Porting Unicon to Windows Mobile 2003

David Price
November 4, 2005

Abstract

This report summarizes the port of Unicon to an operating system intended for mobile computing, Windows Mobile 2003. It describes the features available, unavailable and possible on Windows Mobile 2003 for the Unicon programming language. It also covers building Unicon for use on Windows Mobile 2003 and running Unicon on Windows Mobile 2003.

1.0 Introduction

Handheld computing devices on the market range in capability from simple calculators to processors whose power exceeds desktop computers from recent memory. Handheld processing devices are usually coupled with other familiar devices, such as: cell phones, drawing tablets, input scanners, cameras and Global Positioning Systems. Mid to higher end devices feature operating systems, processors capable of several hundred megahertz, storage on the order of 32 to 64 megabytes, graphics displays with resolutions of 240x320 to 640x480 pixels at 16 bit color (some may even feature 3D graphics) and have networking allowing them to communicate with the internet, desktop computers and other handheld devices. Typically, these devices include software for web browsing, emailing, mp3 and other multimedia playback, simplified word processors, spreadsheets and instant messagers of various types.

Two of the largest operating systems makers, in terms of devices and users, for handheld computing devices are from Microsoft, with its Windows CE derived operating systems, and Palm, with several versions of PalmOS. Many different versions of these operating systems are available on a wide variety of devices from both companies. Currently, Microsoft provides OSes including: Windows Mobile 2003, Windows Mobile 2003 Second Edition, Windows Mobile 2003 Phone Edition and Second Edition Phone and Smart Phone 2003, all based off of Windows CE.net 4.20. Current Palm devices generally run PalmOS versions 4.x, 5.0, Garnet(OS 5.x) and Cobalt(OS 6). Microsoft and Palm also provide Software Development Kits to facilitate application programming for their operating systems and devices, oriented towards the C and C++ languages.

While C and C++ are very popular mainstream languages, they are at the low level end of high level languages and are can be affected by memory and hardware implementations. Programming for small devices with rapidly evolving, reduced functionality operating systems and feature sets may have many more subtleties than a desktop machine with a mature operating system and feature set, proven by thousands of programmers and millions of users. A high level language which hides as many of these subtleties as possible would allow more users to program their devices.

1.1 Unicon

Unicon is a very high level programming language created for rapid prototyping and experimental software development, available from http://unicon.org. It has a wide range of features including graphical, network, database, and multimedia. It is powerful enough to set up virtual worlds where people can interact in simulated real-world environments[42]. Unicon can be compiled and run on most linux-or-unix-based operating systems, and Microsoft Windows desktop operating systems. Unicon consists of a compiler, which creates machine independent instructions from human coded files. These are then turned into virtual machine instructions for the runtime (virtual machine) by a linker. The runtime is small enough (under 4mb, can be built to be well below 1mb) that many small handheld devices can store it without trouble. Unicon has been ported to many operating systems, including: Linux, Unix, Windows 32, Mac OS X. Additionally, the language Unicon is based on Icon [12], has been ported to Amiga, Acorn Archimedes, VAX/VMS and MS-DOS. However, the Icon graphics facilities are not available on all ports. More information on Unicon is available from the Unicon books [14][9], or Unicon technical reports[41].

1.2 Windows CE, PocketPC, and Windows Mobile

The Windows CE .NET operating system is a 32 bit OS, designed to run on systems which have memory, processor and power constraints, but also to provide as many features as possible.

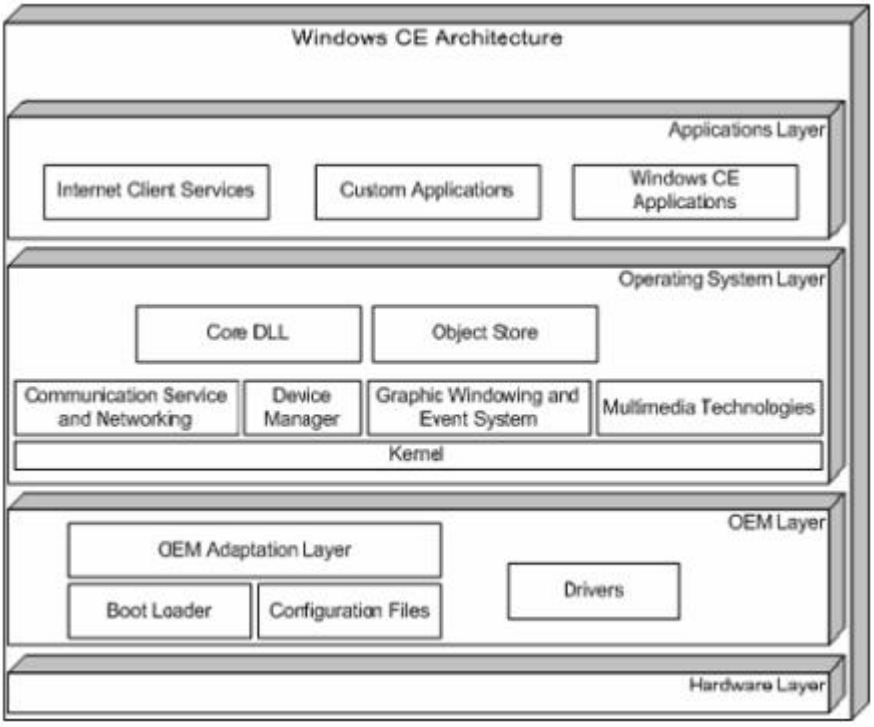


Figure (1) Windows CE operating system layers [19][20] (2003 Microsoft Corporation)

It was also designed to be quickly portable to new hardware, and regionally independent. In 1996, the first version of Windows CE was released, followed shortly by Windows CE 2.0 a year later. Currently, Windows CE 5.0 is the latest iteration. The Pocket PC 2003/Windows Mobile 2003 and Windows Mobile 2003 Second Edition (for Pocket PCs, 2003) OS is derived from the Windows CE .NET 4.20 operating system. Windows Mobile 2003 Phone Edition provides Windows Mobile 2003, with some improvements in internet access and phone usability. Microsoft SmartPhone 2003 OS is intended for devices that are less pocket computer and more phone. Typically, smart phones have smaller resolution screens, no touch screen and fewer hardware expansions available. Phone service providers can require that any third party software for SmartPhone 2003 be digitally signed by them before it is installed on a device.

Windows CE/Windows Mobile/SmartPhone operating systems feature cut down versions of standard C and C++ libraries, Microsoft ATL/MFC API's, and the Windows API. The functions removed from Window CE API's generally fall into two categories: those that are truly missing because of limitations in the hardware or and those that were redundant and nonessential. For instance, on a 240x320 pixel screen, maximization and minimization of windows has been done away with. If there were two functions in the API (or even repeated in the standard API/MFC/ATL) which would accomplish roughly the same thing, quite often one of them has been removed, and if necessary, the remaining function may have had its parameters

expanded in order to replicate what the removed function(s) did.

The Windows Mobile 2003 OS has a large variety of features including graphics, networking/wireless, sound and power management. The Windows APIs allow access to these features. A good deal of code for Windows 95/98/XP/2000 is compatible with Windows Mobile 2003. However, there are some major points of non-compatibility.

1.3 PalmOS

Palm OS was first released in 1996 on the Palm Pilot. PalmOS 5.x/OS Garnet is the newest and most widely used version of Palm OS. (While Palm OS 6 is available, very few devices running it are available on the market). Garnet is a 32 bit native operating system, also designed to run efficiently on power, memory and display size constrained systems, while providing a large number of features. The Garnet OS has only a 300KB RAM requirement for the operating system. It has support for variable length strings by default, and the character set used varies by region[27]. It has its own API, the Palm OS API. This API provides functionality similar to the Standard C library with additions. Additional functionality includes the ability to create graphics and window UI's known as forms and networking via wired and unwired connections, sound playback and recording and power management. From a high level perspective, the design goals and supported features of the Windows Mobile and Palm operating systems are very similar.

1.4 PalmOS Versus Windows Mobile 2003

Smart phones and pocket computers running Palm OS have been much more popular than Microsoft PocketPC devices historically, but that may be changing[33][43]. However, PalmOS is still very popular and has a very large market share. The visible features of these two operating systems look very similar (display sizes, memory size configurations, networking capabilities), looking deeper, they are not as similar.

Devices running Windows Mobile 2003 have some advantages for a Unicons port. Windows CE has a local stack size which defaults to a size of 58 Kbytes and is growable up to a size of one Mbytes [3]. PalmOS version 3.5 and higher devices have a default stack size of 3.25 Kbytes[28][34]. The heap on Windows CE C programs defaults to 188kbytes, PalmOS 3.5 and greater has a 256kbyte (assuming more than 4mb of RAM) dynamic heap, which all dynamic allocations must come out of, including the TCP/IP stack, global and static variables and the application stack. This means that Windows CE should handle larger and more complicated programs much more easily. Related works support this conclusion (see section 2.0).

