FetchRefs documentation

COLLABORATORS			
	TITLE : FetchRefs documentation		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 9, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

FetchRefs documentation

1.1 FetchRefs documentation

FetchRefs 1.1

A feature packed utility that provides you with the most
comfortable access to your AutoDoc and include file references

For that extra touch, first read this document in its entirety.
Then re-read selected parts little by little, as you forget them.

1.2 Introduction

To get one important thing straight right away: by a 'reference' I mean a part of an AutoDoc describing a particular function or a part of an include file, describing a structure or constant. If you do not know about AutoDocs or include files, this is probably not the tool you would most likely use!

Should you know about AutoDocs and include files you will also know that they contain very essential information. Unfortunately it is very time consuming to look something up in these files. FetchRefs is a tool made to minimize the time wasted by looking things up.

FetchRefs works by reading the word your cursor is currently at in your editor. With the help of an index file it figures out exactly where this word is documented. This specific part of an AutoDoc/include file is then loaded and showed by the editor.

An example: When FetchRefs is installed, you can place the cursor at the 'T' in this word: "Text" and press your defined FetchRefs-key. Shortly after a window will pop up. This window will contain the AutoDoc for Text (from the graphics.library). The Text AutoDoc will, no doubt, mention the structure RastPort and you can then press the FetchRefs-key again and a window with the include file where RastPort is defined will pop up - with the cursor placed at the top of exactly the RastPort definition! You can continue this way until you have all the information you need.

As FetchRefs is operated through ARexx it is very flexible; for example, if you do not want to keep FetchRefs (and more memory consuming: its index table) resident all the time, you can simply make an ARexx script that just activates FetchRefs when it is needed. The disadvantage to this is that you will need to know at least a little ARexx if you are not satisfied with the standard setup.

1.3 Special features

Though many other programs help you to get a hand on the same kind of references as FetchRefs does, FetchRefs has some features which make it the absolute winner - in my eyes (but then again, of course I have not seen all the AutoDoc readers which are available. Moreover, I may be a little biassed).

The following feature list should outline some of the most obvious advantages that FetchRefs has (some of these features may apply to other similar products, but no other has it all :-).

- FetchRefs fits into every environment as long as the editor has a decent ARexx port - which an editor of today HAS. Therefore you will not have to learn to operate a completely new program - you just need to know what key/menu you attach FetchRefs to.

- FetchRefs will generate references to AutoDoc files and to include files for the following languages: C, Assembler and E.

- FetchRefs knows of the popular suffixes 'Tags', 'TagList' and 'A' for different interfaces to the same function. This ensures for example that you can get a reference to 'System' even though the function is documented as 'SystemTagList' in the AutoDoc.

- FetchRefs will pop up a window when it finds out that a requested reference exist more than once. In this window you can double click at the file you want to load the reference from. This is extremly handy for things like OpenDevice() which is documented in the AutoDocs for every device.

- FetchRefs can - by option - consider each file in its index file as a reference itself. The effect is that you can follow a thread even if the

'SEE ALSO' paragraph reads something like <dos/dostags.h>. This will simply load the entire file.

· If you are uncertain as to what you are actually searching for, FetchRefs supports wild cards as well.

· FetchRefs's index files are generated by a powerful index file generator which sports both command line options (handy for script usage) and a font and size sensitive GUI. Any type of reference may be turned on or off during index file generation to customize the index file completely.

· FetchRefs itself supports execution from both Workbench and Shell. Thus you can put it in either s:user-startup or WBStartup, as you please.

· FetchRefs comes with ready-to-use ARexx scripts for several editors: AmokEd, CygnusEd, DME, GoldED and TurboText. Adapting one of them to another editor should be relatively easy if you know some ARexx. An additional script lets you look up references from the Shell.

· FetchRefs is free. Complete C source is available for free, too.

If you do not agree that FetchRefs is the best utility to get access to references, please tell me why - and I shall try to make it better.

1.4 Useful information

FetchRefs has very straight forward distribution conditions:

- SPREAD IT
- DO NOT CHANGE ANYTHING
- KEEP ALL FILES TOGETHER
- ADD NOTHING BUT .DISPLAYME FILES TO THE DISTRIBUTION
- DO NOT CHARGE ANYTHING
- IF YOU USE IT, TELL ME SO - IF NOT, TELL ME WHY

1.5 Registration

I would like everyone using FetchRefs to send me a post ↩
card or a
NetMail/e-mail (address: see
author
) telling me so.

Donations of any kind are gladly accepted, but are not required for registering. I will not think bad of you if you tell me that you are using FetchRefs without paying for it; actually it will make me much more happy than not knowing at all that you use it.

Too few people seem to appreciate free software...

1.6 Updates

The most recently released version of FetchRefs should always be available for free freq at Jørgen Valentiners BBS, 'The Amiga Zone', as the magic name FETCHREFS.

To call The Amiga Zone: Line #1 +45 32 52 42 94 (HST/V32b)
Line #2 +45 32 50 87 30 (V32b)
Line #3 +45 32 46 13 40 (ISDNC)

For freq'ing, the FidoNet address is 2:235/314.

I will try to spread new versions of FetchRefs through Fred Fish, Aminet and ADS but no promises!

As the situation is today, I have no trouble in keeping up with update requests sent directly to me; actually I have recieved zero requests so far :-). Should you decide to request an update from me, please consider that sending mail costs me money!

1.7 The source code

FetchRefs is written completely in C and I compile it using a registered DICE - version 2.07.54. The distribution should have included the source.

