A Survey of Selected Aspects of Selected Mobile Platforms

IMEDIA/03/00

Arve Larsen

Oslo
June 2000

Tittel/Title: A Survey of Selected Aspects of Selected Mobile Platforms	**Dato**/Date: June
	År/Year: 2000
	Notat nr: Note no: IMEDIA/03/00

Forfatter/Author:
Arve Larsen

Sammendrag/Abstract:

This report covers selected aspects of selected mobile platforms. The selected platforms cover all the major consumer PDA offerings, as well as the most important platforms for building mobile embedded systems.

The first part of this report presents the *Selected Aspects* described later, followed by a brief description of our use of *Mobile Platforms*. This is followed by a description of *Common Aspects* of the mobile platforms described herein. The main part of the report covers the different *Operating Systems* used in our selected mobile platforms. A brief *Summary* completes the report.

Emneord/Keywords:	mobile platforms, EPOC, Windows CE, Palm OS
Tilgjengelighet/Availability:	Open
Prosjektnr./Project no.:	619015
Satsningsfelt/Research field:	Mobile devices and platforms
Antall sider/No. of pages:	24

Norsk Regnesentral / Norwegian Computing Center
Gaustadalléen 23, Postboks 114 Blindern, 0314 Oslo, Norway
Telefon 22 85 25 00, telefax 22 69 76 60

A Survey of Selected Aspects of Selected Mobile Platforms

Contents

The material in this document is partly based on a report written for Birdstep Technology ASA.

1 Selected Aspects

The purpose of this report is to select and describe aspects of mobile platforms that are important for planning and developing software components, libraries or similar for use on multiple platforms. On this basis we have selected modularisation, state management, memory architecture, memory management, persistent storage, multitasking features, remote access and synchronisation, development environment and language support, as well as supported CPUs.

By modularisation we mean aspects like dynamic link libraries, application architecture, bundled functionality etc. This is especially important when designing code or functionality to be shared by different processes. By state management we mean the operating systems handling of power-on and power-off. This needs to be considered when handling persistent data. Memory architecture describes how memory is allocated to processes, addressing schemes, partitions, memory constraints etc. With typical mobile devices having very limited memory this must be considered in many kinds of applications. Memory management describes how applications should allocate and handle memory. This is essential for creating robust applications. The different platforms have very different strategies for enforcing or supporting good memory management practices. Persistent storage describes the different abstractions provided by the operating system for handling persistent data such as files, databases etc. This is coupled to the memory architecture as well as the platforms support for removable storage such as compact flash cards. The platforms multitasking features, such as threads, locking primitives etc, is especially important when creating complex applications where more than one application or thread may need to access shared data or other resources. Many mobile platforms are used in connection with other, perhaps desktop based, systems. The different operating systems provide different features for remote access and synchronisation. These features may be valuable for backup purposes, integration with office and legacy systems etc. When developing applications, both the available development environments and supported languages should be considered. For low-level programming the target CPU should also be considered.

2 Mobile Platforms

This report considers a mobile platform to be a complete device, with its operating system, input/output devices, communications, extensions etc. When considering mobile platforms, it useful to divide them into groups characterised either by physical design or operating systems.

	Handheld	**Palmsize**	**Auto**	**Other**
CE	X	X	X	x[1]
EPOC	X	(x)[2]		

[1] Microsoft seems to be seeking a wide acceptance for its Windows CE operating system as a platform for both PDAs and embedded systems, targeting a wide range of generic devices as well as specialised devices such as car stereos and game consoles.

Palm		X		
VxWorks				X[3]