Additionally, Palm OS does not use a traditional file system. The file system is flat, files are stored in “chunks”, which have a fixed size (generally slightly under 64k). These chunks contain a header with directory and file information, and the remaining portion of the record contains the file contents. Each chunk is a record in a database which acts as the file system. When transferring files they must be broken down into records when being put on the device, and translated back into a “real” file when being transferred to desktop machine, or device with a different file system.

Finally, Unicon has been ported to Windows 32, since Windows Mobile 2003 API is a

subset of Windows 32 API, a port of Unicon to WM2003 should be much more straightforward than a port to a completely new operating system.

1.5 Development Tools

Currently available for development of Windows Mobile 2003 projects, for free, are Microsoft eMbedded Visual C++ 4.0 (EVC4) [21] and the PocketPC 2003 SDK[23]. EVC4 is a Visual Studio like program, which together with the PocketPC 2003 SDK allows compilation of code to be used in PocketPC 2003 devices with ARMV4 compatible processors and the PocketPC 2003 emulator, which comes with the SDK.

The emulator is a very useful tool, since code can be compiled and automatically downloaded to it for testing and debugging. The debugger can be attached to processes running on the emulator, or automatically run when the program is downloaded to the emulator. The debugger allows easy right click addition of break points to code in the editor, but so far the move forward/backward in execution seems fairly primitive, no backing up and small forward steps.

Figure (2) PocketPC 2003 Emulator

2.0 Related Work

There are many projects that are related to porting Unicon to the Windows Mobile 2003 operating system. Programming languages or compilers that have been ported to Windows CE derived operating systems include Ruby, GCC, Tcl/tk, Java and Perl. All advertise being able to compile, if needed, and run on the devices that they support. The port of Unicon to Windows 32 is also available as a related project similar to porting Unicon to Windows Mobile 2003. Additionally, an older port of the Icon programming language to a device known as the Atari Portfolio is of some interest, as it is perhaps the smallest device that Icon/Unicon has been run on.

Ruby is a completely object-oriented programming language with strong string manipulation capabilities. It includes built-in CGI and networking capabilities. The core of Ruby does not support graphics or GUIs, but add-ons using tk and gtk can provide these features, however they do not appear to have been ported to Windows CE. The Ruby language is interpreted, a text file with Ruby code is directly executed. Ruby has been ported to PocketPC 2002[35], nearly all core features found on other platforms are available. It can be built using Microsoft Embedded Visual C++ 3.0.

Java is an object oriented language that is compiled and run via a virtual machine. Many official and unofficial ports of Java to Windows CE have been done. Sun's PersonalJava is an official port written for Windows CE 2.11, however it has entered end of life and has been replaced by Java 2 Micro Edition[37]. Additionally, IBM (Websphere), Esemertec (Jeode), and HP (ChaiVM) as well as a host of other attempts. The Windows CE 2.11 official PersonalJava port by Sun had functioning AWT and native compile, though threading and many large data structures caused the VM to crash[8].

Perl is also an interpreted language with strong string manipulation abilities. Perl has no built in GUI capabilities. The currently available revision provides many of the features of Perl for other operating systems, but threading and pipes are weak, `fork()` seems unimplemented[16][31]. It can be built with Visual C++, and binaries are available for several processor types. Perl for CE uses a user written library known as celib[15] to replace some missing functions.

Tcl is another language that has been ported to PocketPC, along with its GUI creation package, tk. Tcl is an interpreted language, most Tcl and tk features are present on Windows CE. The celib library came about during the port of Tcl/tk. TCL and Perl use the celib library, written by Rainer Keuchel. This library is a replacement for a great number of the missing standard C functions and functionality. However, this library was written for PocketPC 2002 and earlier versions, it is not recommended for use with Windows Mobile 2003, since a large number of standard C library functions present in celib were added by Microsoft in that release. Ruby for PocketPC 2003 takes a different approach, it has a \wince directory in the source tree, which contains replacement code for only the missing headers and functions required by Ruby.

The Unicon port to Windows 32 provided a good starting place for a port to Windows Mobile 2003, as the APIs for the two operating systems are very similar. The Win32 port supports all of the major features of Unicon and can be built with gcc for Windows, or Visual C/C++.

The Atari Portfolio was an MS-DOS based device, a port of Icon was done in the early 1990's to this platform. An older, smaller memory footprint version of Icon than what was

currently available was used, with reduced memory regions as the device had very limited memory. The base device did not have enough memory to perform on device translation and linking, but eventually using add-in memory, some on-device translation and linking was accomplished[1].

Languages that have been ported to Palm OS include:

- Java, in the form of J2ME IBM WebSphere Micro Environment runtime, [24] and several other unofficial runtimes. No on device JDKs appear to exist, all programs must be compiled on a development machine.

- Lisp (LispMe) [2], with simple graphical/GUI capabilities and much of the functionality of desktop ports.

- Forth (DragonForth) [7], with the ability to program and execute on device

- Basic (SmallBasic) [36], has features including compile and execute on device and simple graphics.

- Python (Pippy) [32][5], with an abbreviated feature list, no floating point, no file IO, no parser, no compiler and Pippy itself can not take user input.

- Tcl, Palm-Tcl [30] and Toucan [4] , has some support for tk, Tcl's famous graphics package, but no on device compile.

These languages and their features tend to support the idea that devices running Windows CE derived operating systems handle larger and more complicated programs much more easily than PalmOS. Most of the languages ported to PalmOS are either simple languages, such as Basic, or have many of their features removed, Pippy, Tcl, for reasons involving limits on PalmOS.

3.0 Implementation

After spending some time learning the basics of project creation and manipulation in EVC4 and how the emulator worked, a project was set up in EVC4 to begin work on porting the Unicon runtime, Iconx, to Windows Mobile 2003. First, the configuration files already present in Unicon under config\win32\msvc were copied to config\winCE\evc4 and config\winCE\msvc.