You are hereby invited to do whatever you want to the source, but are in no way permitted to distribute the raped work by any means. If you really think of a great improvement, either just tell me about it and I will make it or you can make it and send me the source. I will then include it into MY release of FetchRefs. If you believe I have stopped the development and you want to take over, then you willl have to contact me first.
All this is because FetchRefs is MY program and I do not want five different FetchRefs 3.4 around.

If you should want to steal ideas or actual code from FetchRefs to other projects of yours then it is okay. Some credit would of course make me happy and a copy of the work would not harm either; but neither are required.

1.8 Author

I, the author, am 18 years old, own an Amiga 500+ with a GVP 121MB harddisk, 6MB RAM, a modem and a Toshiba CD-ROM. I have been programming since 1985 and currently I use C (I have been doing that since 1990, two years after I got my first Amiga) and study a little assembly. I program for the fun of it; but I would like to make a living of it someday.

I believe I have saved your day now - do you not just hate it when you know nothing about the author of the program you are using? No? NO? Shame on you :-).

If you would want to contact me (for registration, questions or whatever), I would prefer getting a letter (either on paper or electronic) because I can answer letters when I feel like it. I will try to answer all letters I get, no matter in what way they get to me.

Mail: Anders Melchiorsen
Gammel Skivevej 39
7870 Roslev
Denmark

Phone: +45 97 57 19 99

FidoNet: 2:235/314.10
UseNet: and@scala.ping.dk

1.9 Using GenerateIndex

GenerateIndex takes advantage of (see requirements):

- reqtools.library, copyright Nico François
- triton.library, copyright Stefan Zeiger

The index file that FetchRefs requires contains information about all the references. A slightly deeper discussion of the format can be found in the section the~edit~window but probably it is not really necessary for you to know the format.

You can have several index files, but I suggest for your own best that you keep it at one; this way you know exactly where you have what. If you start making several index files you will probably need to change the ARexx script you use, and therefore you should feel used to FetchRefs before doing this. The suggested name for your index file is 'S:FetchRefs.index'.

Requirements

1.10 Requirements

Apart from an Amiga with Kickstart 2.0+ you need a few additional programs if you want to take full advantage of GenerateIndex. These programs are the reqtools.library and the triton.library. Version 38 (release 2) of reqtools.library and version 2 (release 1.1) of triton.library is required.

If you do not have these programs, you will NOT be able to use the GUI of GenerateIndex; however, Shell usage is still available.

As Triton is a rather new package, I have included a minimal copy of the distribution and the Installer script will take care of installing it if you have not got it already.

ReqTools is not all that new and I believe that most people have it installed by now - therefore I did not consider including it worth the space it would take up. Of course this means that you will have to dig it up yourself, should you not have it. As it has been released as freely distributeable software this should not turn out too hard, though.

1.11 Arguments

The Shell template for GenerateIndex is:

```
FROM/M,TO,SETTINGS,RECURSIVELY/S,KEEPEMPTY/S,UNRECOGAREDOCS/S,AutoDoc/S,
C/S,C_DEFINE/S,C_STRUCT/S,C_TYPEDEF/S,E/S,E_CONST/S,E_OBJECT/S,E_PROC/S,
ASM/S,ASM_EQU/S,ASM_STRUCTURE/S,ASM_MACRO/S
```

or, for those who prefer BNF:

```
GenerateIndex [[FROM] {wildcard}} [TO <file>] [SETTINGS <file>]
   [RECURSIVELY] [KEEPEMPTY] [UNRECOGAREDOCS]
   [AutoDoc]
   [C] [C_DEFINE] [C_STRUCT] [C_TYPEDEF]
   [E] [E_CONST] [E_OBJECT] [E_PROC]
   [ASM] [ASM_EQU] [ASM_STRUCTURE] [ASM_MACRO]
```

- which is rather confusing unless you know GenerateIndex well. Therefore I suggest that you get used to GenerateIndex by using the GUI, before trying to use it without the GUI.

If you do NOT specify FROM as a Shell argument, the graphical user interface is opened. This requires that you have installed both ReqTools and Triton; see

requirements
.

If you start GenerateIndex from your Workbench, not all arguments are available (as tool types) and you are forced to use the graphic user interface. Actually, only the TO and SETTINGS arguments are available from Workbench, and even these two should not really be nessecary.

All arguments but FROM, TO and SETTINGS are used to set the scan options of GenerateIndex. Each argument reflect a gadget in the options window and you should read the
options
part of the guide to learn what they actually mean.

FROM

TO

SETTINGS

1.12 FROM

If you specify the FROM argument, it must be a specification of ↩
what files
to scan for references (i.e. either include files or AutoDocs). You can specify as many files as you wish, and if you specify the argument RECURSIVELY you may also specify a directory (containing e.g. all your include files). Note that you are actually specifying wildcards so e.g. "INCLUDE:*.h" is perfectly valid. This argument is the same as selecting files by using the
Add...
gadget of
the~main~window
of the GUI.

The FROM argument is not accessible via Workbench tool types - instead the GUI is provided. This is because FROM is intended to be used in a script that automatically recreates your index file; this script can then be run when you get new manuals or include files.
This is also why all the options arguments are only available for Shell usage - they are mainly useful in a script, otherwise you would prefer the GUI to set them.

An example script for creating an index file is provided (GenIndex.sh), but this example is very specific to my setup and really IS just an example - you will have to make one for yourself if you want one!

