Towards an unified Oberon V4

Claudio Nieder (claudio@dial.eunet.ch)
Ralf Degner (degner@pallas.amp.uni-hannover.de)
1996-08-04

Introduction

(last changed 1996-08-10)

Oberon V4 was ported to several different platforms since it was created. As these implementations were neither created nor later maintained under a strict centralized supervision, the implementation of several modules diverge, even where there is no strict need to. Fixes and enhancements were not always applicated to all implementations.

Triggered by a mesage sent by Guy Laden to the Oberon Developers list, Ralf and I felt that it was time to try to unify the different implementation as much as it is possible, and especially make sure, that all fixes and enhancements introduced in newer Oberon versions, are propagated to all other Oberon implementation. We've decided to produce a proposal of such a system, and make it available to the Oberon community. We will place the module, while we produce them, on some to be defined file server, so that everyone can access them. We will ask current Oberon system maintainers to verify, if our modules are indeed up-to-date and usable on all platforms. If not we will make the necessary adjustments.

This document is our guideline through the work. It records the list of modules which are part of Oberon V4. It will list their current differences and later state the differences which still remain. In the end it will describe the unified modules. So this document will have a quite dynamic nature as long as this project is carried on, and stabilize only, when we are finished with our work. Finally, only what goes into the chapter "The future" will survive as description of the modules in the unified V4. The chapter "The present" will fade out, as it becomes the past.

If you have any remarks or addition to this text, or find some errors, please send a mail to the maintainer of the document, claudio@dial.eunet.ch.

Thank you very much.

PS: As the maintainer of the document, I (claudio) often refer to myself as I instead of always stating my name. I don't know if it is "correct" to do this, but it's in any case easier for me.

The present

(last changed 1996-06-23)

This chapter gives an overview of the existing modules.

List of existing Oberon modules

This is a catalog of the Oberon V4 modules. It describes for each module, the purpose, and if it is/should be specific to a certain implementation of Oberon, or if it is supposed to be generic. DefSpecific means, the the whole module, including the interface is specific to a given implementation. ImpSpecific means, that while the interface is common for all Oberons, the implementation has to be different for every Oberon.

Inner Core

There is one module which interfaces to the basic operating system and is typiclly named after the host OS. The Amiga is somehow special. The AmigaOS is very modular, and this is reflected by the fact, that there exist a whole Range of Modules all named AmigaXxxx which correspond to the AmigaOS libraries.

Module	Type	Purpose
Amiga*	DefSpecific	Interface to AmigaOs, and the Amiga loader.
Console	ImpSpecific	Write to the hosts "console". Used while no viewers available.
Files	ImpSpecific	Perform file operations.
FileDir	ImpSpecific	Enumerate contents of a directory. Exists only on some Unix Oberon.
HostSYS	ImpSpecific	Collects some host specific stuff into one place.
Kernel	ImpSpecific	Memory managment, clock.
Macintosh	DefSpecific	Interface to the Macintosh OS.
Modules	ImpSpecific	Module loader.
Reals	ImpSpecific	Access to floating point representation. Needed by Console and Texts.
Unix	DefSpecific	Interface to the Unix OS.
Win32	DefSpecific	Interface to the Windows 32 bit API.

Module Kernel is actually DefSpecific, because there exist different versions of the module, using different techniques fo garbage collection and for finalization. Modules might also be DefSpecific, because of differences in how the Module structure is exported. This is especially true considering, that some Oberons have implemented the so called "Object Model" while others didn't.

Module Modules, and all modules it imports directly or indirectly, are stored into the boot file. This forms the minimum set of modules needed to load further modules.

FileDir should probably be replaced by the Directories module. Also it is the onlzy module which is not needed for booting, although it is included in the Inner Core. Why? I have to look up the exact distinctions between inner core and outer core, or make up my own ones.