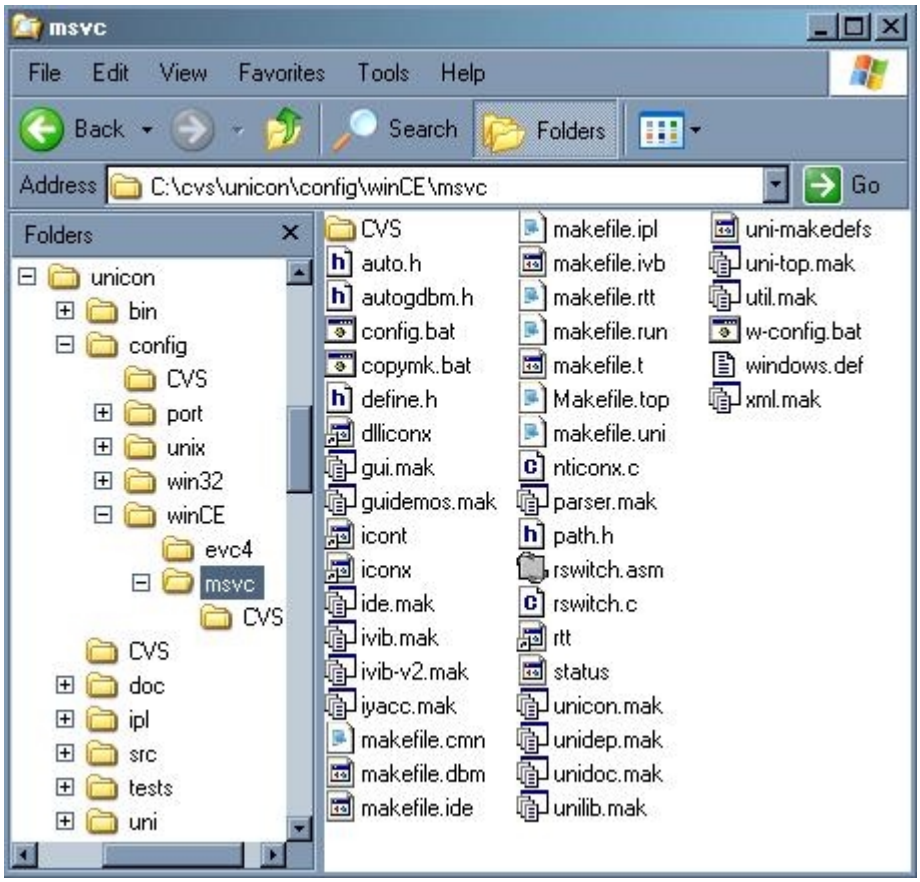


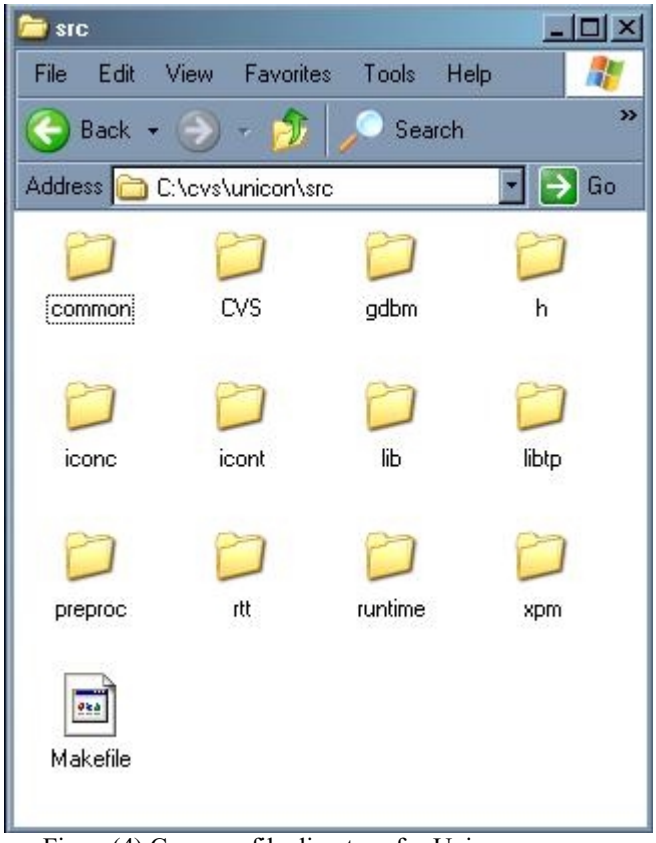
Figure (3) New configuration directories for Windows Mobile

The Makefile used in the base Unicon directory (Makefile.top) had a new build target, WCE-Configure, added. This new build target copies configuration files and makefiles to each Unicon subdirectory from the config\winCE\msvc directory. The Makefile for the runtime directory (makefile.run) was edited so that the build terminates early.

The Unicon source file directories contain code used in the build of Unicon tools, such as iconx, the runtime, and icont, the translator. Most files in these directories contain C code, there are some Unicon language files and assembly language files.

- Common – Contains generic functions used in several Unicon programs
- Gdbm – Gnu DataBase Manager library
- H – header files used by files in the other directories
- Iconc – Unicon compiler, creates executable C code from Unicon source code
- Icont – Unicon translator, creates “icode”, instructions for Unicon VM
- Lib – Contains code built into library files linked in the build of Unicon programs
  - Lib\wince and lib\gdbmce important to build for WM2003.

- Libtp – LibTP library, used for messaging extensions
- Preproc – RTT preprocessor
- Rtt – Contains files to build the program rtt
- Runtime – Contains files processed by rtt and built into iconx
- Xpm – XPM pixmap library



Figure(4) C source file directory for Unicon.

An important part of Unicon, particularly to this project, is the program rtt. Rtt is a program which translates .r and .ri files, which are close to C language but have some special extensions, into C files that can be compiled. The C files generated from .r and .ri files in the Unicon source code make up most of the runtime, Iconx. The runtime executes icode files, generic "machine code" files compiled from files written in the Unicon language.

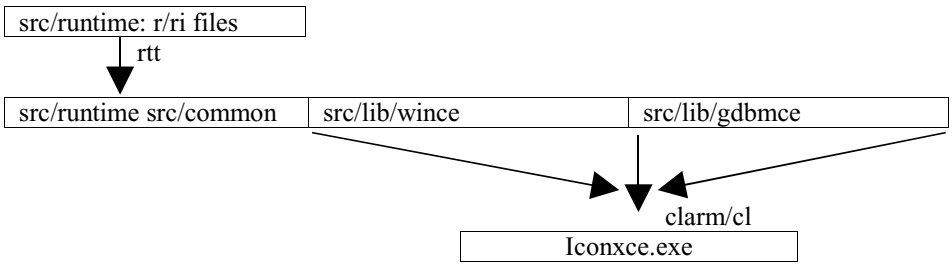


Figure (5) build process, clarm generates device executable, cl generates emulator executable

The build terminates after rtt translates the files in the runtime directory from .r and .ri files into C files, rather than after all of Unicon is built. This stage of the process used Visual Studio 7 and its tools; the rtt used to translate the r and ri files is for Windows 32, it is unmodified. The C files generated by rtt were added to an EVC4 project.

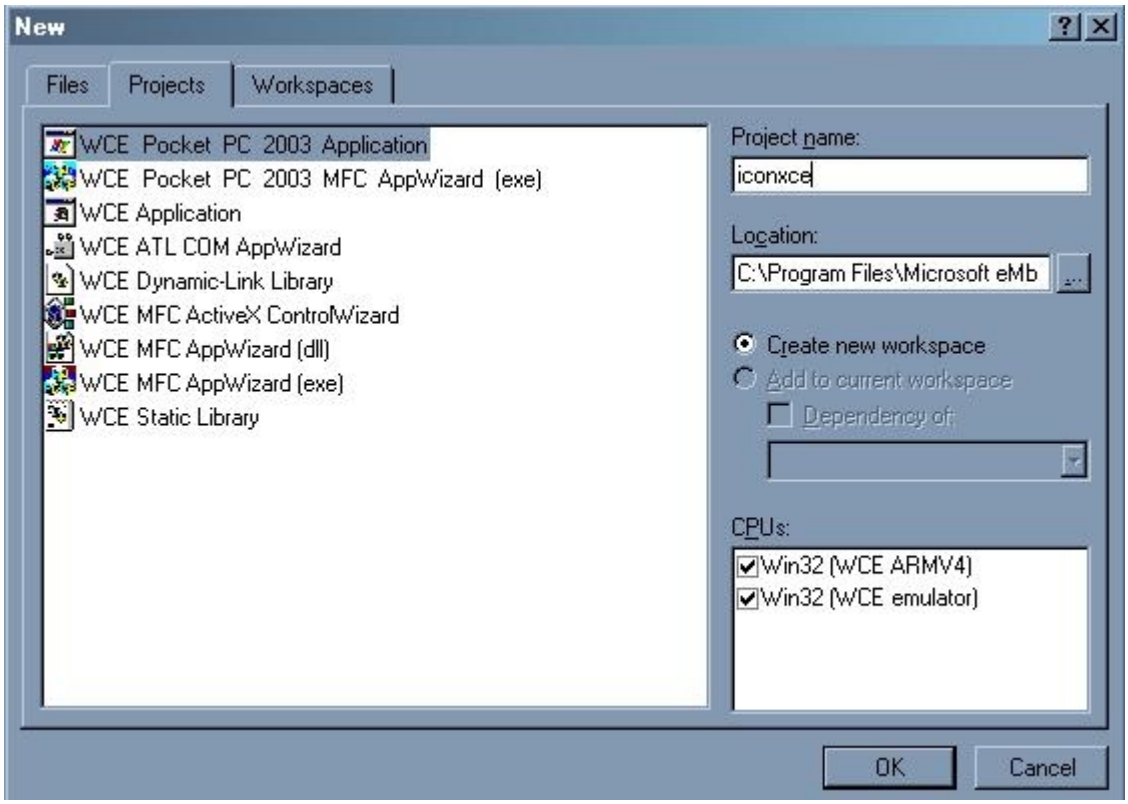


Figure (6) Iconx project for Windows CE

This project was created inside the IDE for a number of reasons. The first reason being to use and learn the EVC4 IDE. The IDE allows quick download of the compiled files to the emulator and device, interactive debugging and communication with the running process on both testing platforms. While most changes were done to the r and .ri files, rather than the C files added to the project, the IDE uses the C files in place, when they are updated by running the Unicon build process, the IDE automatically loads the new versions. The IDE can export its makefiles, it is possible to set up a simple "nmake WCE-Configure" option from the command line. This make option would most likely use Visual C to generate the C files from r/ri files using rtt, and then run the EVC4 command line tools to build the executable.

As a side note, all changes to files in the Unicon directories /runtime, /config (to a lesser extent, since it is a Unicon for WinCE directory, /common files (generally all changes but those done to added files, such as /wince) are behind #ifdef WINCE or #ifndef WINCE preprocessor commands.

3.1 Standard C Library Problems

Because many standard C headers are missing from WinCE, the port must provide replacements for missing functionality. Since this seems like a problem that could be very common among software ports to WinCE, a search was begun for solutions. The Ruby project was found to have similar requirements to this project. Ruby had placed its replacements in a directory under the source tree named 'wince'. The replacements they had created and borrowed were freely available for use/copy/modification, and supposed to work with PPC2003. However the replacements were for PPC2002 under WinCE 4.0 (PocketPC 2002), not WinCE 4.20 (Windows Mobile 2003), the current version.