1.13 TO

```
                The TO argument specifies an index  file. If this file is already ↩
                    present
it will be loaded  and therefore any new  references will get appended  to
it. If it  is not present,  the name is  just used as  a "save name"  when
GenerateIndex is run in its script mode; i.e. when the
                FROM
                 argument  is
specified. You can not use the
                FROM
                 argument without also using TO.

If you enter TO as a tool type and start GenerateIndex via the  Workbench,
the specified file will get loaded before the GUI opens. If the  specified
file does not exist, the default file name of the file requester will just
be set to the  name. Thus you  can just press 'Ok'  in the file  requester
when you are about  to save an  index file and the  name specified in  the
tool type will get used.

By  default,   both   FetchRefs  and   GenerateIndex   are  set   to   use
S:FetchRefs.index as index file so you should not need to change anything.
This is  due to  the tool  types  I have  defined for  you; if  you  start
FetchRefs or GenerateIndex from  the Shell, no  default name is  specified
(as tool types are only read when run from Workbench).

The warning mentioned in
                Load~data...
                 is valid here, too!
```

1.14 SETTINGS

```
                The settings argument lets you specify a previously saved settings ↩
                    file to
load upon startup. If you do  not specify this argument, the items in  the

                Options
                   window   will   get   set   to   the   settings   saved   in
ENV:FetchRefs_GI.prefs (when you start from Workbench - data are saved  to
this file when you select Save or Use in the Options window).

If you start  from Shell  and do  not use  this argument  you are  telling
GenerateIndex that you want to set the options by hand (by using the other
Shell arguments) - thus no options will be activated if you do not specify
either SETTINGS  or  some of  the  option-activating arguments.  That  is,
ENV:FetchRefs_GI.prefs is  not automatically  loaded when  you start  from
Shell.

This argument  can also  be used  to simplify  a Shell  script; you  could
select whatever options you want with  the GUI, save them and then in  the
Shell simply  specify the  settings  file instead  of  typing in  all  the
settings yourself!
```

1.15 Options

GenerateIndex has several options concerning what files it should pick up and what kind of references it should search for in these files. These options can be set both by specifying arguments on the command line (Shell) and by using the GUI (the~options~window). The SETTINGS argument will load a previously saved set of options to quickly select your prefered ones.

In the following I will refer to them with the names from the~options~window. The Shell arguments for the same actions have less natural names but you should easily be able to figure out which is which.

Whenever a checkmark appears in a box in the options window, this option is turned on. Specifying any command line arguments will also turn an option on. Otherwise the option is probably off :-).

The options window is split in two parts. One which specifies what files you want to scan and what they should be scanned for. Below this are some gadgets with more global meaning; that is, they count for all kinds of files.

Files options

Other options

1.16 Files options

The many gadgets in the top box are used to select exactly what references GenerateIndex should look for when scanning files. To be able to operate this efficiently you will need to know how GenerateIndex scans the files.

For each file it first tries to figure out whether it is an AutoDoc, some kind of include file or something else.

Files are identified by certain keywords in the first 2KB of each file. The identifier list is currently as follows which should cover most (all?) cases. The list is case sensitive and "\n" means "new line". A file is considered recognized when just one of the keywords is found. The keywords are looked for in the order shown below.

AutoDoc: "TABLE OF CONTENTS"

```
        C include: "#if", "#define"

        E include: "\n(---) OBJECT", "\nCONST", "\nPROC"

Assembler include: "IFND", "EQU", "STRUCTURE", "MACRO", "BITDEF"
```

Depending on the state of the option
Unrecognized~files
GenerateIndex
may either ignore files it cannot identify or it may handle them as AutoDocs.

Once the file type is determined GenerateIndex checks if you are interested in this kind of files at all. You select this by the four gadgets at the very top, above the horizontal line. If you have not selected this kind of file it is simply skipped. That is, it is forgotten completely, no matter how you set the
Keep~files~without~references
option.

If you HAVE selected this kind of file GenerateIndex will start its scanner. If it is an AutoDoc there is only one thing to scan for: functions. Otherwise, if it is an include file, there are several things and each single part of the scanner can be turned on or off.

If you turn all the parts of a scanner off but still leave the file type selected it will generate a file without references everytime you do a scan. However, if you have set the
Keep~files~without~references
option
the files are not completely forgot and may be used along with the FILEREF option of
FR_GET
. Thus it is NOT the same to select/not select a file type even if you have not selected any of the sub parts of the particular scanner.

Okay, enough generic information, now I will go on to more specific talk about each of the scanners. It is of course only necessary for you to read about the kind of scanners you intend to use. On the other hand you really SHOULD read these parts. The scanners are rather poorly implemented in some respects and it may be good to know about limitations they may have.

AutoDocs

C includes

E includes

Asm includes

1.17 AutoDocs

Three things are important for the AutoDoc scanner.

A) The file MUST start with a line "TABLE OF CONTENTS". Otherwise the file is not even recognized as an AutoDoc (exception: set

Unrecognized~files
to "AutoDocs").

B) Each function must have a header like this:

nofrag.library/GetMemoryChain nofrag.library/GetMemoryChain

That is, the same string must be present both at the start and at the end of the same line. The last part of the string (everything after the last NON-alphanumeric character) is the function name. No leading spaces, etc. are allowed.

C) The end of each function must be marked by a "form feed" character (^L, ASCII value 12). The exception is the last function which may simply have the end-of-file as end marker.

All these requirements should be met by any AutoDoc if it wants to be "a real" AutoDoc. Consequently, unless you have an invalid AutoDoc these rules should not give any problems.