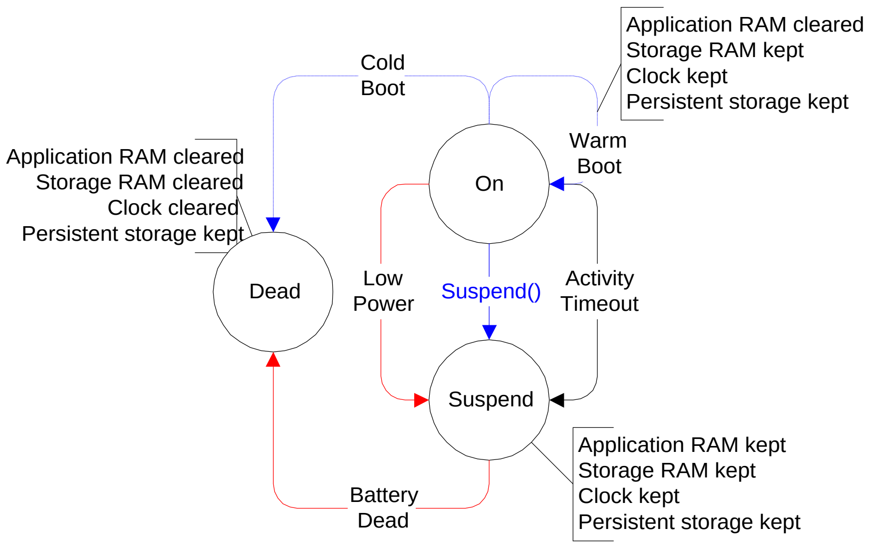
A third dimension can be described on the basis of communication support. For instance “Communicator” is often used to denote a combination handheld PDA and mobile phone. Similarly “Smartphone” is often taken to mean combination of a palmsize PDA and a mobile phone.

3 Common Aspects

This section outlines common aspects of the different platforms.

3.1 State Management

Most of the platforms share more or less the same state model. In the On state the device is perceived as being on, with applications running etc. The On State itself may have several different sub-states, allowing for efficient power management. In the Suspend State the device is perceived as being off, with no applications running. However, all internal memory (application and storage RAM) as well as the system clock is kept alive. In addition, some processes and hardware components are active, allowing the system to wake up in response to specific events (connections, alarms etc.).



Please observe that the Suspend State requires battery power. Whenever the battery fails, the device reverts to the Dead State and all RAM content is lost.

Platform specific details such as state transition signalling etc. are described separately.

[2] Several EPOC based SmartPhones have been previewed and even announced, but so far none have made their way into the public market.
[3] VxWorks target is mainly embedded systems. It is included here because some highly specific mobile applications use and require the kind of environment provided by this OS.

3.2 Storage Solutions

All of the platforms have both volatile and non-volatile memory built in.

3.2.1 Volatile Memory

The main operating memories of these platforms is based on standard volatile RAM, allowing fast access, and direct addressing on a word (or byte) level. This memory is used to store user data as well as user installed applications. Following the state model, this memory is perceived as persistent because of the Suspend State. Whenever the device reverts to the Dead State, both applications and data in this storage is lost.

3.2.2 Non-volatile Memory

All of the devices have basic applications loaded in ROM. In addition most support persistent storage on flash memory or similar types of storage.

3.2.2.1 ROM

The main content in ROM is the Operating System itself. In addition, core applications like calendar, address book etc. are often stored in ROM. The platforms themselves may allow for custom applications to be loaded into ROM, but usually the device manufacturers want to control such applications themselves.

3.2.2.2 Flash

Flash memory is solid state persistent memory. The most common varieties, PCMCIA and CompactFlash (CF), both provide a device interface based on a standard disk interface (ATA). Each operating system provides a file-system on to of the device interface. To ease interoperability, most platforms use a FAT-based file-system. When considering the use of flash storage for application purposes, the file system employed should be taken into account. Some simple file systems like FAT16 are not very efficient for larger volumes (such as 128MB or more)[4].

Linear flash provides a linear address space instead of the ATA interface. This enables linear flash to be used as direct RAM, but most implementations provide a block device abstraction instead (similar to the ATA interface of CF and PCMCIA cards).

From an application perspective, most flash memory perform and behave like disks. This means that access is based on whole sectors, and that performance depends upon the performance of the card. In addition, solid state flash memory requires that each sector must be erased before new data can be written.

In small mobile devices, CF cards are most common. The standard defines two different physical sizes, CF I (thinnest) and CF II (a bit thicker). The amount of available storage increases steadily. At present, 192 MB CF I cards are available as well as 488 MB CF II cards.

[4] For 128 MB drives, FAT16 uses allocation units of 4 KB. Similarly 8 KB is used for 256 MB drives, 16 KB for 512 MB drives and so on. This is especially important when dealing with many small files or detailed (for instance record based) access to larger files.

The original CF standardisation assumed solid state storage. Now, several manufacturers also deliver miniature harddrives on CF cards. These are usually CF II cards with current maximum capacity of 340 MB and 1 GB announced for later this year.

