ref main
```

There are some other options. One of the most useful is **-a** which instructs *ref* to display all selected tags. (Without **-a** it just displays one of the selected tags.) For example, if your current directory contains many programs, each with its own "main" function, then this would display the headers for all of them:

```
ref -a main
```

You can also use *ref* to generate an HTML document listing all tags, or just the ones that match some criteria. Here's an example which lists all tags as an HTML document:

```
ref -ha >tags.html
```

ref uses the same syntax for restrictions and sorting hints as elvis' :tag command. This syntax will be described later. You should

check the manual page for a list of options.

14.4 Using tags with elvis

When starting elvis, you can use the **-t***tagname* flag to start with the cursor at the definition point of a given function in your program's source code. It automatically performs the following steps:

1) Scan the tags file for a tag named *tagname*.
2) Load the file indicated by the tag's second attribute.
3) Search for the line indicated by the tag's third attribute.
4) Within that line, search for the tag name.
5) Move the cursor there.

Once elvis is running, there are many commands available which deal with tags. The most essential is :tag *tagname*. It does all the same steps as the **-t***tagname* command-line flag, plus it saves the cursor's original position on a stack. Later, you can use :pop to bring the cursor back to its original position.

When elvis is in visual command mode, you can move the cursor onto a word and hit ^] to perform a :tag search on that current word, or ^T to perform a :pop command.

If you have a mouse, then you can use the left button to double-click on a word in the text, to have elvis perform a :tag search on that word. Double-clicking the right button anywhere in the text will perform a :pop command.

The uppercase K command runs program on the word at the cursor position. The program is chosen by setting the keywordprg option. By default, it runs the ref program, so the word's definition is displayed temporarily at the bottom of the screen.

14.5 The TAGPATH

You can have tags files in several directories, and configure ref and elvis to search the appropriate ones by setting the TAGPATH environment variable. The value of TAGPATH is a list of directories or tags files, delimited by either a ':' character (for UNIX) or a ';' character (for most other operating systems, including Microsoft's).

In a typical large project, you will have some directories which contain library functions, and some which contain the code for specific programs. With this arrangement, you would set TAGPATH to search the current directory followed by each of the library directories. Something like this...