1.18 C includes

The special thing about the C scanner is that for struct/union references GenerateIndex will only use a part of the entire file as the reference. Apart from AutoDocs this is the only case where the entire file is not just loaded by FetchRefs when you request a reference look-up. Even though the cursor is placed at the right place in all the other cases it is still a bit faster and more memory efficient to just load the part that is really required. I hope to make more scanners work this way in the future.

The limits for each of the struct/union references are from the end of the previous struct/union up till the start of the next struct/union. This way you can read comments and other stuff related to this struct/union no matter if it is right before or right after the actual definition.

A note on the typedef scanner is that it is extremely poorly implemented. It will just use the last word in the typedef as the name of the reference. While this works for most of (all?) the typedef's in the Commodore includes it will fail when used on things like this:

```
typedef void ((*__sigfunc)(int);
```

Unfortunately it is hard to make the scanner much better without half-way compiling the includes (which is something I do not want to do :-).

The C scanner does not yet understand comments and therefore you can get

some references included even though they are actually commented out.

1.19 E includes

DISCLAIMER: Unfortunately I know very little about E. The E scanner is very simple and more or less expects files generated by ShowModule version~1.7. This is because all I know about the format of E files is gathered from a few fragments of the output from ShowModule. Of course GenerateIndex MAY work on other files but I cannot give any guarantees at all. I would like feed-back as to how sufficient you find the E scanner.

Also as a result of my limited knowledge of E, the scanner does not (yet) figure out exactly what part of the include file each reference is limited to. Therefore FetchRefs will load the entire file and just make the editor jump to the line where the reference begins. This works, but of course it would be nicer if you only had to wait for the right part of the include file to load.

There is not much to say about the different scan methods; CONST will of course toggle whether any constant definitons such as MODE_OLDFILE should be included in the index. OBJECT toggles indexing of the objects (in other languages knows as data structures).
The PROC scanner can be used if you have some modules and have run them through ShowModule to get a list of all the functions in the modules. By indexing this list you can quickly get a quick reminder on the arguments of a function. Of course this should not be used if you have a complete AutoDoc for the module. However, you seldom have an AutoDoc if it is for example your own, private module.

1.20 Asm includes

To be honest: making the Assembler scanner was an unplesant experience and it did not even turn out perfectly! The problem is that most of the things the scanner looks for is actually just macros. Therefore I had to not just scan for certain words but also to some extent emulate the macros used.

I think I have managed to get it working most of the time but I also think that it is still possible to confuse GenerateIndex, especially if you scan 'exec/types.i'. A line such as

```
        STRUCTURE    MACRO
```

will make GenerateIndex believe that both a STRUCTURE (named MACRO) and a MACRO (named STRUCTURE) is defined while only the latter is actually the case. More problems exist with 'exec/types.i' but most of them will just generate a few weired references that you will never meet anyway.

I do not use the Assembler scanner myself and is therefore very interested in your opinion on how well it works!

The only other thing really worth mentioning is that GenerateIndex will unroll BITDEF definitions and generate both the bit number and bit mask

name. For example: "BITDEF~MEM,CLEAR,16" will generate two references (named MEMF_CLEAR and MEMB_CLEAR) to the same line. Apart from this, the Assembler scanners should work as expected.

The Assembler scanner WILL understand comments and ignore the stuff in them. A comment is everything after a ';' or a '*'. This may lead to minor problems with constants declared as ';' or '*' but usually they should occour in situations where they are not confusing GenerateIndex.

1.21 Other options

These options are global in that they apply to each and every ↩
file no
matter if it is an AutoDoc, a C include file or whatever.

Scan drawers recursively

Keep~files~without~references

Unrecognized~files

1.22 Scan drawers recursively

With this option you can specify if you want the directory ↩
scan to be
recursive. That is, if GenerateIndex comes to a directory, should it look into it for more files?

The scan will always take directories you specifically specify into account. Let us say you have this directory tree:

```
a --- b --- c
```

Now, if you are on the level with the directory 'a' (as well as some files, possibly), you could multi-select several files/drawers. Let us say you included 'a'. If you have recursive scan turned on, GenerateIndex will pick all files in directory 'a', all files in directory 'b', all files in directory 'c' and so on - for all drawers that are on a lower level than 'a'.
Had the recursive scan been turned OFF, GenerateIndex WOULD scan all files in 'a' (that is, all files on the same level as directory 'b'), but it would not go any deeper. Therefore it would pick no files on the same level as 'c'.

The same is the case when you specify Shell arguments (
FROM
), only this
time you will have to enter the directory names instead of clicking on them. If DH0:a is a directory, specifying 'DH0:a' as a FROM file and leaving recursivity off will scan all *files* in 'a' but not any *directories*.

Damn, recursivity is so simple, yet so hard to clearly explain :-(.

1.23 Keep files without references

This option toggles whether files that contains no references should be
included in the index anyway. This is useful if you use the FILEREF option
of
FR_GET
(in FetchRefs).

For example, if you have turned scanning for #define's and typedef's off, exec/types.h would contain no references as it contains no structs. But if you search for 'types' you will still want types.h to show up; therefore the file must be included even though it contains no references.

On the other hand; if you do not use FILEREF, there is no reason for including empty files - that will just make the index file bigger. However, other than that it will not hurt so the general advice is to activate this option.

1.24 Unrecognized files

If GenerateIndex is unable to figure the kind of a file out it can either just forget the file or it can treat it like an AutoDoc.

