OL.doc

COLLABORATORS			
	TITLE : OL.doc		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 30, 2024	

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: 2.1 $
    $Author: fjc $
      $Date: 1995/01/25 23:35:36 $

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

 Linkers               Using different linkers with OL
 Error Reports         What can go wrong and how to fix it

 The Author            Contacting the author
 Bugs & Suggestions    Reporting bugs and suggestions
 Changes               Changes since the last release
 To Do                 Bugs to fix and improvements to make
 Release History       Release history of the program
```

1.2 Distribution and Copyright

```
OL is part of Oberon-A and is:
```

1.3 Running OL from the CLI

```
Format:         [PROG] <program> [SCAN]([LINK]
                [SETTINGS <filename>]
                [SYMSEARCH <pathlist>] [OBJSEARCH <pathlist>]
                [WITHPATH <directory>] [PROGPATH <directory>]
                [SYMEXT <extensionlist>] [OBJEXT <extensionlist>]
                [WITHEXT <extension>] [LINKCMD <linker>]
                [LINKARGS <arguments>] [ALINK | BLINK | DLINK]
                [VERBOSE | QUIET] [MAKEICONS | NOICONS]

Template:       PROG/A,SETTINGS/K,SYMSEARCH/K,OBJSEARCH/K,
                WITHPATH/K,PROGPATH/K,SYMEXT/K,OBJEXT/K,WITHEXT/K,
                LINKCMD/K,LINKARGS/K,ALINK/S,BLINK/S,DLINK/S,
                VERBOSE/S,MAKEICONS/S,QUIET/S,NOICONS/S,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 an application. The application 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 ".with file".

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

Individual preferences settings can be over-ridden by providing new values for them on the command line. The following settings are currently supported:

SYMSEARCH: one or more directories to be searched for symbol files. If more than one directory is specified, the entire list must enclosed in quotes.

OBJSEARCH: one or more directories to be searched for object files. If more than one directory is specified, the entire list must enclosed in quotes.

WITHPATH: the directory in which the .with file is to be written.

PROGPATH: the directory in which the program executable is to be written by the linker.

SYMEXT: one or more file extensions that are used for symbol files. If more than one extension is specified, the entire list must be enclosed in quotes.

OBJEXT: one or more file extensions that are used for object files. If more than one extension is specified, the entire list must be enclosed in quotes.

WITHEXT: the file extension to be used when creating a name for the .with file.

LINKCMD: the full path for the linker's executable.

LINKARGS: arguments to be passed to the linker. If there is more than one argument, enclose the entire list in quotes.

ALINK, BLINK, DLINK: the format to be used for the .with file. These options are mutually exclusive, and only one may be specified.

VERBOSE: OL produces a verbose description of its operation.

QUIET: turns off the VERBOSE setting if it is set in the preferences file.

MAKEICONS: OL will create an icon for the .with file and the program executable, if there is not one already. The default icons are found in "ENV:OL/def_with.info" and "ENV:OL/def_prog.info".

NOICONS: turns off the MAKEICONS setting if it is set in the preferences file.

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 10000 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=10000
```

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 SYMSEARCH are entered as "SYMSEARCH=<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 Using different linkers with OL

OL currently supports three linkers: ALink, BLink and DLink. This support consists of generating different .with 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 .with 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.

Using OL with DLink

Run OL with the DLINK setting. Edit the LINKCMD setting to point to the dlink program. The LINKARGS setting can contain any arguments recognised by DLink, except for "-o" and "" arguments. The options should include "-L0" to suppress DLink's default search paths for library files, as these are fully specified by OL.

DLink has an unfortunate convention that makes it a bit more difficult to use with Oberon-A than either BLink or ALink. It treats all files with an ".o" or ".obj" extension as object files and copies them complete into the executable. Files with an ".l" or ".lib" extension are treated as libraries and scanned, so that only code that is used is included. This is unfortunate because the Oberon-A compiler by default uses the ".obj" extension for all object files it creates. This results in huge executables being created by DLink, containing a great deal of dead code and data.

To obtain the best performance from DLink, the object file for the main program module should have an extension of ".obj", while all other object files should have an extension of ".lib". This can be achieved by changing the OBJEXT setting for OC to ".lib", except when compiling the main module. The OBJEXT setting for OL should then be

set to ".lib .obj" so that it will search for object files with a ".lib" extension first, then for object files with a ".obj" extension. DO NOT GIVE THE MAIN MODULE A ".lib" EXTENSION. This will confuse DLink and the executable program it produces will not run.

1.7 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.

1.8 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 :-).

1.9 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.

1.10 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.
```

1.11 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.