Outer Core

Module	Type	Purpose
Display	ImpSpecific	Driver for the display.
Input	ImpSpecific	Keyboard and Mouse input.
Fonts	Common	Access to fonts.
Oberon	ImpSpecific	Event loop, and basic messaging, and some other stuff.
Texts	Common	The fundamental data type of Oberon.
Viewers	Common	Manage display resource into separate areas.
X11	ImpSpecific	Interface to the X window system. Used on Unix only

Fonts is shown as Common although it is actually ImpSpecific. This is caused by efficiency and usability concerns. Oberon supplies its own fonts, and thus there is no need for Fonts to be OS specific. But displaying characters with the help of the OS is usually much faster. Also it allows to use additionally all fonts supplied by the OS. Of course the use of OS supplied fonts in documents will make them non-portable across Oberon implementations.

Texts and Viewser need at some rare places a call to an OS specific procedure which causes the display to be synchronized.

Additional drivers

Module	Type	Purpose
Display1	ImpSpecific	Additional display functions.
Printer	ImpSpecific	Printer access.
SCC	no comment	Access to a Ceres specific serial device.
V24	ImpSpecific	Access to serial device.

There seem to be different architectures for printer driver. I have not yet understood them.

V24 exists for Amiga, Mac and Windows only. A quick look at the module reveals some differences in the interface. The V24 driver was originally deviced just for the Ceres serial port, so it's interface was probably not designed for being usable on a whole range of different operating systems.

Display1 is implemented using directly the functions of the underlying OS and window system, therfore I've added it to the list of drivers.

Basic user interface

Module	Type	Purpose
MenuViewers	Common	The most used viewers class in Oberon.
System	Common	Implements a basic set of commands.

Actually System is implementation specific, as it performs several commands using the OS interface modules directly. This is done, because lower Oberon modules do not offer the needed functionality in an OS independent way.

Compiler (under construction)

Module	Type	Purpose
OPB		
OPC		
OPL		
OPM		
OPP		
OPS		
OPT		
OPV		
Compiler		

Text System (under construction)

Module	Type	Purpose
Edit	Common	The basic Oberon editor.
TextFrames		

Graphics System (under construction)

Module	Type	Purpose
Draw		
GraphicFrames		
Graphics		

Picture System (under construction)

Module	Type	Purpose
PictureFrames		
Pictures		
Paint		

How to boot Oberon

Loading of Oberon is performed by a helping program. I call this program the boot–loader. This program is written outside of the Oberon system, and runs on the host operating system. It is not written in Oberon, but in C for most implementations (the Obron for Amiga is written in Modula–2). To my knowledge, there are two boot methods in usage, one where the boot loader itself is able to load Oberon object files, and one, where the loader just knows how to load a bootfile. The bootfile contains a prelinked version of the minimum set of modules needed to further load the Oberon system. I consider the bootfile method the one to prefer.

Selfcontained boot program

This is the method choosen in Sun 3 Oberon, and later ported to the Amiga. The boot loader program knows itself where to look for the object files, and about the inner structure of the object file. It thus duplicates the code implemented in module Modules and part of module Files. To avoid duplication of code, Amiga Oberon implements Modules by calls back into the loader program.

Bootfile

This is method choosen for the Unix imlementations except Sun–3. The boot loader program doesn't know anything about Oberon object files. It just loads a prepared boot file and calls a predefined entry point within the boot file.

The bootfile is build by a linker which links a minimum set of Oberon modules. For Unix Oberon these are the modules Unix, Console, Kernel, Files, Modules. Thus there is no code duplication as in Sun–3 Oberon and everything is implemented in Oberon and not partly in the loader language, as in Oberon for Amiga.

The great advantage of this boot mechanism, is that the code which has to be written in a foreign language can be very minimal, and it avoids code duplication.

The future

Project status

(last changed 1996–08–10)

Persons

This lists all persons with which I had contact with during this project. It helps me keep track of who offerd what help, and how I can reach them.

Aubrey McIntosh <mcintosh@ccwf.cc.utexas.edu>

Offered to do Solaris and Windows specific modules. Can test AIX (RS/6000) and MacOS (PowerMac) versions.

Claudio Nieder <claudio@dial.eunet.ch>

I coordinate the effort and will do some programming for Irix.

Doug Danforth <danforth@csli.stanford.edu>

Offered to do Windows specific modules.