If you scan AutoDocs that lack the "TABLE OF CONTENTS" line in the beginning (which is wrong) then you should set this option to "AutoDocs" as they will otherwise not be scanned. On the other hand, if you are not scanning AutoDocs you will not want GenerateIndex to try to read other files as AutoDocs as this can be a rather confusing experience for GenerateIndex.

Generally you should only use this option (i.e. set it to "AutoDocs") if you experience problems when scanning AutoDocs.

1.25 Windows

GenerateIndex has several windows, each used to do different things.
Following this is a description of each window and what you can do with
it.

The~main~window

The~edit~window

The~options~window

1.26 The main window

The main window is the window that is first opened when you start
GenerateIndex. This window is for the biggest part covered by a listview
gadget, containg the names of all the files you have indexed so far. Below
this is a little gadget showing how many references the currently selected
file contains. Even further down are four gadgets and if you press the
right mouse button you will find some menus.

Add...

Edit...

Rescan

Delete

Menus

1.27 Add...

This gadget will open a file requester. Any file you select here will get
scanned and added to the list. Depending on the kind of file it is and how
you set the
options
, the actual scanning procedure may vary.

You can select several files in one go: hold down a shift key and select
all the files you please by clicking on each of them a single time. It is
not possible to select files in different levels of the directory hierachy
- to do this you will have to select 'Add...' several times.

Depending on the set
options
, directories selected may be scanned
recursively or not.

1.28 Edit...

Clicking on this gadget will open
the~edit~window
. The same effect can
be achieved by double clicking on an item in the list view.

1.29 Rescan

This will scan the currently selected file again, with the options ↩
set at
the time you select rescan. This could be used if you set some options
wrong and do not feel like selecting files again. However, as you have to
rescan each file seperately it will probably be faster to just
delete
them all and then
add...
them if you have a big number of files to
rescan.

A thing to note is that the file could disappear if you rescan it with
some options that produce no references and do not have the

Keep~files~without~references
option set.

1.30 Delete

Without any warning, this gadget deletes the currently selected ↩
file from
the list. This is mainly useful if you by accident included a file too
much when you generated the index or if you added some files with wrong

options
.

1.31 Menus

The following menu items are attached to the main window of ↩
GenerateIndex.

Clear

Load~data...

Save~data...

Options...

About...

Quit

1.32 Clear

This will clear the entire list of indexed files and thus begin a new index file. A warning will be given before flushing the list.

1.33 Load data...

'Load data' will load an index file previously saved with Save~data...
.

If some files are already indexed, GenerateRefs will ask whether it should clear the list before loading or if it should append the file you specify to the end of the list.

Warning: In the current version, FetchRefs has no ID in its index files. Therefore it can not distinguish real index files from anything else but will treat all specified files equally. Loading a non-FetchRefs file is a VERY bad idea!

1.34 Save data...

'Save data' simply saves the entire index to a files which can then later be loaded by FetchRefs or GenerateIndex.

By default, FetchRefs is set to use S:FetchRefs.index as its index file, so by specifying this name you do not have to change any settings for FetchRefs. This is due to the tool types I have defined for you; if you start FetchRefs or GenerateIndex from the Shell, no default name is specified.

1.35 Options...

This menu item opens
the~options~window
.

1.36 About...

About shows the version number and compilation date of the GenerateIndex you are using, as well as my name. Nothing interesting in here :-).

1.37 Quit

Quit will abort GenerateIndex. If you have not saved your data file it will be lost without a warning. The exact same action is performed if you click in the close gadget of the main window.

1.38 The edit window

This window is much like
the~main~window
, only this time the list
contains all the references in the file currently selected in the main window. Furthermore, more information is shown.

Some of the information entries speak for themselves; however, it should be noted that the Offset and Length are counted as bytes in the original file.

'Line' is counted in lines from the start of the reference (Offset) and is the number of lines to jump in the reference to find the start of the "important" stuff.
For example, when indexing C include files, all comments from the previous struct and up till the current is included in the reference. This is because the comments before a struct definiton often contain useful notes. However, the most important information is the struct itself and therefore Line will be equal to the number of lines the comments before the struct fills out.

FR_GET
will return this number upon a successful reference
look-up so the ARexx script can figure out where to position the cursor.

If Length is -1 it simply means "the entire file". The actual length is figured out by FetchRefs once it is fetching that particular reference.

Should you not understand the above explanations, do not worry! Most probably you need not know about it - actually you should only rarely need to open the edit window at all!

The Okay gadget will close the edit window while "Delete" will delete the selected reference. This is useful if you have several references of the same name but only really needs one of them - this way you can delete the others and keep FetchRefs from popping up its window, asking which one you want.

The original idea was that this window should allow you to change the informations as well, but I came to think of what that could be used for and I figured out: not much. So, the only editing capability is really the ability to delete specified references.

1.39 The options window

This window is opened by selecting the menu item
Options...
. It contains
many gadgets to configure the scan phase of GenerateIndex to act exactly like you want it to.

Furthermore it has a little menu to enable you to load and save your settings. This is only needed if you need more than one set of settings.

Otherwise you will use the Save and/or Use gadgets which will save the settings to the files "ENVARC:FetchRefs_GI.prefs" respectively "ENV:FetchRefs_GI.prefs".

The Cancel gadget and close icon of the window will both leave the options window and set the options back to what they were when the window was first opened.

This should all be close to the standard Preferences behaviour and hard to misunderstand ;-).

See the
options
part for further information on what the other gadgets actually do.

