OL.doc

COLLABORATORS			
	TITLE : OL.doc		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		September 19, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

OL.doc

1.1 OL.doc

```
                    $RCSfile: OL.doc $
Description: Documentation for the Oberon-A pre-link utility

 Created by: fjc (Frank Copeland)
  $Revision: 2.2 $
    $Author: fjc $
      $Date: 1995/07/02 16:56:50 $

Copyright © 1994-1995, Frank Copeland.
```

```
Links marked with "+" are new, those with "*" have been changed.


               ~Description~~~~~~~~
                  What does it DO?

               ~Distribution~~~~~~~
                  Copyright and distribution

                        Running OL ...

               ~Shell~~~~~~~~~~~~~~
                  ...from the Shell

               ~Workbench~~~~~~~~~~
                  ...from the Workbench

               ~FPE~~~~~~~~~~~~~~~~
                  ...from the FPE utility

               ~Preferences~~~~~~+~
                  Preferences settings


               ~Linkers~~~~~~~~~~~~
                  Using different linkers with OL
```

1.2 Distribution and Copyright

OL is part of Oberon-A and is:

See Oberon-A.doc for its conditions of use and distribution.

1.3 Running OL from the CLI

Format: [PROG] <program> [SCAN] [LINK] [SETTINGS <filename>]

Template: PROG/A,SETTINGS/K,SCAN/S,LINK/S

Purpose: To prepare a list of object files to be linked together to create a given executable program. Optionally, to call a linker to perform the link.

Path: Oberon-A:OL

Specification:

OL must be given the name of a module which is the main module of a program. The program will start execution at the start of the main body of that module, and will have the same name.

If the SCAN argument is given, OL will recursively scan the symbol file of the main module, and those of the modules it imports, to produce a list of the modules included in the program. It will then locate the object files of those modules and write their names to a file intended to be processed by a linker. This file will be referred to as the "linker file".

If the LINK argument is given, OL will run the default linker, passing it the name of the linker file and the default arguments.

OL has a number of
preferences~settings
that affect its operations. The
SETTINGS argument can be used to specify the name of a preferences file that is loaded before any other arguments are processed. If no SETTINGS argument is specified, the default preferences file is "OL.prefs". Preferences files are searched for first in the current directory, then in "PROGDIR:" (the directory containing OL), and finally in "ENV:OL".

Preferences files can be viewed and edited with the OLPrefs tool.

The Shell stack should be set to at least 12000 bytes. See the Stack command in the AmigaDOS manual.

1.4 Running OL from the FPE utility

A tool button in the FPE window can be configured to run OL (see FPE.doc). In the button editor, set the Command field to the full path name of the OL program. Set the Arguments field to "!P" plus any options that are desired. Specify a console window as the Output field. Put at least 12000 in the stack field.

For example:

```
Command="OBERON-A:OL"
Arguments="!P SETTINGS=OL.prefs SCAN"
Console="CON:0/11/540/189/Pre-linking.../CLOSE/WAIT"
Stack=12000
```

To pre-link a program:

1. select the program using the Open menu item.
3. click on the tool button OL is bound to.
4. sit back and relax for a few seconds.

1.5 Running OL from the Workbench

See
Running~OL~from~the~Shell
for a general description of OL's
operation and the effect of the various arguments.

OL can be run in two ways:

1. Select the icon for OL, then double-click the program to be linked. DON'T do it the other way around, or you will run the program instead of OL.

2. Select the icon for the drawer in which the program executable is to be written, then double-click the OL icon. In this case the name of the program must be specified as a tooltype in OL's icon, as "PROG=<program>". Use this method when a program is being linked for

the first time.

All arguments available when running OL from the Shell can be specified as tooltypes in OL's icon. The tooltypes can be edited by clicking the icon and selecting the "Information" item from the Workbench "Icons" menu.

For switch arguments like SCAN the name of the argument is entered as a tooltype. Keyword arguments like SETTINGS are entered as "SETTINGS=<argument>". The standard WINDOW tooltype is also understood by OL. If it is omitted a default console window is opened.

A typical list of tooltypes might look like this:

```
WINDOW=CON:0/0/640/200/Compiling.../CLOSE/WAIT
SETTINGS=OL.prefs
SCAN
(LINK)
```

Enclosing the LINK argument in parentheses disables it without the need to delete the entire tooltype. To enable it, edit the tooltype to remove the parentheses and save the icon.

If you often use two or more different preferences files and/or argument lists, it may become tedious to constantly edit OL's tooltypes to change the arguments. This can be solved by using the Shell command MakeLink to create a copy of OL and creating a seperate icon for it. See the OL-Small icon in the Oberon-A directory for an example.

The default stack should be set to at least 12000 bytes.

1.6 Preferences settings

Preferences settings are used to customize the operation of OL. OL loads its settings from a file, which can be specified on the command line or in the tooltypes. The default settings file is "OL.prefs". When searching for settings files, OL looks first in the current directory, then in "PROGDIR:" (the directory containing OL), then in "ENV:OL". Settings files can be viewed and edited using the OLPrefs utility.

The settings are:

Search Paths

Directories to be searched for symbol and object files. These can be absolute paths, or relative paths. For example, "OLIB:" is an absolute path, and "Code" is a relative path, meaning the sub-directory "Code" in the current directory. Up to ten search paths can be specified. The current directory is searched first by default.

Output Paths

Directories in which to output linker files and executables. Again, these can be absolute or relative paths. The default is to output in the current directory.

Extensions

OL constructs file names by appending an extension to the module or program name. Extensions can be specified for symbol, object and linker files. The defaults are ".sym", ".obj" and ".with" respectively.