For the Unicon to WinCE port, a src\lib\wince directory was created and a \sys was added under this path.

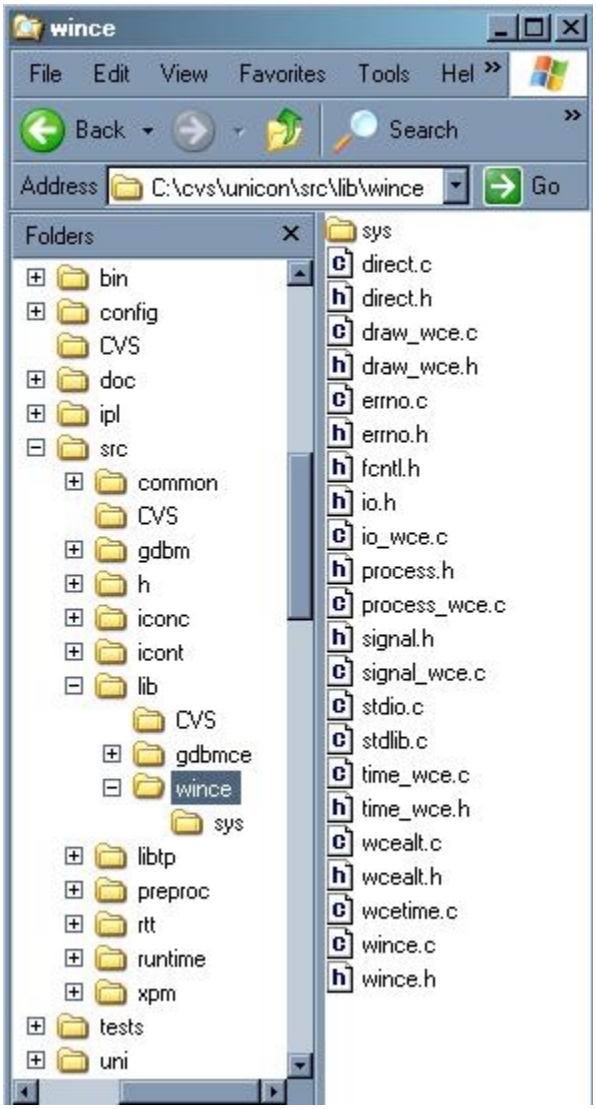


Figure (7) Library for missing standard C and Windows 32 API functions

These files were added to a new EVC4 project and compiled as part of the iconx project.

This project was setup as an empty library named wince_stdc. It was inserted into the Iconx workspace, and a dependency on this project was added to the Iconx project. See appendix C for a full list of functions provided by the wince library.

3.2 Configuration Changes to Unicon

Portions of Unicon that are not necessary for basic function were disabled, at least for the first iteration of the project. Options taken out through define.h:

Messaging (NoMessaging)
Dbm (NoDbm)
ISQL
PosixFns (NoPosixFxns)
Coexpr (NoCoexpr)

Some of these options may make their way back in over time (The current version has Dbm and some posix functions have been implemented), but they are mostly unnecessary to the core functionality of Unicon.

3.3 Unicode

Perhaps the most visible change when porting is the fact that all text arguments to or from Windows Mobile API function calls use Unicode, rather than ASCII. Windows Mobile is all Unicode for two related reasons. The first is that, as mentioned before, the OS was designed to be regionally independent. Unicode contains many alphabets, with its ability to represent over 16,000 characters, where ASCII is limited to 256. Unicode would have to be implemented anyway for multi-region support. The second reason for the lack of ASCII calls is to save on space and overhead. The addition of the extra byte to every character does add some overhead, but it is an amortized overhead. The programmer can declare ASCII characters, but they will not be usable for any but the code they write.

The most pervasive change in the code for Unicon was changing all Windows API function calls that require character or string parameters or have return values of character or string. This port does not convert Unicon to support Unicode, Unicon still uses ASCII for internal representation of strings and characters. The method of changing these function calls to use Unicode is straightforward. Two helper functions and a macro provide most of the ability to convert ASCII strings and characters to Unicode for calls into the WindowsCE API.

`wce_mbtowc`

Any string variables passed from Unicon (ASCII) into WinCE API (Unicode) functions or assigned to WinCE string structure data members are wrapped with a call to `wce_mbtowc`. Its implementation calls the WindowsCE function `MultiByteToWideChar` twice. The first call returns the length of the ASCII string passed to it, the second uses the retrieved length to convert the string to ASCII.

TEXT

All constant strings passed into Windows API calls are wrapped with the macro, TEXT or _T, which the preprocessor converts to the symbol that is used to denote a Unicode string. So, "foo" becomes `TEXT("foo")`.

`wce_wctomb`

The function `wce_wctomb` has behavior identical to `wce_mbtowc`, however it uses the WinCE API `WideCharToMultiByte` function, taking a Unicode string and giving back an ASCII string. It is used for any WindowsCE API function that returns a string.

3.4 Additional Differences

Besides removing ASCII support from the Windows Mobile 2003 programming API's, there are many other underlying changes. Graphical calls such as: `Arc`, `Pie`, `CreateDIBitmap`, `CreateBitmapIndirect` are missing, the port must deal with this in some manner for graphics to properly work. There are no environment variables, though reading and writing registry entries as a simulation can be done. There is no sense of a current working directory; paths are specified absolutely. WindowsCE also removes an older set of media controls, MCI (Media Control Interfaces), used primarily for MIDI play back in Unicon, and a large portion of the C standard library. Several files in the Unicon runtime source code directory and the common directory were modified due to API changes from Windows 32 to Windows Mobile 2003 such as described above. Several new files were added to the standard C replacements to because of these problems. Further discussion of what was added, changed and removed can be found in Appendix A.

4.0 Testing

The Unicon test program gpxtest.icn from tests/graphics and the tests in the general category were run on the PocketPC 2003 SDK emulator and hardware. The only appreciable difference found, was that the device could not handle Unicons coexpressions, since they are written in x86 assembly instructions presently.

Device Technical Specifications:

Dell Axim X50
Intel PXA270 ARMV4 Processor at 416 MHz
64 Mbytes SDRAM, 64 Mbytes ROM
240x320 Resolution, 16 bit Color
WM8750 Sound Chip, 16 bit stereo, up to 44.1KHz
Windows Mobile 2003 Second Edition

Gpxtest.icn was the first program tested. Once the initial start up command line bugs related to translating back and forth between ASCII and Unicode strings were ironed out, it did not appear that the test program was being run correctly. However, after a few days, the window for the test program was found to be popping up behind currently open windows.

Figure(8) simple, lines, rects, star, pretzels and spiral

Figure (9) arcs, copying, rings, fontvars, stdfonts, stdpats

Figure (10) patts, attribs, gamma, balls, slices, details

Figure (11) rainbow, whale, cheshire

Figure (12) dialog test

Once that was determined, gpxtest.icn was copied to gpxtestce.icn, where six tests at a time are displayed (on a 3x3 grid). Most of the tests run properly, with the exceptions of: the star, it has a black center when compared to the results on Windows XP; the slices test (uses `Pie`) the pattern tests; setting a brush to BS_PATTERN seems to be supported, but there are apparently extra issues related to drawing them on WinCE versus Windows 32. The code for creating a pattern brush seems to compile without error. Some of the attribs tests (posx, resize, …) did not work due to lack of support on the hardware and were disabled.

The general tests could not be compiled and batch run on the emulator; they needed to be compiled as .cmd files. The icn files were built to Unicon icode .cmd files using a Perl script. A Unicon program called unicon_tests.cmd was created; the names of all test programs were entered into this program as a list. Iconx resides in the root directory, and for every test program in the list, unicon_tests runs:
`popen(cmd, "p")`, where cmd is "\\iconx \\tests\\" || name || ".cmd > " || name || ".txt". The output text files were compared to the .std files for each program using diff –bu.
The results of this test are in Appendix B.

The emulator, and the device seem to have some trouble running many programs in rapid succession, particularly when they are somewhat high memory or high I/O resource or computationally challenging. There may also be memory leaks in Iconx related to porting it to PocketPC. There may be redirection buffer limits that are set lower than they should be, perhaps because Unicode characters take more space and a buffer size is set in bytes. Several programs did not produce output, or produced truncated output that provided correct results when run by hand rather than run as a batch.

5.0 Partial Implementation of Advanced Unicon Features

The goal for the project was most of Unicon, to have the features most used in Unicon programs portions available . A few of Unicons advanced features (described in section 3.xyz) are currently unimplemented, or partially implemented. A summary of the states of these features is as follows.