```
setenv TAGPATH=tags:/usr/src/libproj/tags:/usr/src/libio/tags
```

The exact syntax depends on your command interpreter. And of course the exact directory names will depend on your project.

When your current directory is one which contains the source code for some program, and you do a search for (as an example) "showitem", elvis would look for it first in that program's tags file, and if it isn't found there then it'll look in each library's tags files until it does find it. The ref program searches the same way.

Actually, elvis uses an option named tags to store the search path. The default value of that option is taken from the TAGPATH environment variable, though. If you don't set TAGPATH (or the tags option), then elvis will search only in the current directory.

The default path for ref is a little more sophisticated. That's because ref is intended to be general reference utility for all library functions, while elvis' tags facility is mostly intended for navigating through the source code of a single program.

Note to system administrators: ref can be *so* handy that I suggest you make a tags file for the functions in your system's standard libraries. If licensing restrictions prevent you from making the library source code available to all users, then you should use ctags -r to generate a "refs" file. If you don't have access to the library source code yourself, then perhaps you can make something useful from the lint libraries.

14.6 Enhanced tags

The C++ programming language supports "overloading," which means that different functions can have the same name. Since tag names are derived from function names, different tags will have the same name. This creates a problem because the tag name has traditionally been the only way to select a tag, so you could easily get the wrong one. Elvis' implementation of tags has some extra features to solve this problem.

There are two tactics for solving the problem. The first tactic is to be more selective; i.e., use information other than just the tag name to select tags. This definitely helps, but it is an absolute impossibility to resolve all such ambiguities prior to run-time, so we also need a second tactic: collect all possible tags into a list, and use heuristics or explicit hints from the user to sort the list so the most likely alternative is tried first, the second most likely if the first was rejected, and so on down the list. Elvis uses both tactics.

In the tags file, elvis permits tags to have extra attributes. Each attribute has a name and a value. The first three fields are named **tagname**, **tagfile** and **tagaddress**. Those names are implicit; the names don't appear in the tags file, only the values do.

If a tag has any extra attributes, they will be appended to the tag line. In order to allow the original vi/ex to read tags files which have additional attributes, a semicolon-doublequote character pair is appended to the tagaddress, before the first extra attribute. Due to an undocumented quirk of the original vi/ex, this will cause

vi/ex to ignore the remainder of the line. The extra attributes will not adversely affect the behavior of the original vi/ex.

The extra attributes have explicit names. In the tags file, the extra attributes are generally given in the form **<TAB>***name*:*value*. Different tags may have different extra attributes; many will have no extra attributes at all. The attributes may appear in a different sequence for each tag.

In a single tags file, elvis supports up to 10 distinct attribute names -- the 3 implicit names for the standard fields, plus up to 7 explicit names for extra attributes. (This is a limitation of elvis, not the enhanced tag format.)

The name can be any series of letters or digits. Lowercase letters are preferred.

The value can contain any character except NUL. Any backslash, tab, or newline characters should be stored as ****, **\t**, or **\n**, respectively.

If an extra attribute has a value but no name or colon, then the name is understood to be "kind".

The extra attributes are intended to describe the contexts in which the corresponding program symbol can appear. Typically the name is a type of lexical scope, and the value is the name of that scope; e.g., "function:init" for a tag which is only defined inside the init() function. Elvis can use these as hints to figure out which tags might make sense in the current context, and ignore those that don't. Although the extra attributes have no preset names, the following names are recommended:

kind
: The value is a single letter which indicates the lexical type of the tag. It can be "f" for a function, "v" for a variable, and so on.

  Note that since the default attribute name is **kind**, a solitary letter can denote the tag's type (e.g, **"f"** for a function).

file
: For tags which are "static", i.e., local to the file. The value should be the name of the file.

  If the value is given as an empty string (just **"file:"**), then it is understood to be the same as the **tagfile** field; this special case was added partly for the sake of compactness, and partly to provide an easy way handle tags files that aren't in the current directory. The value of the **tagfile** field always relative to the directory in which the tags file itself resides.

function
: For local variables. The value is the name of function in which they're defined.

struct
For fields in a struct. The value is the name of the struc. 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
For member functions and variables. The value is the name of the class.

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

arity
For functions. The number of arguments.

The ctags program has been hacked slightly to support some of these, but not all. Its new **-h** flag enables generation of the extra hint attributes; if you invoke **ctags** without any flags, then **-h** is one of the flags that it uses by default. For example, the usual command for generating tags for all source files in the current directory is...

```
ctags *.c *.cpp *.h
```

The current hacked-up ctags distributed with elvis will only generate **file** and **class** hints, and even **class** isn't as effective as one might hope.

Some pseudo-tags may be inserted at the top of the tags file, to describe the characteristics of that particular tags file. These tags all begin with a "!_" so that even if the tags are sorted, the pseudo-tags will always appear at the top of the file. The pseudo-tags all use the old tags format, so they can be parsed (and then ignored) by older tag reading programs.

```
!_TAG_FILE_FORMAT       2       /supported features/
!_TAG_FILE_SORTED       1       /0=unsorted, 1=sorted/
```

The !_TAG_FILE_FORMAT pseudo-tag's **tagfile** field is 2 for new-style tags, or 1 for old-style tags. The !_TAG_FILE_SORTED pseudo-tag's **tagfile** field is 1 if sorted, or 0 if unsorted. The **tagaddress** field is used simply as a comment in both tags. If these tags are missing from a tags file, then the file is assumed to be in the new format (which is still backwards compatible with the old format), and sorted. If a tags file is unsorted then it *must* contain a !_TAG_FILE_SORTED field indicating that.

These may be followed by more pseudo-tags describing the ctags program itself. Elvis' version of ctags produces the following

information:

```
!_TAG_PROGRAM_AUTHOR    Steve Kirkendall        /kirkenda@cs.pdx.edu/
!_TAG_PROGRAM_NAME      Elvis Ctags     //
!_TAG_PROGRAM_URL       ftp://ftp.cs.pdx.edu/pub/elvis/README.html  //
!_TAG_PROGRAM_VERSION   2.1     //
```

The new tags file format also addresses another limitation of the old format: the old format allows fields to be delimited with any whitespace. This is a problem because space characters are becoming more common in file names these days, so we occasionally need to put spaces into the **tagfile** field. To support this, the new format dictates that fields must be delimited by a single tab character, *not spaces*. This shouldn't cause any backward compatibility problems because traditionally ctags has always used tab as the delimiter.

Also, the interpretation of the **tagaddress** field has been refined. Traditionally, it has been defined as either a line number or a nomagic regular expression, but it has actually been implemented in vi/ex to support *any* ex command there. Supporting any command could produce a security hole, so the new format only supports addresses. It supports more complex addresses though, because they can be useful in some circumstances. For example, the tag line for a "val" field in a struct named "item_s" could look like...