Erwin Oertli <oertli@inf.ethz.ch>

Offered to do OS/2 specific modules.

Fritz Engebretsen <newtonst@speakeasy.org>

Offered to test the MacOs versions.

Guy Laden <laden@math.tau.ac.il>

Offered space on his server for UnifiedV4.

John Landahl <jlandahl@sci.nwfsc.noaa.gov>

Offered to test the Irix versions.

Peter Froehlich <p.froehlich@amc.cube.net>

Offered to do work on the compiler after September 1996.

Ralf Degner <degner@pallas.amp.uni–hannover.de>

Maintains Oberon for Amiga.

Status

Module	Type	Status	Last modified	Owner
AsciiCoder	Common	Completed.	1996–08–01	–
ClockElems	Common	Completed.	1996–08–04	–
Console	Common	Completed.	1996–06–17	–
Display	ImpSpecific	Interface extended. Otherwise not yet touched.	–	–
Amiga.HostSYS	SymSpecific	In work. Will be extended as necessary.	?	Ralf
SGI.HostSYS	SymSpecific	In work. Will be extended as necessary.	1996–07–31	claudio
In	Common	Completed.	1996–07–31	–
Input	ImpSpecific	In work.	?	Ralf
Out	Common	Completed.	1996–07–31	–
Reals	Common	Completed.	1996–06–17	–
Texts	Common	Completed.	1996–07–31	–
Viewers	Common	Completed.	1996–06–16	–

The possible values of Type are:

Common	The same source is used across all platforms.
ImpSpecific	Different sources are used, but the implementation differs.
SymSpecific	Although the same objects are exported, the symbolfile differs, because of different constant values.
DefSpecific	The module interface is not the same on all platforms.

When a module is in work, an owner is assigned to the module. Nobody else should modify the module. Modules without an owner are free to be claimed by somebody who wishes to make changes to the module.

Unified V4 Implementation

(last changed 1996–07–31)

Distribution

Traditionally the Oberon distributions were binary distributions. One received a package with all modules precompiled and no source. Because of that, the same modules were contained in every distribution.

As it is now possible to distribute the sources, a different approach is possible. The binary, and platform dependent distribution, will contain the loader, the bootfile, and a minum set of modules, which enables the compilation of further source. The remaining of the Oberon system is made available as source. Thus one sees more clearly, what is common for all platforms.

The minimum set of modules will consist of

- Bootfile with modules "OS", HostSYS, Reals, Console, Kernel, Files and Modules.
- Additional inner core module FileDir.
- The outer core modules Display, Input, Fonts, Oberon, Texts and Viewers.
- The basic user interface MenuViewers and System.
- The compiler modules Compiler, OPB, OPC, OPL, OPM, OPP, OPS, OPT and OPV.
- The helping module Folds and FoldElems. (* FoldElems are useful, but unfortunatly not well integrated. *)
- Maybe some unpacking tool like AsciiCoder. This depends on how the Distributions are made available.

"OS" stands for operating system specific modules. It is an open point, if there is only one "OS" module, or if there should be several modules, e.g. one for the bootfile with the minimum functions needed there, one for the remaining part. Or maybe it would be better to just include whatever is needed directly in the needing modules (Display, Input etc.) themselves. Or even split the implementation specific modules in a Xxxx and XxxxBase, where XxxxBase would contain the OS stuff.

While AsciiCoded files occupy more place then a pure Oberon Text file, there is a great advantage with them. They are text files, which can be moved to almost any Platform without problems. With other kind of distributions there might be problems, especially on the MacIntosh. In any case, only archives which can be unpacked using a Oberon program should be used. This only guarantees, that everybody will have the unpacking tool.

The Modules

Console

(last changed 1996–06–23)

Common implementation for all platforms.

Console exports a small set of procedure which allow a program to write to an output device almost always present in the underlying system. Console is typically used to write messages, which might occur at times where no Oberon viewer is available, e.g. during boot time. On Unix systems it will write to standard output.

Interface

Every procedure prints the value of the passed variable. Bool, Dump, Hex, Int and Real write a space in front of the value. Hex print the value of the LONGINT parameter as an eight digit hexadecimal number.

