Some notes on WinAos

Felix Friedrich
Computer Systems Institute, ETH Zürich, Switzerland
friedrich@inf.ethz.ch

February 11, 2009

1 What is WinAos?

WinAos is an emulation of the operating system Aos ("Native Aos") on Windows. Native Aos is an operating system written in and for the programming language Active Oberon that supports the programming of multithreaded applications by ways of so called Active Objects. Aos has a zoomable user interface and contains a lot of sample applications, in particular it provides editors, some multimedia components and an Integrated Development Environment. Aos is the successor of the Oberon system, a single-threaded operating system with a very powerful and interesting GUI. Oberon is still supported as one (thread) of many windows in the Aos GUI.

WinAos is the successor of (and initially based on) the ETH Plugin Oberon for Windows by E. Zeller. A lot of functionality of this Oberon emulation system was kept and is still contained in the system. Besides the pure emulation of the Aos GUI, it is therefore also possible to use it for seamless integration of the Oberon system in Windows, where for instance Oberon frames can be displayed and used like Windows windows. This dual nature of WinAos is also reflected in the two different implementations that come with a WinAos system and which can be selected by the user, cf. Section 5.

2 Purpose

The purpose of WinAos is to provide the functionality of the Aos operating system on Windows-based computers. This is achieved by a replacement of all very-low-level functionality of Aos by calls to the WinAPI library. The WinAos kernel is very much interface compatible to the kernel of the native Aos system and, together with some in- and output modules, provides the necessary translations to the Windows API.

3 Compatibility

Since the base modules in WinAos are supposed to be interface-compatible with the ones of Aos, in general Modules that can be compiled within Aos can also be compiled within WinAos. Applications that run on Aos do normally also run on WinAos.

Of course there is low-level functionality of Aos that cannot be used under Windows in the same way. Examples are the direct access to devices (such as the USB subsystem) as these are shielded by windows from the user. In these cases workarounds normally are available such as the access of USB connected hard-disks via the file system of the Windows system.

4 Work- and Search-Path

WinAos uses the file system provided by the Windows system and therefore - unlike Native Aos[1] - immediately features the use of directories. In WinAos an ordered set of search paths and a working path can be specified in a configuration file (cf. Section 7). Whenever a file is looked for without specification of a relative or absolute path, it is first searched in the working directory and if not found there the search paths are traversed in the order in which they have been specified. Whenever a file is written without specified location it is written to the (current) working path.

[1]Naturally different file systems can be mounted from Aos but the native Aos file system does not support directories

To understand this is particularly interesting and important for understanding the usage of object-files. As an example consider a work path being specified as `C:/MyWork` and search paths being specified as

`C:/Aos/WinAos/ObjE; C:/Aos/WinAos/Aos`.

Assume you use, modify and compile a module file, say `PCP.Mod` (a part of the compiler, by the way), the first time. Then at its first usage the object file `PCP.Obw` may not be contained in your work path but, say in one of the search paths, and therefore is loaded from the search path in memory. Now compiling `PCP.Mod` results in writing a new object file to the work path. The next time you restart the system (or unload the module) this file will be available on and taken from the work path.

In effect this means that if you update your WinAos system by exchanging everything but the Work path you may still be using 'old' object files resident in the work path. Therefore it is wise to remove old object files from your local work path whenever you update WinAos.

5 Internal and External View

WinAos comes in two different configurations, one of them being the 'internal' the other being the 'external' version. The major difference is that the Oberon subsystem UI of Aos is started as an Aos window in the first case while it is started as an (external to Aos) Windows window in the latter case. The advantage of the first case is that it is absolutely compatible with native Aos and therefore is suited best for the typical Aos system programmer while the external version can be customized to be much closer to a windows look and feel and subsystems (such as the ants software platform) even appear as ordinary application windows within Windows OS.

As the external version needs a separate handling for the Oberon (Windows-)windows it is not binary compatible with the internal one. This is the reason why only the one or the other can be run and not both versions can be compiled to the same system. To be still able to choose between the two, the different versions are compiled in separate directories (note that naturally they share the same kernel) and just incorporating the one or the other directory in your search paths makes up the decision between the internal or the external version. The search path can be modified in the configuration files `aos.ini` and `myaos.ini`, cf. Section 7.

6 Configuration of the Oberon subsystem

The configuration of the Oberon subsystem is contained in a file whose file name is specified in the Aos configuration file. For the internal version this normally is `Oberon.Text` while it usually is `OberonExternal.Text` for the external version.

7 Configuration of WinAos

WinAos is configured via the file `aos.ini` (and, optionally, `myaos.ini`) which has to be located in the same directory as the executable `Aos.exe`. This file is read at startup and is mandatory for a functioning WinAos system. It contains an optional line pointing to an alternate configuration file, a specification of the search path, the work path, the default object file extension, the Oberon configuration file name and a (sequence of) commands that is (are) executed at startup. A tilde in a separate line ends the configuration.

A configuration line is of the form

```
identifier = "value"
```

Here identifier currently can be one of `AlternateConfig`, `Paths.search`, `Paths.Work`, `Defaults.extension`, `oberon` and `cmd`. If two configuration lines have the same identifier then the first line is taken for configuration!

A typical configuration file looks as follows

```
; external, starting with Oberon
AlternateConfig = "myAos.config"
Paths.Search="Work;ObjE;Src;Src/vyants;Aos;../source;Doc.vy.ants"
Paths.Work="Work"
Defaults.Extension=".Obw"
```

```
oberon="OberonExternal.Text"
cmd="SEQ Oberon.DoStart;AosFSTools.Mount WORK WinRelFS ./"
~
(rest ignored)
```

In this example first the file `myAos.config` is read in, then the search path is set (if not already done in myAos.config!), the work path is set (if not ...) etc. Note that this configuration is a typical set up for the external version. The `Defaults.Extension` entry is particularly interesting, if you want to build a new version with different object file suffix (for example to be able to fall back to previous versions if modifications are complex).

If, together with the above example, the file `myAos.config` just contains the following line

```
Paths.Work="/tim/Work"
```

then everything from the file `Aos.Text` is kept for configuration but the work path is set to the user's work path /tim/Work.

8 Building a new WinAos system

Building a WinAos system consists of compiling all necessary source files and linking the kernel. The steps necessary are contained in the files `Win32.Aos.Tool` (external version) and `Win32.Aos2.Tool` (internal version). The `Release.Build` command contained in the respective Oberon-text files opens a script file to compile the release. Note that the compiler option `\s.Obw` determines the used suffix (.Obw) and `P/Aos/WinAos/ObjE/` determines the out- and input path (/Aos/WinAos/ObjE/) for the compiler. To link the kernel to an executable file, execute the command

```
PELinker.Link \.Obw \P/Aos/WinAos/ObjE/ Win32.Aos.Link
```

being also contained in the file `Win32.Aos.Tool`. The executable will be generated to your work path. The file `Win32.Aos.Link` contains the files to be linked and other directives to the linker. It is out of the scope of this document to describe the PELinker in more detail.