```
val     file.h  /^struct item_s {$/;/^  int val;$/   struct:item_s
```

... which would allow the editor to skip past any "int val;" definitions in other structs, to find the correct "int val;" in the item_s struct.

This form of tags file is also supported by Darren Hiebert's Exuberant ctags and by Vim, in addition to elvis.

If you ever need to convert a new-style tags file back to the old style, you can do so via the ref utility. Run it like this:

```
ref -ta >oldtags
```

14.5 Restrictions and hints

The syntax of the :tag command has been extended. Previously you could only supply a single **tagname** value to search for. Now you can supply multiple acceptable values for any attribute, and control what happens when a given tag lacks a given attribute.

The arguments of the :tag command are now whitespace-delimited expressions of the following forms, to define a set of restrictions that possible tags must meet to be selected:

name:*value*
: Reject tags which have an attribute named "name", but that attribute's value isn't in the list of acceptable values. E.g.,

"file:foo.c" accepts global tags, or tags which are static to the file "foo.c", but rejects tags which are static to other files.

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

name:**/***value*
Like *name*:*value* except that when a tag has no attribute named *name* then the **tagaddress** attribute's value is required to contain *value* as a substring. "class:/Foo" would find tags in class "Foo" PLUS global tags whose address mentions "Foo" -- probably friends of the Foo class.

value
Short for **tagname**:*value*.

The parser also allows you to add some sorting hints to the command line. These hints are added to the history that elvis uses to guess which overloaded tag to list first.

name:**+***value*
If a tag has an attribute with the given name and value, then cause it to appear near the beginning of the sorted list. I.e., tags with this name and value are more likely to be the intended tag, but you can't be certain.

name:**-***value*
If a tag has an attribute with the given name and value, then cause it to appear near the end of the sorted list. I.e., tags with this name and value are less likely to be the intended tag, but you can't be certain.

All of these restriction expressions and the sorting hint expressions allow you to give multiple acceptable values. You can either give each value in a separate expression, or give a comma-delimited list of values to a single expression.

A nul 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 which struct type.

The :tag command automatically adds a **file**:*filename* restriction (where *filename* is the name of the file being edited in the current window) to any tag search you request. This causes it to ignore tags which are static to other files. The :browse command doesn't do that. See the Browsing section, below.

14.6 History

The sorting hints are persistent. They aren't forgotten immediately after a tag search; a hint from one search will influence the sorting order for following searches. The degree of influence is weighted, so more recent hints will have more influence than older hints. Eventually, each hint's weighting factor drops to zero, and the hint is forgotten only then. The history uses two lists of name/value pairs: one for storing recent successes, and one for recent failures.

While searching for a tag, elvis builds a list of tags which matched the restrictions. That list is sorted primarily by the **tagname** attribute's value, but when multiple tags have the same name, elvis looks for the attributes of those tags in the lists of successes and failures, and uses the weights of any matches to compute the likelyhood that a particular tag is the one that the user really wants. The more likely tags are inserted into the list before any less likely tags with the same name.

Expressions of the form *name*:+*value* add a name/value pair to the success list, and expressions of the form *name*:-*value* add a name/value to the failure list. Name/value pairs are also added automatically in the following circumstances:

* If you perform a tag search on the same name twice in a row, then elvis assumes you're rejecting the first tag that it found. The attributes of that tag are added to the failure list.
* If you perform a tag search on a different name, then elvis assumes that the previous tag must have been the right one, so its attributes are added to the success list.

It should be stressed that the tag history has no effect on which tags are selected from the tags file. It only affects the order in which they're presented, if more than one tag meets your restrictions.

14.7 Browsing

The result of any tag search is always a list of matching tags. The :tag command keeps this list hidden, and moves the cursor to the single most likely member of that list. This is not always the best way to select a tag.

Elvis has a :browse command which performs a tag search, and then builds an HTML document from the list. The document shows all tags which matched your search criteria; the current window will then switch to this document. There is also a :sbrowse command which displays the same document in a new window.