Dump prints the content of the passed variable in hxadecimal, each byte as 2 digit hexadecimal number. The bytes are printed in the same sequence as they are stored in memory, from the lowest to the highest address. While passign a LONGINT variable with value 1 will always result as 00000001 when printed by Hex, it could be printed as 01 00 00 00 or 00 00 00 01 by Dump, depending whether the machine uses little–endian or big–endian numbers.

Display

(last changed 1996–06–23)

Display has been extended by a procedure called Synchronize. This procedure has to be called after drawing operations were performed, to make sure, that the display seen by the user is up to date. Oberon.Loop will itself call the procedure after the execution of some command, so that normal user commands do not have the need for calling Synchronize.

HostSYS

(last changed 1996–07–31)

(Almost) common interface for all platforms, different implementations. The "Almost" comes from the fact, that although the same constants are available on all implementations, their actual values may differ. Thus the symbol file is not identical.

This module captures those system specificness, which are considere to be important for the "lower OS" only. No user written module should ever have the need to access HostSys.

Interface

BigEndianMachine is TRUE for Amiga and SGI. It is FALSE for ???.

IsNameChar returns true, if the given character should be recognized by the Texts scanner as part of a name.

IsNameChar1 is the similar to IsNameChar, but is used for testing the first character. It generally accepts less characters as IsNameChar (e.g. no numbers).

StdOut prints a string of given length to an output device which should be almost always present, even when Viewers are not availbale, e.g. during boot time. On Oberon implementations based on Unix this would be stdout. On native Oberon implementations this could be some serial port. If no such output stream is available, then it should simply do nothing, but **never** cause an error.

Note: the passed string has not to be 0X terminated. The length passed thorugh parameter len is the only valid indication of the number of characters to write.

toHost transforms a character in the host character set to the equivalent character in the Oberon character set.

toOberon transforms a character in the Oberon character set to the equivalent character in the host character set.

Note: toHost and toOberon are implemented so that toHost(toOberon(ch))=ch and toOberon(toHost(ch))=ch.

In
(last changed 1996–07–31)

Common implementation for all platforms.

Interface

Input
(last changed 1996–06–23)

Common interface for all platforms. Implementation is system specific. Maybe the system specific part is moved to a module InputBase.

Interface

Keyboard input is read via Read. Available tells, how many characters are known to be available.

Mouse input is received through Mouse, which returns the pressed butons and the current position. The maximum x and y values can be limited. The limits are set via SetMouseLimits.

Time returns the elapsed time since system startup. This values unit is 1.0/TimeUnit seconds.

Out
(last changed 1996–07–31)

Common implementation for all platforms.

Interface

Reals
(last changed 1996–06–23)

Common implementation for all platforms.

Interface

Convert/ConvertL convert non–negative REAL/LONGREAL values into a string. Only the non–fractional part is converted. The string contains the value in reverse order and padded with zeroes. Thus converting 123.789 with n=6 return "321000".

ConvertH/ConvertHL convert a REAL/LONGREAL to an 8/16 digit hexadecimal number, representing the internal storage of the value. They are used by the ..RealHex procedures in Texts. They are mostly useful for debugging purpose.

Note: The strings returend by all four Convert... procedures do not have a terminating 0X.

Expo/ExpoL return the exponent (base 2) of the REAL/LONGREAL value.

SetExpo/SetExpoL modifies the exponent (base 2) of the REAL/LONGREAL value.

Ten/TenL return the value of 10 to the power e, where e is of course the passed parameter and has nothing to do with the e=2.718... used in maths.

System and *Host*System
(last changed 1996–06–23)

System implements a basic set of commands. These commands allow the user . The definitio of System should be the same for all implementations. But some implementations might have the need for additional commands. These shall not be added to system, but separated in a moudule names *Host*System, e.g. AmigaSystems, UnixSystems etc.

Texts
(last changed 1996–07–31)

Common implementation for all platforms.

Interface

Viewers
(last changed 1996–06–23)

Common implementation for all platforms.

Interface