1.40 Using FetchRefs

Please note that before you can use FetchRefs at anything, you ↩
will have
to generate an index file. Without this index file FetchRefs would become very, very slow at finding a reference.
Therefore; before you do anything else, you should make sure that you've read and understood the
Using~GenerateIndex
chapter. Ok, so the index
file is ready? Then you may continue.

FetchRefs can be started either via Workbench or the Shell; both has advantages as well as disadvantages, which I will not discuss deeply here. Depending on what method you prefer, you will have to enter the arguments differently (tool types versus command line options). I will asume that you know how to pass arguments, and just list what they actually are to FetchRefs.

The~arguments

The~ARexx~interface

1.41 The arguments

The syntax of FetchRefs is (in standard AmigaDos notation):

```
FILES/M,PORTNAME,RUNONCE/S
```

If you prefer this BNF-like approach, then it should mean the same:

```
FetchRefs [FILES {wildcard}] [PORTNAME <name>] [RUNONCE]
```

As you can see(?), you can specify as many FILES arguments (index files) as you please, and you do not have to enter the word 'FILES'. However, if you want to use any of the other options you will have to specify the keyword also.
If you want to use tool types (by starting via Workbench/WBStartup), you simply enter each argument as a tool type. To enter several index FILES you simply make several tool type lines, all starting with 'FILES='

In the following sections, each argument is explained.

FILES

PORTNAME

RUNONCE

1.42 FILES

The FILES argument specifies what index file(s) FetchRefs should initially load. This/these file(s) are generated by GenerateIndex and are unreadable for anything but FetchRefs/GenerateIndex. From the Shell you can enter as many FILES as you please, simply by listing them one after another. It is, however, suggested that you keep the index to one file, for two reasons:

1. It is faster to load one big file than ten small ones
2. You get less confused with just one file

One issue to remember is that what you specify via the FILES argument are not really file names but rather wild card specifications. This means that you can use all the standard things like # ? [] and so on to specify the FILES. If you enter a wild card that matches no files, a warning is written to the Shell window (nothing happens when run from Workbench). This can be handy mostly if you specify the name without wildcards but make a typing mistake; except from that warning, FetchRefs will just consider it a wild card with no match and do nothing more about it.

It is not required that you enter any FILES. For example, people with very limited memory might want to have no files loaded and then use the ARexx function

FR_NEW
to load the index file right before they use
FR_GET
to fetch the reference.

Concerning memory usage, only the FILES that can fit in memory are loaded. If several FILES are specified, some might be loaded while others might not, depending on the amount of free memory.
If an unfragmented piece of memory which is at least as big as the index file exists, a fast loading routine is used. If this is not the case, a slow routine is used. This slow routine does not care whether the memory is unfragmented, as long as it exists - but it is both real slow and requires some more memory than the fast one (to keep track of all the small chunks it allocates).
Generally this is no problem as no-one want to have an index file so big that they have no memory left - instead one would chose to have less references in it, e.g. by not indexing #define statements (for C include files).

1.43 PORTNAME

The PORTNAME argument specifies what the name of FetchRefs's ARexx port should be. The default is 'FETCHREFS'. If a port of the specified (or default) name already exists, a suffix of ".nn" will be applied where nn is a number increasing from 01 to 99 until a free port is found. If FetchRefs is unable to find a free port name it will exit with an error message telling you that there is no free store. This is very unlikely, however.

By the way, I do not see any reason why you'd want to change the name of the ARexx port.

1.44 RUNONCE

If you use the RUNONCE argument, FetchRefs will not allow several ↩
copies
of itself to run at the same time. Instead, when you run FetchRefs for the second time, it will signal the already running copy to quit. You can then run FetchRefs for a third time and have it installed again.

The way FetchRefs uses to signal the already running copy is to send it the ARexx message
FR_QUIT
. Therefore the new FetchRefs needs to know the
port name of the running copy. Unless you specified a special name when the first copy was run, this is no problem - 'FETCHREFS' is used by default both times. However, if you DID change the portname for the first copy, you need to specify the SAME name (using the
PORTNAME
argument)
when you run FetchRefs to quit it!

RUNONCE is only needed when you want to quit an already running copy. You can specify RUNONCE even when no FetchRefs with the specified port name is running but in this case the RUNONCE argument is simply ignored.

1.45 The ARexx interface

FetchRefs has an ARexx interface which makes it fit into ↩
actually any
environment as long as the editor used also has ARexx support. The ARexx part of FetchRefs is really divided in two parts: the actual ARexx functions of FetchRefs and the supplied scripts, which are needed to interact with the editor.

For the most basic usage of FetchRefs you do not need to know particulary much about any of these, but if you want to customize FetchRefs a bit you will probably want to know how and why FetchRefs does what and when.
In any case you should read the
FR_GET
part as it covers a big part of
what FetchRefs offers.

One VERY important thing to remember is that ARexx MUST be running in the background before you can use the ARexx interface of FetchRefs (or any other product for that matter - including your editor). This is normally done in the s:startup-sequence by calling RexxMast, so unless you have changed your Amigas start up procedure this should not be a problem.
If you HAVE mingled with the s:startup-sequence, or if you are not starting using the standard Workbench disk, you should make very certain that you are still executing RexxMast.
If you use Workbench, you can move the RexxMast icon from your System drawer into your WBStartup drawer; but once again: this is ONLY if it is not present in your s:startup-sequence.

The~ARexx~commands

ARexx~scripts

1.46 The ARexx commands

FetchRefs supports just a few ARexx commands but they cover ↩
everything
FetchRefs has to offer... actually ARexx is the only way to activate FetchRefs!

