OL.doc

COLLABORATORS			
	TITLE : OL.doc		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 9, 2023	

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: 1.4 $
    $Author: fjc $
      $Date: 1994/08/08 21:01:00 $

Copyright © 1994, Frank Copeland.
```

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

The OL program is a pre-link utility. Its task is to prepare a list of
object files that must be linked together to produce a given program.
This list is output to a file that is input by the linker.


               Distribution
                        Copyright and distribution

Running OL ...

               From the CLI

               From the Workbench

               From FPE

               Program Operation
                   How OL works

               Linkers
                             Using different linkers with OL

               Error Reports
                       What can go wrong and how to fix it
```

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

Usage: OL {option} <program>

Options: {SRC | SOURCE <directory>}
DST | DESTINATION <directory>
ALINK | BLINK | DLINK
LINK
LINKER <path>
OPT | OPTIONS <options>

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:

Given the name of a module, OL attempts to generate a list of object files that must be linked together to create an executable program with the module as its main entry point. If successful, it outputs a file in the current directory containing the list of modules in a format understood by a linker specified as an argument. The file's name will consist of the module name plus a ".with" extension. If instructed to do so, OL will then call a linker to process the ".with" file.

At present, three linkers are supported: ALink (distributed with Commodore's Native Developer Kit), BLink (included with Oberon-A) and dlink (part of the Freeware DICE distribution). To specify a

particular linker, the ALINK, BLINK or DLINK option must be given. If not, the default is BLINK. Only one linker may be specified.

When searching for symbol and object files, OL will look in the current directory first. If any SOURCE parameters have been specified, it will then search those directories in the order they have been given. If unsuccessful, it will then search the "OLIB:" directory, unless the DLINK option is specified.

If a DESTINATION parameter is specified, OL will output the .with file it generates in the specified directory instead of in the current directory.

If the LINK parameter is specified, OL will attempt to call a linker, passing it the ".with" file it created. If a LINKER parameter is specified, the following parameter is taken to be the name of the linker. Otherwise, the name of the linker is determined by whichever of the ALINK, BLINK or DLINK parameters was specified. When OL is called directly or indirectly from the Workbench (ie - from FPE), the LINKER parameter is required to enable OL to find the linker. If the OPTIONS parameter is specified, the following parameter is appended to the command to call the linker. Use this to specify any additional parameters to be passed to the linker. Enclose multiple parameters in quotes.

The CLI stack should be set to at least 10000 bytes. See the STACK command in the AmigaDOS manual.

Example:

```
OL OC SRC Code/ DST Code/
OL ORU SRC Code DST Code LINK BLINK OPT "VERBOSE SMALLDATA"
```

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="DH1:Oberon-A/OL"
Arguments="!P SRC Code/ SRC OLIB: DST Code/"
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

OL cannot be run from the Workbench. Sorry. This will come in a future version.

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

Call OL with the ALINK option. When running ALink, use a command line of the form:

```
ALink WITH <program>.with [options]
```

Using OL with BLink

OL generates BLink format files by default, but it can also be called with the BLINK option. When running BLink, use a command line of the form:

```
BLink WITH <program>.with [options]
```

Using OL with DLink

Call OL with the DLINK option. When running DLink, use a command line of the form:

```
DLink <program>.with [options]
```

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 less suitable for 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 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.

A workaround is built into OL, which helps to reduce the impact of this convention. A more complete solution, involving a change to the compiler, will be available in a future release of Oberon-A. To use the workaround, you must create a file called "OLib.lib" containing the object code of all the library modules normally kept in the OLIB:

directory. This can be created easily by issuing the CLI command:

```
Join OLIB:#?.Obj to OLib.lib
```

You must now make sure that OL cannot find the original object files, either by deleting them, or moving them to another directory. You cannot leave them in OLIB:, because OL will automatically scan it.

When the DLINK option is set, OL now ignores any object files it cannot find, instead of reporting an error as normal. It assumes any missing object files are in OLib.lib, which it automatically includes in the .with file.

1.7 Program Operation

For each module it processes, starting with the one named on the command line, OL goes through the following process:

* It checks that the module has not already been processed. If it has, OL skips it.
* It locates the module's symbol file.
* It locates the module's object file.
* It adds the module's name and the path name of its object file to the list of modules it maintains.
* It opens the module's symbol file and searches for references to other modules. For each one it finds, it recursively repeats this process.

It then locates OberonSys.lib.

If all is well, it creates a .with file and outputs text to it in the specified format.

If the LINK option has been given, it then calls the specified linker, passing it the name of the .with file.

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 as a parameter.

* " !! 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 as a parameter.

* " !! 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 find OberonSys.lib"

  OL could not locate the file "OberonSys.lib", which contains the code for the run-time support system. Check that you have specified the name of the directory it is in as a SOURCE parameter, or copy it to OLIB:.

* " !! 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.