The arguments to :browse differ from :tag in the following ways:

* :browse does **not** automatically add any restrictions. (:tag adds **file**:*filename* to each search.)
* If you invoke :browse with no arguments, then it will assume you wanted **tagfile**:*filename*, where *filename* is the name of the file being edited in the current window.
* If you invoke :browse with a single argument, elvis first tries to interpret it as a restriction or sorting hint in the normal

way. But that search yields no tags, elvis may retry the search using your argument as a file name (**tagfile:***argument*), or as a class name (**class:/***argument*).

By default, :browse only searches through the "tags" file in the current directory. When invoked as :browse! (with a "!" suffix) it collects matching tags from all "tags" files as specified by the tags option.

Here are some examples of :browse commands.

:browse term
Show all tags named "term"

:browse
Show all tags defined in the current file.

:browse foo.c
Show all tags defined in the file "foo.c".

:browse tagname:=
Show all tags which have a **tagname** attribute. Since all tags have a **tagname** attribute, this shows every tag in the tags file.

:browse class:/DbItem
Show all tags in the DbItem class, or friend functions of that class. It may also include some non-friend functions which merely use DbItem, but there's no easy way to avoid that.

:browse DbItem
If there is a tag named DbItem, then show it. Otherwise this is the same as :browse class:/DbItem

Each matching tag in the generated document has a hypertext link to the point in your source where the corresponding symbol is defined. By following the hypertext link, you can go directly to the appropriate point in your source code. As usual, the tag stack can be used to :pop back to the same browser document, from which you may then proceed to a different tag, or :pop back one more level to wherever the cursor was located before you gave the :browse command.

If you wish, you can define your own format for the browser document. Elvis searches through the elvispath for a file named "elvis.bro". If found, then blank lines in it will be used to delimit it into three sections:

* Everything before the first blank line is the header. It is copied into the start of each browser document. $1 is replaced by the command line arguments, and $2 is replaced by the number of matching tags found. This is a straight-forward text substitution, not an evaluation like the following section...
* Everything between the first blank line and the last blank line is repeated for each tag. For each tag, it is evaluated using the simpler syntax, with $1 being replaced by the **tagname**, $2 by the **tagfile**, $3 by the line text extracted from the **tagaddress**, and $4 by a URL combining the **tagfile** and **tagaddress** attributes. You can also use parentheses to enclose more complex

expressions.
* Everything after the last blank line is the trailer. It it copied into the browser document literally.

The ref -ha *restrictions...* program generates a similar HTML document. It always interprets its arguments as restrictions, and the format of the HTML document can't be reconfigured. Those are the only differences.

14.8 The tagprg option

As an alternative to elvis' "restrictions" method for finding tags, you can set the tagprg option to a shell command line which locates the tags.

When you give a :tag command, elvis evaluates the tagprg option's value using the simpler expression syntax. Any instance of **$1** in the value will be replaced with the command-line arguments. Also, any text inside parentheses will be evaluated; this gives you a way to access other options' values, so you can do things like pass the value of the tags option to the program so it knows which tags files to search through.

The resulting string is then executed, and its output is parsed as though it was a tags file. All of the tags that it outputs are considered to be matches, since using tagprg disables the use of restrictions.

Elvis builds a list of the matches, and sorts them using the same history mechanism that is uses with restrictions. However, the "name:+value" and "name:-value" sorting hints are not detected in the arguments.

Once the list has been built, elvis moves the cursor to the first match. You can step through all matches in the list by hitting ^] or by giving the :ta command with no arguments, as usual.

Note that the program's output should be in the standard tags file format. At a minimum, this means "tagname TAB filename TAB address". If you want to use a function searching program that uses a different format, you'll need to pipe its output through a custom-made filter that converts its output to the standard tags format.

One common technique is to use the :local command in an alias, to set tagprg temporarily for a single search. The :text alias in lib/elvis.ali is an example of this.