Below all the ARexx commands are described. They are also provided in the file called FetchRefs.doc (without icon) in a format that is directly suitable for processing by GenerateIndex (standard AutoDoc format). This file could be the data for your first test run :-).

FR_ADD

FR_CLEAR

FR_GET

FR_NEW

FR_QUIT

1.47 FR_ADD

```
FetchRefs/FR_ADD                                        FetchRefs/FR_ADD

   NAME
        FR_ADD -- load additional index files

   SYNOPSIS
        FR_ADD FILE/M

        FR_ADD [wildcard ...]

   FUNCTION
        FR_ADD will load extra  index files and add  them to the  internal
        list. The index files already in memory are not removed.

   INPUTS
        FILE/M - wild card specification(s) for the index files to load.

   RESULTS
        None.

   BUGS
        None known.

   SEE ALSO
        FR_CLEAR, FR_NEW
```

1.48 FR_CLEAR

```
FetchRefs/FR_CLEAR                                      FetchRefs/FR_CLEAR

   NAME
        FR_CLEAR -- remove any index files from memory

   SYNOPSIS
        FR_CLEAR

   FUNCTION
        Free all memory allocated to store loaded index files. Most of the
        memory FetchRefs uses is for the  index file(s), so this will  put
        FetchRefs in a low-memory sleep  mode. By later calling FR_ADD  or
        FR_NEW the original state can again be achived.

   INPUTS
        None.

   RESULTS
```

None.

BUGS
None known.

SEE ALSO
FR_ADD, FR_NEW

1.49 FR_GET

FetchRefs/FR_GET FetchRefs/ ↩
FR_GET

NAME
FR_GET -- get a reference into a file or the clipboard

SYNOPSIS
FR_GET FIND/A,TO/A,PUBSCREEN,FILEREF/S,CASE/S

FR_GET keyword filename [public screen name] [FILEREF] [CASE]

FUNCTION
Searches the internal list for a name matching the FIND keyword.

The FIND argument is a wild card. Thus you can search on things like "Open#?" and get a long list of functions starting with "Open". Many more wild cards exist, if you do not know how to use them you should do yourself the favour of learning them - they are quite powerfull in other situations, too. Of course you can ignore the wild card feature completely if you know exactly what you are searching for - or if you do not know wild cards :-).

One important note here: though FR_GET supports all wild cards, the provided ARexx scripts do not! Only # and ? are supported through the scripts, except
the~script~for~Shell
which also
supports all wild cards. Confusing, eh?

When no matches are found, an error is returned. If excatly one is found, the reference is written to the filename specified by the TO argument. If several references exist to the same keyword, a window pops up (on the default screen or whatever screen is specified by the PUBSCREEN argument, see below). This window contains a listview gadget with the file names of all the files that contains a reference of the requested name. A double click on any of these files will get the reference bound to this file. If the window is closed or Esc is pressed an error, "FetchRefs: Aborted!", is returned.

The screen, on which this window is to be opened, is specifed by using the PUBSCREEN argument. You specify the name of a public screen which may not be in private mode (they rarely are). If the specified screen is not available (non-existent or non-public) or

if you do not specify PUBSCREEN at all then FetchRefs will open the window on the currently active screen. Should this not be public, the default public screen (usually Workbench) is used.
No matter where the window opens that screen will be brought to front (if it is not already there). When you have finished the selection and the window closes, the screen is again put behind the other screens (but only if it was brought to front in the first place).

If the FILEREF argument is given, each of the files in the index file will be considered a reference themselves. The name of the reference is the filename without any leading path or suffixes. For example, the file 'DINCLUDE:Amiga30/exec/types.h' would be considered a match if you search for the reference 'types'. The reason for this truncation is mainly due to the way FetchRefs works otherwise; types.h must be truncated at the dot if 'types' was a structure and not a file name - and FetchRefs really has no way of knowing what it actually is, until the match is already found; so, the most sensible idea seemed to trucate everything at the first non alpha-numeric character.

Depending on whether you set the CASE flag or not, the comparsion of the reference names is either case sensitive (CASE specified) or case in-sensitive. While case in-sensitivity is a good idea when you are not certain what you are searching for, case sensitivity will reduce the number of matches when you search for something that exist twice, only with different capitalization - and with a reduced number of matches, you get less confused :-).

To put the reference into the clipboard instead of a file, a filename of "CLIPnn" should be specified, with nn being the number of the clipboard unit you wish to use. The "CLIP" word must be in uppercase, otherwise the name is considered an usual file name.

INPUTS

```
FIND/A    - name of reference to search for. Wild cards accepted.
TO/A      - file name to put the result into. "CLIPnn" specifies
            the clipboard unit nn.
PUBSCREEN - public screen to open "select reference" window on.
            Default is the currently active screen (if public,
            otherwise the default public screen).
FILEREF   - let a reference search on the base name of a file
            match with the entire file.
CASE      - activate case sensitive search.
```

RESULTS

rc will be 0 on success, 10 otherwise.

rc2 contains additional information; if rc is 10 then it will be a text string describing the error, otherwise rc2 is a number specifying at what line the most important part of the reference begins. What is most important depends on the kind of file the reference is in, but basically if rc is zero, you will most likely want to jump to line rc2 after having loaded the generated file into your editor.