5.1 Posix

Right now, posix functions as a whole are not being built for Iconx on PocketPC. There were many errors in the compile stage when attempting to build posix functions in. Many of them are no doubt easily implementable; it was easier to set them aside as whole to begin with. `Stat, gettimeofday, send, receive, fetch, rmdir` and `mkdir` should work.

5.2 Messaging

Messaging is currently disabled, but Windows Mobile does support Winsock. The posix functions `send()` and `receive()` have been implemented and the project is linked against the Winsock libraries.

5.3 Coexpr

Assembly code for Coexpressions context switching needs to be written and the EVC4

compiler does not support inline compilation for the ARMV4 processor[22].

5.4 ODBC Support

Most ODBC support seems to be through MFC, and that support has a truncated list of functions compared to what "big" Windows can handle. It is stated that Windows CE does not support ODBC and that MFC for Windows CE does not support ODBCException in the documentation for PocketPC 2003. There seems to be a few third party pieces of software, which allow devices to synch with ODBC databases around. Sybase software[38] claims there is no ODBC driver manager for Windows CE, but File Data Sources can be created on desktop Driver Managers and copied over. This information is old (2000), but still seems to be the case.

5.5 DBM Support

DBM (gdbm 1.7.3), was ported to an older version of WinCE by DejaVu[6] software. This code is freely available and free to use. This WinCE version of GDBM was setup as a new EVC4 library, it was inserted into the Iconx workspace. The C files for Iconx were rebuilt using MSVC, with DBM enabled. The new library, called gdbmce, was made a dependency for the build of Iconx. The gdbmce library successfully compiled, but a few problems cropped up when it was linked in the build of Iconx.

To make GDBM work on Windows CE, DejaVu Software created a file called cehack.c and cehack.h, the function of these files is similar to the files in wince_stdc for the Unicon project, cehack files provide missing standard C library functionality. This led to some naming conflicts between files in wince_stdc and gdbmce, the files of the latter now reside in lib/gdbmce. To resolve the conflicts, conflicting function headers and function bodies were commented out of cehack files and the gdbmce library is linked against wince_stdc when it is built. The function `read` in the gdbmce implementation provided some error checking and functionality above that found in the wince_stdc version, and the wince_stdc `read` was replaced with the gdbmce version.

5.6 Compiling of Unicon Programs

Currently, there is limited self-hosted compilation for Unicon programs under PocketPC. To simplify matters, the Unicon program generated should is not linked to the Unicon runtime, Iconx. An icode only file (generated by giving the output file a .cmd extension) is generated with Unicon. This means that Unicon programs can not be double clicked to run, Iconx must be run with the cmd file as an argument. It should be possible to generate an icode only file using the Unicon compiler on any Windows machine, then run that icode file with Iconx on Windows CE. It is not necessary to set up a special cross compile linker on the host machine to bundle Iconx for WindowsCE with the generated icode.

5.7 Display Considerations

On the small (240x320 or 320x240 generally) screen of a PocketPC, efficient GUI design becomes very important. The average full screen or even partial screen GUI designed for a desktop PC is too large to be fully displayed. Even non-GUI programs can be difficult to display. On a 240x320 screen under WM2003, even output to the Unicon console can be too large to fit.

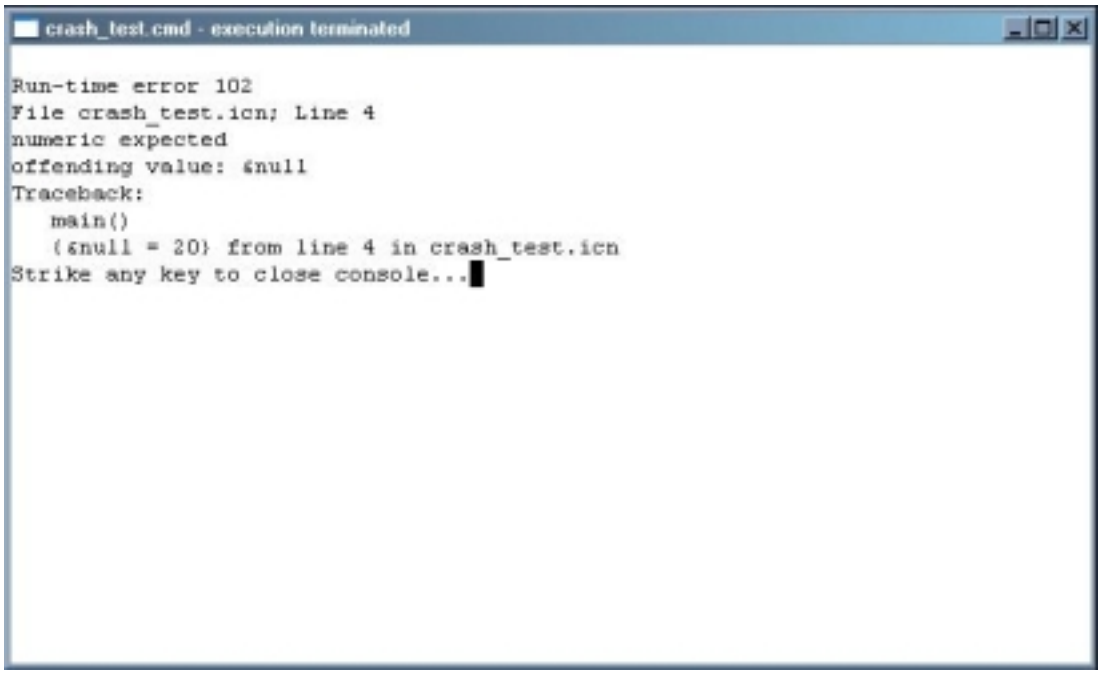


Figure (13) console error message on desktop PC

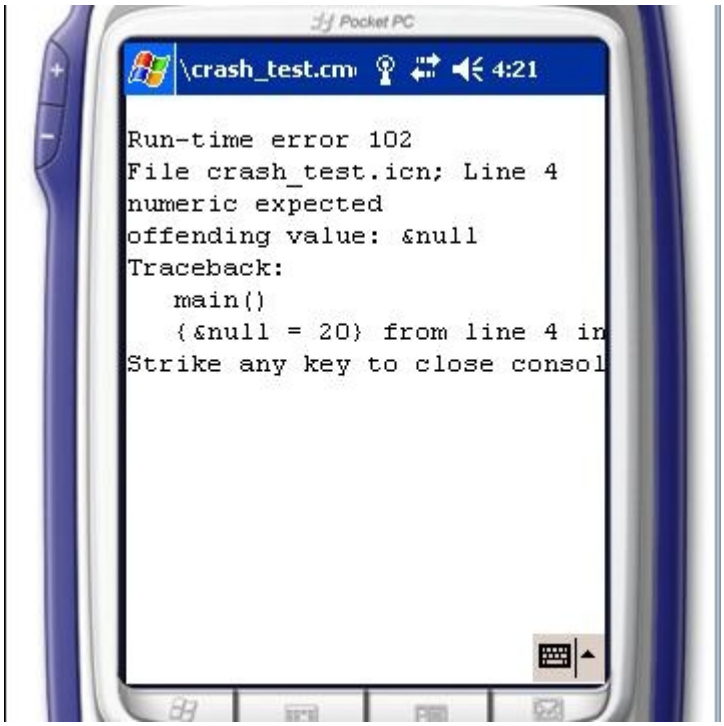


Figure (14) console error message on PocketPC Emulator

There are a number of ways of dealing with these drawbacks when running a program designed for display larger than what a PocketPC can provide. The first is to redesign the GUI or output for use on a PocketPC device. For GUI's, this option is probably the best in terms of the quality of the GUI, meaning the ability to quickly and effectively navigate it and accomplish what one wants to do. For console output, this could be as simple as placing new lines, but it could mean substantial rethink and redesign effort for a complex GUI. Another possible method of dealing with the small screen is to shrink all GUI components by a certain factor when compiling the program for a PocketPC. This could work out well for a medium sized GUI with a lot of empty space and a few controls.

A heavily populated graphical program, a good example of which is the gpxtest program, this strategy may not work so well.