At present both solid state and disc based cards have physical limitations. Solid state flash memory has an upper limit on how many times one physical sector may be written. Most guarantee a minimum of 300 000 writes per physical sector, but also state that 1 000 000 writes could reasonably be expected. In addition, many cards employ a dynamic mapping from logical to physical sectors, limiting the actual writes to a specific physical sector. When updating a logical sector this means that the data is read from one physical sector, updated in a sector buffer and written to another physical sector.

The IBM MicroDrive have another limiting factor. Each card is specified to handle 300 000 load/unload cycles, i.e. how many times the drive head is moved off the physical discs in order to prevent damage when the card shuts down.

The minimum (and most typical) sector size of most block devices like CF cards are 512 bytes/sector, with 1024, 2048 and 4096 bytes/sector being not to uncommon. In extreme cases 64kbytes/sector may be used.

3.2.2.2.1 Typical CF performance figures

IBM (MicroDrive)	Sector Size: 512bytes Load/unload cycles: 300 000 (head park when powering off)
Sandisk (Solid state)	Sleep to write 2,5 msec max Sleep to read 2,0 msec max Reset to ready 50 msec typical, 400 msec max Data Rate to/from Flash 4,0MB/s burst Data Rate to/from host 6,0MB/s burst
Lexar (Solid state)	Up to 10MB/sec burst Sleep to read/write 25 msec Sustained write min 600KB/s (4X), min 1,2MB/s (8X) 10 million images guaranteed
Kingston (Solid state)	300 000 cycles pr. logical sector

4 Operating Systems

This chapter describes the different operating systems used by the selected platforms. Each OS is given a brief introduction before the selected aspects are covered.

4.1 Windows CE

Microsoft’s Windows CE is a member of Microsoft’s WIN32 family of operating systems (together with Windows NT and Win95/98). All operating systems in this family provide the same WIN32 API. The Windows CE architecture allows the OS itself to be adapted to create different platforms. Specific platforms may support other Microsoft specific technologies, such as COM, ATL and MFC. The core OS itself supports multiple processes with multiple threads.

Microsoft has not been kind to the public when naming their various Windows CE offerings. Firstly, they deliver the generic operating system through a platform builder pack. The basic operating system have been through several versions (at least 1.0, 2.0, 2.1, 2.11, 2.12, 3.0). The general trend is a move towards richer functionality, as well as optimising core components such as the OS kernel. Microsoft has presented Windows CE as their embedded operating system for small footprint systems (thereby distinguishing it from Embedded Windows NT). In version 3.0, they have strengthened the real-time capabilities, as well as including the word ‘embedded’ in several of the related products such as the corresponding visual tools.

In addition to the OS itself, Microsoft markets several generic platforms that in principle offers exactly the same environment. These are used by most (if not all) Windows CE consumer devices. The generic platforms are summarised in the table below.

Abbrv.	**Name**	**OS**	**Comment**
HPC	Handheld PC	(1.0) 2.0	Later versions (2.11) sold as “Windows CE, Handheld PC Edition, version 3.0”
HPC Pro	Handheld PC Pro	2.11	Sold as “Windows CE, Handheld PC Professional Edition, version 3.0”
PPC	Palmsize PC	2.1	
PPC 1.2	Palmsize PC	2.11	Initially marketed as “Colour PPC”
Pocket PC	Pocket PC	3.0	Successor to PPC, marketed as “Powered by Microsoft Windows for Pocket PC”
Auto PC	Auto PC		Later versions marketed as “Microsoft Windows CE for Automotive 2.0”

It is important to realise that device manufacturers are free to provide their own adaptations of the core OS. This means that even though several manufacturers provide the same generic platform, the actual implementations may be different. A consequence of this is that if an application is to be certified, it must be tested against specific devices, not only generic platforms (although most applications will work on most devices built from the same generic platform).

4.1.1 Modularisation

Windows CE uses the common application/DLL architecture of the WIN32 family. A running application constitutes a process with at least one thread. In addition most Windows CE devices also support COM-objects, enabling different applications of utilising the same components. This is hardly surprising since one goal of Windows CE is to provide a programming environment similar to other WIN32 platforms.

4.1.2 State Management

Windows CE follows the state model described in common aspects quite closely. In particular, when entering the Suspend State, no signals are given. To the application programmer, this state change could be considered as a context switch. When the system reverts to the On State and the application becomes active, no changes should be apparent.