BUGS

```
        None known.

   SEE ALSO
```

1.50 FR_NEW

```
FetchRefs/FR_NEW                                        FetchRefs/FR_NEW

   NAME
        FR_NEW -- clear internal index list and load a new

   SYNOPSIS
        FR_NEW FILE/M

        FR_NEW [wildcard ...]

   FUNCTION
        This is a combination  of FR_CLEAR and FR_ADD  and results in  the
        internal list being  set to  nothing but what's  specified by  the
        FILE arguments.

   INPUTS
        FILE/M - wild card specification(s) describing what files to load
                 instead of the current list.

   RESULTS
        None.

   BUGS
        None known.

   SEE ALSO
        FR_ADD, FR_CLEAR
```

1.51 FR_QUIT

```
               FetchRefs/FR_QUIT                                  FetchRefs/ ↩
                  FR_QUIT

   NAME
        FR_QUIT -- force FetchRefs to quit

   SYNOPSIS
        FR_QUIT

   FUNCTION
        Will send a ^C signal to the FetchRefs process that owns the ARexx
        port. This  will force  FetchRefs to  free all  allocated  memory,
        close down the ARexx port and exit.
        A similar  effect can  be achived  by using  the C:Break  program,
```

```
        running FetchRefs again with the
                RUNONCE
                 argument or by  sending
        a ^C by any other means.

   INPUTS
        None.

   RESULTS
        None.

   BUGS
        None known.

   SEE ALSO
        exec.library/Signal :-)
```

1.52 ARexx scripts

Included in the FetchRefs distribution are some pre-made ARexx ↩
scripts for
popular editors (the list is shown by the Installer script).

I consider it your problem to figure out how you execute the script from your particular editor. The basic idea, however, is to assign the command "execute arexx script 'GoFetchRefs'" to a key and then simply press that key when your cursor is at the start of the word you want a reference for. If the cursor is in the middle of something, FetchRefs will believe the word starts here and most likely NOT find any reference (this is a feature, btw).

If you want to change the options of FetchRefs (as described in
FR_GET
)
you must locate the line starting with "FR_GET" and change it according to your needs. This line is present in all the scripts.

If you enhance a script or invent an entirely new script for an editor that is currently not yet supported, I would be very interested in getting a copy so I can distribute it along with further versions of FetchRefs that might be.

The~script~for~Shell

1.53 The script for Shell

Apart from all the scripts for editors, a generic one is also supplied. This script does not need any editor and just prints the reference to a Shell window. You execute it from a Shell prompt like this:

```
  Shell> rx GoFetchRefs OpenScreen
```

Of course you change "OpenScreen" to whatever you want to search for. Wild cards are accepted, so you could go "rx GoFetchRefs (Open|Close|%)Screen" - if you should happen to know these wild cards ;-).

1.54 Problems and tricks-tips-hints

Below is a list of the problems I figured you could encounter ↩ . It is probably very incomplete as it is quite hard for me to guess what you cannot figure out. Therefore I would very much like you to send me a letter if you come across a problem that is not described here; only that way I can include the problem in this list and possibly save others from the trouble you had.

Unsupported~editor

Limited~memory

FetchRefs~does~not~understand~AmigaGuide~AutoDocs!

1.55 Unsupported editor

The greatest problem is most likely that you use an editor which I have not provided a script for. I cannot really help you with that one, because if I had had access to that editor I would have made an ARexx script for it. If it is a freely distributeable editor you can send me a copy and I will provide the script (remember that I cannot make the script if I do not know the editor's commands so I also need a copy of the documentation).

If your editor does not have an ARexx port or the port is very limited, you are probably out of luck and should look around for another editor. The functions that the port should be capable of doing are:

- Get the current word (the word the cursor is at)
- Open a new window/view
- Load a file into the newly created window

A function to jump to a specified line is handy for include file references and it will give a more integrated system if there is a way to tell if anything went wrong. However, this is not really required. If the editor does not support the above mentioned functions, I honestly cannot see how it should work. It is, however, some of the most basic functions I can think of, so if the editor has an ARexx port, it probably supports them in some way.
If the editor cannot return the current word or if it has a 'wrong' way of thinking what a word is (it is NOT from white space to white space) then you are not completely out of luck: if the editor can tell about the

entire currently active line as well as the column the cursor is at, you will be able to get the word from these. I do this myself in for example the GoldED script.

I have been able to make scripts for all the editors I have tried, but not without some problems. By reading the various scripts you may be able to learn a few tricks, though a certain limitation of a certain editor can be hard to work around.

If you should succeed in creating a new script I would be happy to get a copy of it so that I can include it in a future release.

1.56 Limited memory

The index file that FetchRefs uses is quite big to keep in RAM ↩ all the
time - considering that it is not used very often.

A solution would be to make a script the invokes the
FR_CLEAR
and

FR_ADD
ARexx commands. This way you can flush the index whenever you do not use it. With a relatively fast harddisk this will make each loop-up take just a little longer and you only use the RAM for the index file when you actually need it.
I have not implemented this into the ARexx scripts provided as I wanted to keep them quite simple - by getting sophisticated, more people will experience problems with their particular setup.

1.57 FetchRefs does not understand AmigaGuide AutoDocs!

The original AutoDocs are plain text files with a very simple format. There exist, however, programs to convert these files into different formats - a very popular one being the AmigaGuide format (just like this guide). With the AutoDocs as a guide you can jump in the text by simply clicking on the function name, etc. that you want information about.

This does not work very well together with FetchRefs as GenerateIndex does not understand how to scan guides for keywords. Therefore you need to have the original AutoDocs installed if you want to use FetchRefs - however, with FetchRefs running, regular AutoDocs are just as useful as those fancy guides.