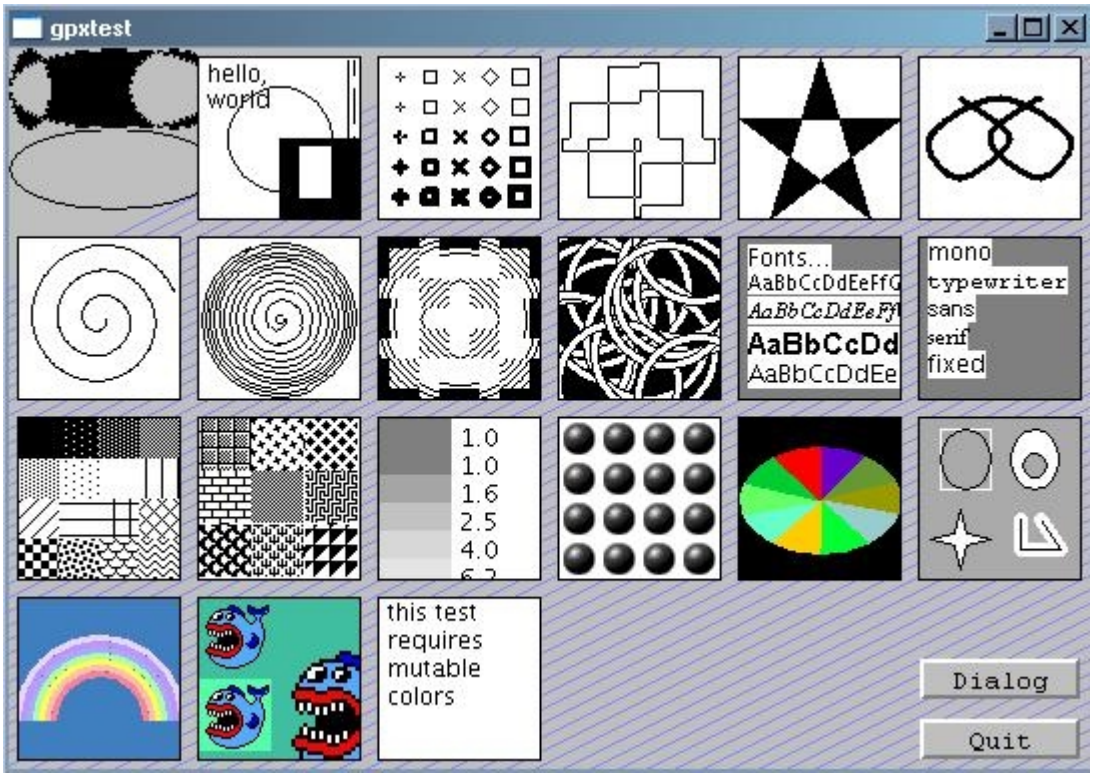


Figure (15) Unicon gpxtest resized to 240x320 (minus Windows command bars), preserving aspect ratio, giving a size of 1.3"x2.13"

While there is nothing truly wrong with the program in this case, it is very hard to read and many close together text fields, buttons, or other input mechanisms could be difficult to manipulate accurately.

A third method would be to add "background" scrollbars to any window when compiling a program for WM2003. The GUI or console would be displayed full sized, larger than the display screen, with the added scrollbars allowing the user to access all of the data or interface. This would keep the GUI or console legible, but would add some undesirable complexity to its use.

An additional consideration is that of drop down menus. Using the scrollbar solution, very long menus may be difficult to use. If the GUI is shrunk, clicking the correct option may also prove difficult.

6.0 Conclusion

The project will most likely not work under versions of PocketPC before 2003 (WinCE .NET 4.20). The ChooseFont function and CHOOSEFONT structure were not available in versions of WinCE before 4.0, at least in the included headers and library files. Additional functionality may have been added since PPC2002 (WinCE 3.0) as well, but overlooked. Standard C library functionality has been added to PPC2003 from PPC2002, but more investigation is necessary to determine exactly what has been added.

The Windows Mobile 2003 API is a subset of the "larger" Windows 32 operating system APIs. Porting Unicon to WM2003 from Windows has been straightforward for the most part. Most graphical facilities and "normal" computation/file management of Unicon should be easily implemented, the biggest challenge may be confining ones self to the small screen on the device. The weakest area of the PocketPC may be in the database area, very little seems to be present or available. The multimedia portion of Unicon may have to under go a large rewrite in order to be used, as MCI is not present in WinCE. Perhaps the greatest setback, perhaps not the largest, but the problem that seems like it could be easily dealt with, is the large portion of the C standard library that is missing. Perhaps the Microsoft team believes (or hopes) that most programmers are already using Windows APIs in place of standard C.

6.1 Future Work, Status of Unicon Features

Some of Unicon features are currently not built by default, the status of these features is as described below.

Posix
As a whole, posix is disabled, however, stat, send, receive, fetch, rmdir, mkdir and many other functions are built into iconxce and are usable.

Messaging
Messaging is as an option is disabled, but most required functions have been implemented. However, they have not been tested.

Coexpr
Coexpr is disabled, this option requires ARM assembly code and the ARMv4 compiler found in EVC4 does not support the ‘inline’ keyword.

Odbc
Odbc is disabled, support under Windows CE is very weak. The ODBCException does not exist, and if Microsoft eventually implemented support for ODBC in the future, it is likely to be MFC based.

DBM

DBM is enabled, it has not been fully tested. There some are some limitations due to hardware. Maximum file size is 32 Mb (which is actually half the ROM on the test device). Also, on WM2003, there are no atomic file actions (slow DB access) and there is weak file locking, meaning only one process can safely use a DB at a time.

7.0 Acknowledgements

A large thanks to the Ruby and Gdbmce projects for allowing their standard C and GDBM works to be copied, edited and distributed. Thanks to Dr. Jeffery for the Unicon Windows 32 port. Thanks to Microsoft for allowing the free use of VC7 for students, and general free use of EVC4/WM2003 SDK.

8.0 References

[1] Alexander, B. (1991). Icon in your pocket. *The Icon Newsletter*. 35, 1-2.

[2] Bayer, F. (2004). LispMe Home Page. Available: http://www.lispme.de/index.html#lispme/index.html

[3] Boling, D. (2003). Programming Microsoft® Windows® CE .NET, Third Edition. Microsoft Press.

[4] Cody, M. (2003). Toucan 1.2 (a feathered friend for the Palm) Available: http://toucan.sourceforge.net/

[5] Collins, J. Gorlick, M. M. (2002). Python to Palm Pilot Port. Available: http://www.isr.uci.edu/projects/sensos/python/

[6] DejaVu Software. (1999). GDBMce 1.7.3ce GNU's Database Manager, ported to Windows CE. Available: http://www.dejavusoftware.com/gdbmce/index.html

[7] Dragon FORTH for PalmOS. (2005). Available: http://sourceforge.net/projects/dragonforth/

[8] Fitton, D. & Montrose, R. (2003). Java Support on Pocket PC. Available: http://www.comp.lancs.ac.uk/computing/users/fittond/ppcjava.html

[9] Griswold, M., Griswold, R., Walker, K. & Jeffery, C. (2004) The Implementation of Icon and Unicon. Available: http://unicon.sourceforge.net/book/ib.pdf

[10] Herrera, C. (2004). Comparing Windows Mobile 2003 Pocket PC Phone Edition and the

SmartPhone. Available: http://www.pocketpcfaq.com/faqs/ppcpe-sp.htm

[11] Jeffery, C. (2002) Version 10 of Unicon for Microsoft Windows. Available: http://unicon.org/utr/utr7.html

[12] Icon Programming Language. (2005) Available: http://www.cs.arizona.edu/icon/

[13] Jeffery, C & Martinez, N. (2003) The Implementation of Graphics in Unicon Version 11. Available: http://unicon.org/utr/utr5a.pdf

[14] Jeffery, C., Mohamed, S., Pereda, R. & Parlett, R. (2004). Programming with Unicon. Available: http://unicon.sourceforge.net/ubooks.html

[15] Keuchel, R. (2002). CELIB DLL. Available: http://www.rainer-keuchel.de/wince/celib.html

[16] Keuchel, R. Perl 5.6 for Windows CE (2003). Available: http://www.rainer-keuchel.de/wince/perlce.html

[17] Maeda, S, Thomas, D. & Hunt, A. (2002). The Ruby Language FAQ. Available: http://dev.rubycentral.com/faq/rubyfaq.html

[18] Matsumoto, Y. (2000). The Ruby Programming Language. Available: http://www.informit.com/articles/article.asp?p=18225&rl=1

[19] Microsoft Corporation. (2003). Comparison of Windows CE .NET 4.2, Pocket PC 2002, and Windows Mobile 2003 Software for Pocket PCs. Available: http://www.microsoft.com/downloads/details.aspx?FamilyID=111fe6d5-b0e1-4887-8070-be828e50faa9&displaylang=en

[20] Microsoft Corporation. (2003). Comparison of Windows CE .NET 4.2, Pocket PC 2002, and Windows Mobile 2003 Software for Pocket PCs Available: http://www.windowsfordevices.com/articles/AT5612706156.html

[21] Microsoft Corporation. (2003). eMbedded Visual C++ 4.0. Software product. Available: http://www.microsoft.com/downloads/details.aspx?familyid=1DACDB3D-50D1-41B2-A107-FA75AE960856&displaylang=en