Linker

The linker setting determines what format to use when generating linker files, either ALink or BLink. The LinkCmd setting specifies the full path of the linker. The LinkArgs setting specifies any arguments to be passed to the linker along with the name of the linker file.

Miscellaneous

If the Verbose setting is TRUE, OL produces a long-winded description of the link.

If the Make Icons setting is TRUE, OL creates icons for any executable files it outputs, if they do not already have one. The default icon is expected to be in the "ENV:OL" directory, with the name "def_prog". If it cannot be found, the system default tool icon is used instead.

1.7 Using different linkers with OL

OL currently supports two linkers: ALink and BLink. This support consists of generating different linker file formats for each linker. There is no reason why OL in its current form should not be usable with other linkers, as long as they understand one of the supported linker file formats. At least one person uses AmigaOberon's OLink, using the BLink format.

Using OL with ALink

Run OL with the ALINK setting. Edit the LINKCMD setting to point to the alink program. The LINKARGS setting can contain any arguments recognised by ALink, except for the FROM, WITH and TO arguments.

Using OL with BLink

Run OL with the BLINK setting. Edit the LINKCMD setting to point to the BLink program. The LINKARGS setting can contain any arguments recognised by BLink, except for the FROM, WITH and TO arguments.

Using OL with OLink

Run OL with the BLINK setting. Edit the LINKCMD setting to point to the OLink program. The LINKARGS setting can contain any arguments recognised by OLink, except for the FROM, WITH and TO arguments.

As of version 2.10, OL no longer supports DLink. This support may be

reinstated in future.

1.8 What can go wrong?

Error messages are written to the standard output. The errors can be one of:

* " !! Bad key in module %s"

  The key in the symbol file for the named module is different from the key expected by another module that imports it. This is caused by changing the definition of a module without recompiling all the modules that import it. The remedy is to determine the modules that import it and recompile them with the NEWSYMFILE option. See ORU.doc for a description of a way to automate this.

* " !! Could not find symbol file %s"

  OL could not find a symbol file with the given name. Either the file doesn't exist, or you haven't specified the directory it is in in the SYMSEARCH setting.

* " !! Could not find object file %s"

  OL could not find an object file with the given name. Either the file doesn't exist, or you haven't specified the directory it is in in the OBJSEARCH setting.

* " !! Out of memory"

  Says it all, doesn't it?

* " !! Could not open %s"

  OL found the given file, but for some reason could not open it for reading.

* " !! Bad tag in symbol file %s"

  The identifying tag in the symbol file (the first 4 bytes) is not what OL expects. This can mean:

    * the file is not actually a symbol file, or is corrupt.
    * the symbol file was generated by an obsolete version of the compiler.
    * I have forgotten to update OL after changing the compiler :-).

* " !! Bad modAnchor in symbol file %s"

  A reference to an imported module in the named symbol file is corrupt.

* " !! Module name too long in symbol file %s"

  The name of an imported module is too long. This should never

happen.

* " !! Bad name in symbol file %s"

  The first module name in the symbol file is not the same as the name of the symbol file.

* " !! Could not create %s"

  OL could not create a file with the given name. AmigaDOS has had some sort of fit.

* " !! Error closing %s"

  OL could not close the named file.

1.9 Who is responsible for THIS?

OL was written by Frank Copeland.

All bug reports, suggestions and comments can be directed to:

Email : fjc@wossname.apana.org.au

Snail Mail :

Frank J Copeland
PO BOX 236
RESERVOIR VIC 3073
AUSTRALIA

Remember the J. It saves a lot of confusion at my end :-).

1.10 Reporting bugs and suggestions

You are encouraged to report any and all bugs you find to the~author
, as
well as any comments or suggestions for improvements you may have.

Before reporting a suspected bug, check the file ToDo.doc to see if it has already been noted. If it is a new insect, clearly describe its behaviour including the actions necessary to make it repeatable. Indicate in your report which version of OL you are using.

1.11 Release History

0.0 The initial version, written in Modula 2 and compiled by the Benchmark compiler.

```
0.1   The initial conversion from Modula 2 to Oberon, compiled by the
      v0.0 compiler.

0.2   Bug fixes and improvements.

1.0   Start of revision control.

1.1   * Command line arguments slightly changed.
      * Changed format of .with file to suit Commodore's ALink.
      * OLIB: is now the default symbol file search path.
      * [bug] The module name passed as a parameter was case-sensitive.

1.2   Source code edited to use new Amiga interface.

1.3   * Added ALINK, BLINK and DLINK arguments.
      * Changed to output in different formats depending on the linker.

2.0   Added option to call linker directly.

2.1   Cleaned up for release

2.2   Minor modifications

2.3   * Changed to support new symbol file format.
      * Automatically includes module Kernel.
      * Searches for external libraries found in symbol file.

2.4   Modified to use new interface to module Strings.

2.5   Modified to use modules In and Out for console IO.

2.6   Further minor modifications.

2.7   * Added support for preferences settings.
      * Added Workbench interface.
      * Improved support for DLink.

2.8   Intermediate version

2.9   Greatly simplified the command line, removing options that
      modified preferences settings.

2.10  Removed support for DLink; it wasn't worth the effort.
```

1.12 What does it DO?

OL is *not* a linker, although in time it may become one. However, used correctly it will appear as if it were a linker to the user. It is used to integrate a third-party linker such as BLink with the rest of Oberon-A, to remove the need to write a custom linker.

OL performs two tasks, which may be done seperately or together. They are:

- to recursively scan the symbol file of a program's main module, and

those of the modules it imports, to construct a list of the modules making up the program.

- to call the linker itself, passing it any parameters it requires.