[22] Microsoft Corporation. (2005). Microsoft C++ Language Reference for eMbedded Visual C++ Processor Specific Options (2005). Available: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vcce4/html/evgrfProcessorSpecificOptions.asp

[23] Microsoft Corporation. (2003). SDK for Windows Mobile 2003-based Pocket PCs. Software product. Available:

http://www.microsoft.com/downloads/details.aspx?familyid=9996b314-0364-4623-9ede-0b5fbb133652&displaylang=en

[24] PalmSource. (2005). Java on Palm OS. Available: http://www.palmos.com/dev/tech/java/index.html

[25] PalmSource. (2004). Memory, Databases, and Files. Available: http://www.palmos.com/dev/support/docs/protein_books/Memory_Databases_Files/MDF_TOC.html

[26] PalmSource. Palm OS® Garnet (5.4) (2005) Available: http://www.palmos.com/dev/tech/oses/garnet540.html

[26] PalmSource. Palm OS. (2005) Available: http://www.palmsource.com/palmos/

[27] PalmSource. (2004). Palm OS® Programmer's Companion, Text. Available: http://www.palmos.com/dev/support/docs/palmos/PalmOSCompanion/Text.html

[28] PalmSource (2004). Palm OS® Programmer's Companion, Memory. Available: http://www.palmos.com/dev/support/docs/palmos/PalmOSCompanion/Memory.html

[29] PalmSource. (2004). Programming Palm OS in a Nutshell. Available: http://www.palmos.com/dev/support/docs/palmos/PalmOSCompanion/Nutshell.html

[30] Palm-Tcl. (2002). Available: http://palm-tcl.sourceforge.net/

[31] Perl for WinCE devices. (2005). Available: http://perlce.sourceforge.net

[32] Pippy: Python for PalmOS (2003). Available: http://sourceforge.net/projects/pippy

[33] Rojas, P. (2004). Windows Mobile passes Palm in PDA sales. Available: http://www.engadget.com/entry/7391854652028497/

[34] Rollin, K. (2003). What are the Palm platform limits I'll likely encounter?. Available: http://flippinbits.com/twiki/bin/view/FAQ/PlatformLimits

[35] Ruby. Ruby Project, Port to Windows CE. (2004) Available: http://www.opensource.apple.com/darwinsource/WWDC2004/Ruby-14/

[36] Small Basic. (2005). Available: http://smallbasic.sourceforge.net/

[37] Sun Microsystems. (2005). Java 2 Platform, Micro Edition (J2ME) Available: http://java.sun.com/j2me/

[38] Sybase Software. ODBC Programming. (2000). Available:

http://manuals.sybase.com/onlinebooks/group-sas/awg0702e/dbpien7/@Generic__BookTextView/12501;pt=13894

[39] Tcl PocketPC (2005). Available: http://wiki.tcl.tk/2946

[40] Tcl Windows/CE (2005). Available: http://wiki.tcl.tk/8688

[41] Unicon and Icon Technical Reports. (2005). Available: http://unicon.org/reports.html

[42] Unicron: A Virtual Computer Science Community. (2005). Available: http://www.cs.nmsu.edu/~jeffery/vcsc/

[43] Windows Mobile surpasses Palm OS in 2003 European PDA sales. (2004). Available: http://www.windowsfordevices.com/news/NS2827737955.html

Appendix A: changes

Header Changes

sys.h

It was necessary to undefine the macros Operation and Precision to avoid conflicts before including stdlib.h. Once stdlib.h is imported successfully, they are redefined.

define.h

HostStr was changed to "WinCe", to match the project.
A new define, #define WINCE 1 was added, this is the define used heavily for conditional compilation by VS7 and EVC4.

config.h

ARM defines (#define ARM 0) in config.h for Unicon also causes conflicts in various PPC2003 library files. When compiling for the PocketPC 2003 emulator, defining the symbol ARM causes conflicts between conditionally defined code for the emulator (x86 processors) and devices (ARMV4 processors).
Trace is defined as "TRACE" in order to avoid conflicts as well.

Modified Files

rmswin.ri

The file rmswin.ri was heavily modified, but the idea and structure of what a "window" is has not changed [11][13]. Most changes were strictly emulation or port type changes, where a function or "style" was not present and had to be replaced. Some missing API functions or "styles" (things like PS_DASH, which dictate how a line looks) had behavior that could be emulated through the use of other API functions using special parameters or similar "styles". The number of changes was not unexpected, as this file implements much of the graphics found in the Unicon language. The changes fall into two categories, invisible and visible. Invisible changes are changes that should not be noticed when running a Unicon program. Visible changes are those which may produce differing output or differing functionality when compared to Windows 32.

Important Visible Changes:

MCI

As mentioned previously, MCI (media control interface) API's are not present in WinCE. All MCI usage was removed. This has the affect of disallowing MIDI play back through Unicon.

Line Types

Only two of Unicons supported six line types are present under WinCE. All but PS_DASH and PS_SOLID have been removed.

ChooseFont

The `nativefontdialog` function has been disable. It uses Windows API calls: ChooseFont, CHOOSEFONT, CF_SCREENFONTS. These are supposed to be present in commdlg.h, but are not actually present in any WM2003 API headers. There are references to the first two inside the compiled library file, commdlg.lib.

Mutable Colors

Mutable colors are not supported, `set_mutable`, `mutable_color` have been changed to only return failed.

`ResizePalette`

`ResizePalette` is a Windows 32 API function, not present in WM2003, the ability to resize the palette has been removed by returning Failed in the `alc_rgb`. This also affected mutable colors.

`IsZoomed`

`IsZoomed` calls were removed, as there is no `IsZoomed` in WM2003.

`Nativemenubar`

`Nativemenubar` has been removed. It may be possible to implement, but much of the WindowsCE API for it is wrapped up in MFC.

`Drawarcs`, `Fillarcs`

`Drawarcs` and `fillarcs` use Windows API functions calls `Pie` and `Arc`, not present in WM2003. Presently arcs can be drawn with `drawarcs,` but a suitable replacement for the `Pie` is still under work.

Process Creation

The concept of processes creating background processes is mostly ignored in WM2003. In the function `mswinsystem,` the ability to call from the command line using the & has been disabled for WinCE.

Other Visible Changes:

WS_OVERLAPPEDWINDOW is not supported. Some of the styles that make up WS_OVERLAPPEDWINDOW are present, and can be combined to create an approximation. The WM_GETMINIMAXINFO message is not present, and was removed. SYSTEM_FONT_FIXED has been changed to SYSTEM_FONT, FIXED it was mostly for backwards compatibility with earlier Windows OSes and missing.

Invisible Changes:

The `TextOut` function is not present in WM2003 and is replaced with `ExtTextOut`. This function allows the same output with a special argument set.
The FW_DEMIBOLD font option does not exist in WM2003, however its value (600) is the

same as the FW_SEMIBOLD option, so DEMIBOLD was replaced with SEMIBOLD. `CreateDIBitmap` is not available, a call to `CreateCompatibleBitmap,` followed by `StretchDIBits` can be called with arguments telling it not to stretch the DIBitmap, acts as a replacement.

Other files

Modifications to other files in the project fall into three categories:

Changes:

Something was different in WM2003, requiring a define or function to be altered to a certain extent. Changes for calls or returns using Unicode are not listed.

fsys.r:

Invisible - `#define pclose _pclose` and `#define popen _popen` removed, these functions are not correctly found in xrsys.c otherwise.

Visible - `Getenv` has been removed, there are no environment variables to be found, and it conflicts with another declaration under WM2003.

Visible - `WinAssociate` has been simplified because of a missing `FindExecutable` function.

imain.r:

Invisible - The type of stdin/stdout/stderr differs from that of Win32, `detectRedirection`'s calls to `fstat` have changed to reflect this.

Invisible - The seed directory of `_tempnam` has changed to \Temp.

Invisible - The declaration of `WinMain` `LPSTR` (ASCII) was changed to a LPWSTR (Unicode).

Invisible - `GetCommandLineW` (Unicode) used in place of `GetCommandLine` (ASCII). `GetCommandLine` should be usable, but does not seem to correctly return the Unicode string from the command line.

init.r:

Invisible - `setbuf` does not exist, setvbuf is used in its place with _IONBUF parameter.

Invisible - The variable "this" is disliked by the compiler, it is a C++ keyword, but the EVC4 "C" compiler recognizes it as a keyword. This has been changed to this1.

rsys.r

Invisible - #define pclose _pclose removed, for the same reason as fys.r.

Invisible - NTGCC ifdef, which hides libc replacements, has been expanded to #if NTGCC || WINCE for up missing functions (popen/pclose).

Visible - `popen` has been simplified, since there is no concurrent run. The parent process waits for the child process to complete its work before moving on.

The testing phase has used `popen("\iconx \path_to_icodefile.cmd > icodefilename.txt", "p")` extensively.

common/redirerr.c

Visible - Due to the way stdout/stderr are declared, the default `setbuf` calls and `stderr->file = stdout->file` has been replaced with `dup2(fileno(stdout),fileno(stderr)).`

lib\wince\io_wce.c

Invisible - The functions read, write, open, and rename were defined by EVC4 as having a _ on the name (_read). This was causing problems when EVC4 was compiling and linking.

lib\wince\direct.c/.h

Invisible - A change similar to io_wce.c was done, `_chdir` became `chdir`.

Additions:

The Ruby standard C replacement files provided a good starting point, but they were missing function calls and definitions needed in Unicon. New files were added. These files contain code pulled from the WM2003 SDK MFC and ATL example implementations of functions missing from the standard Windows Mobile 2003 API.
Perhaps Microsoft did not wish to duplicate calls in the standard API and MFC/ATL in order to save space, or purposely left certain things as MFC/ATL only in order to encourage the use of those libraries. In any case, the example source code given in the MFC and ATL directories for WM2003 show how to implement many "missing" calls.

lib\wince\wcealt.h/.c

Invisible - These two files were added to lib\wince and contain replacements for `CreateBitmapIndirect`, `SetMenu` and `FrameRect`.

lib\wince\wcetime.c

Invisible - This file was added to lib\wince and contains a replacement `GetMessageTime`.

lib\wince\stdlib.c

Visible - This called a `rb_w32_getenv` function when doing a `getenv`. The `rb_w32_getenv` function was moved to stdlib.c, but it is not an adequate, it will have to be replaced for full environment variable functionality.

lib\wince\stdio.c

Invisible - Replacements for `tempnam` and `tmpfile` functions were added to this file. These two functions have declarations in lib\wince.h.

lib\wince\io_wce.c

Invisible - `Remove` was implemented as a call to `DeleteFile`.

lib\wince\process.h/process_wce.c

Invisible - #define for _P_NOWAITO added.
Visible - `spawnvp` implemented as a call to `CreateProcess`, however it can not handle NOWAIT cases.

Removals:

Conflicts with between WinCE API functions and replacements in the standard C replacements no longer needed.

lib\wince\wince.c/.h

Invisible - `DeleteFile/DeleteFileA` and `CreateProcess/CreateProcessA` functions have been removed, they have apparently been added to the API since PocketPC2002.

Header Files II

Once these steps were completed, it became necessary to modify some headers further.

rmacros.h

BUFSIZ and PATH_MAX defined as MAX_PATH. The PATH_MAX is the maximum length of the path, and BUFSIZ is the max size for many char arrays used as "user" buffers. BUFSIZ is normally found in stdlib.h, but it is not present in WinCE.

proto.h

Include for dos.h removed, there is no dos.h in WM2003.

rproto.h

The definition of `coswitch` was changed to `int coswitch(word *old, word *new1, int first);, from … new …` , the EVC4 compiler saw new as a reserved word.

rstructs.h

"this" in b_proc_list was changed to this1.

rt.h

Includes commdlg.h and wince.h/wcealt.h. Commdlg.h is supposed bring in `ChooseColor, ChooseFont` and `GetOpenFileName`.

grttin.h

Modified due to the changes in the configuration, addition of standard C library replacements and new symbols used in WM2003. Additions include, TCHAR, macro for Unicode character under WinCE, PROCESS_INFORMATION, cWnd, CHOOSEFONTW, STARTUPINFO, SECURITY_ATTRIBUTES, HRGN and HGDIOBJ.

Appendix B

(k/k* means some sort of output text file was created, 0 means no output file, ? means the program may have locked the emulator up or did lock the emulator up. Ok/fine means results passed diff. Hand means the test completed successfully when run individually, dat means the test had its input file read in from a file opened in the program, rather than as a redirection argument)

"augment", k* - fine
"btrees", k* - ok, hand, dat file
"cfuncs", 0 -.u file?
"checkc", k* - writes tab instead of \t. more digits written for pi, etc. &output written file(&output)
"checkfpc", k* - 2e+13 -> 2.000000e+13.
22222222222222222222222222222 written
2.222222e+028
--36. ^ 9 ----> -1.01559e+14
--36. ^ -9 ----> -9.84640e-15
+-36. ^ 9 ----> -.101559e+15
+-36. ^ -9 ----> -9.84640e-015
all 2.222222e+xx written as e+0xx
"checkfpx", k* - same as above
"checkx", k* - same as checkc
"ck", k* - binary files differ?
"coexpr", k* - binary files differ? – coexpr not implemented
"collate", k* - \t printed as tab
"concord", k* - ok, hand, dat file
"diffwrds", k* - ok, hand, dat file
"endetab", k* - only testing entab/detab(s) appears
"env", k* - works, but no environment variable support
"errors", k* - errors stop at line 128, reports to end

when not redirected. Max redirect size? (9.17k)
"evalx", k* - ok, hand
"fncs", k* - results stop after (6 0's) 000000abcdef...
ok when run alone
"fncs1", k* - open(gc1.icn) seems to fail
line 19 fncs1.icn error
"gc1", k* - only "filling..." ok when run alone
"gc2", k* - tests stop after <a>::=1|2|3
"gener", k* - only first test, no [ok]
"helloc", k* - no std files, output seems ok
"hellox", k* - no std files, output seems ok
"ilib", k* - crash, seems to die when looking for /tmp
"kross", k* - ok, hand, dat file
"large", k* - dies before: large.icn : 28 | foo(1024)
"meander", k* - ok, hand, dat file
"mem01c", k* - prob? 1\n2\n3\n4\n5\n6\n
"mem01x", k* - prob? 1\n2\n3\n
"mem02", ?
"mffsol", k* - ok, hand, dat file
"mindfa", k* - prob - waiting for input?
"numeric", k* - ok
"others", k* - one is only output
"over", ?
"pdco",0
"prefix", ?
"prepro", k* - ok
"proto", 0
"recent", k* -probs reading file? dies after ffffff9b
"recogn", k* - ok, hand, dat file
"roman", 0 - open window
"scan", k* - ok
"sieve", k* - ok
"string", k* - ok
"struct", k* - &output is file(&output)
"tracer", 0 - probs
"transmit", ?
"var", k* - ok
"wordcnt" k* - ok, hand, dat file

Appendix C

Provided by Ruby
(Worked, or only required a small amount of fixing up)
*getcwd

chdir
_rmdir
_mkdir
ftime
_stat
fstat
utime
GetMessageTime
wce_FILETIME2int64
wce_int642FILETIME
wce_getFILETIMEFromYear
wce_getYdayFromSYSTEMTIME
wce_tm2SYSTEMTIME
wce_SYSTEMTIME2tm
wce_FILETIME2time_t
wce_time_t2FILETIME
time
*localtime
mktime
*gmtime
ctime
*asctime
tzset
*_fullpath
mblen
*bsearch
*freopen
*fdopen
signal
_getpid
rename
_unlink
open
close
read
remove
_lseek
_findnext
_findclose
_isatty
GetModuleFileNameA
GetProcAddressA
GetCommandLineA
wce_SetCommandLine
wce_FreeCommandLine

GetFileAttributesA
SetFileAttributesA
MoveFileA
MoveFileEx
GetVersionExA
WaitForMultipleObjectsEx
CreateProcessA
CreateEventA
FormatMessageA
FindFirstFileA
FindNextFileA
CreateFileA
CharNextA
CharPrevA
GetLogicalDrives
GetUserName
LoadLibraryA
LoadLibraryExA
*wce_fopen
wce_SetCurrentDir
*wce_replaceRelativeDir

Found elsewhere, but did not initially work
(substantial amount of work required to fix up)

CreateBrushIndirect
FrameRect
wce_CreateFont
wce_GetScrollRange
wce_GetScrollPos
wce_IsIconic
wce_CreateBitmapIndirect
GetArcPoints
ReleaseArcPoints
Pie_wce

Found elsewhere

(Worked from the start)

setbuf
_findfirst
write

Fully implemented for the project

*tmpfile
*_tempnam
_spawnvp

Not implemented
(function present, but result is failure return value)

wce_SetMenu
clock
*getenv
raise
abort
_cwait
_spawnle
_execl
execv
_spawnvpe
_chsize
_umask
_chmod
dup
_pipe
_access
_open_osfhandle
_get_osfhandle
GetProcessTimes
GetEnvironmentVariable
SetEnvironmentVariable
GetEnvironmentStrings
FreeEnvironmentStrings
GenerateConsoleCtrlEvent
LockFile
LockFileEx
UnlockFile
UnlockFileEx
CreatePipe
GetStdHandle
SetStdHandle
ReadDataPending