One consideration is that when the system enters the suspend state, the user perceives the device as being off. This means that removal and insertion of CF cards etc. are likely to be done in the Suspend State. When the device is reactivated, the system sends a WM_DEVICECHANGE message and a NOTIFICATION_EVENT_DEVICE_CHANGE as if the card was removed while the device was on. Since the same signals are sent if a card is removed or inserted while in the On State, applications using such cards should process these signals or actively check the presence of a card before accessing it.

4.1.3 Memory Architecture

Windows CE provides a 1 GB virtual address space for applications. This is divided into 32MB slots, reserving one slot per process. This gives a maximum of 32 processes. (There is no similar limit on the number of threads.) In addition to this Windows CE provides 1 GB of virtual memory for memory mapping between processes. Physical memory is allocated to processes one page at a time. Typical page sizes are 1KB and 4KB.

Uncompressed ROM programs (including DLLs) can run in-place. Compressed ROM programs will be decompressed and executed in RAM. Most CE devices have volatile RAM. Some may have non-volatile RAM (usually linear flash[5], none discovered so far). Se State Management for providing pseudo non-volatile RAM through the suspend state.

Two types of memory are available to application programmers. Part of available memory is set aside as RAM for use by applications for storing volatile data. The rest of available memory is called the object store and are used to store persistent data.

In version 3.0 the maximum size of the object store have been raised from 16 MB to 256 MB. The maximum size of a single file has been raised from 4 to 32 MB. The maximum size of a database volume has been raised from 32 to 256 MB.

4.1.4 Memory Management

Windows CE relies on standard C/C++ memory allocation, types etc., with some Microsoft specific solutions. One important aspect of Windows CE programming is that standard C++ exception handling is not supported. Instead Microsoft provide both WIN32 and MFC exception handling. These provide more or less the same functionality, but are realised in a completely different way.

There are no specific requirements for memory management, but when available memory falls below the hibernation threshold (typically 128 KB) the system will send a WM_HIBERNATE message to each visible application (i.e., with a button on the taskbar). Upon receiving the WM_HIBERNATE message the application should free as much memory and resources as possible.

---

[5] See http://www.microsoft.com/windows/embedded/ce/previous/developer/hardware/architecture/ftl.asp

Windows CE specifies two further thresholds, the low-memory threshold and the critical-memory threshold. These effectively limit the amount of memory available through the VirtualAlloc function. Any request for more memory through VirtualAlloc that breaks one of these thresholds results in a “System Out of Memory” message to the user who are prompted to close an application. If available memory is about to fall below the low-memory threshold, WM_CLOSE is used to close applications. If the application does not close within 5 seconds, the user is prompted with an “End Task or Wait” dialogue. End Task results in the application being terminated by TerminateProcess (with no grace whatsoever). If available memory is about to fall below the critical-memory threshold, TerminateProcess is used instead of WM_CLOSE.

To reduce the risk of a single memory allocation through VirtualAlloc failing, Microsoft recommends using small memory allocations, typically 4kb or less, i.e. not allocation more memory than you know you are going to need in the near future. This scheme has no effect whatsoever if the running applications actually require more memory than is available.

4.1.5 Persistent Storage

In Windows CE the term persistent storage applies both to RAM kept alive in the Suspend mode (but lost in Dead mode), and to data stored on persistent devices like CF-cards.

Windows CE has three different types of interfaces for accessing both types of persistent storage. The CE file system is used to store data files and applications. It is transactioned, FAT based, and each device can handle up to nine volumes (storage cards or partitions on a storage card). The CE database provides a very simple but limited way of storing tabular or record based data. The CE registry allows storing of common application information in a central registry[6].

Although all Windows CE platforms share the same basic abstractions and interfaces, some present different logical file systems to the user. For the Windows CE Palmsize platform only specific parts of the total file system is presented to the user through standard components for file selection and retrieval. Specifically, only directories starting with My Documents (but all of them regardless of which storage card they are on) are shown.

4.1.6 Multitasking Features

Windows CE provide more or less the same multitasking features as the other WIN32 platforms. This includes multiple processes, multiple threads pr. process and preemptive multitasking. An important distinction is that when the main thread of a process terminates, the whole process terminates.

Threads are scheduled according to priority. High priority threads are handled first. Scheduling between threads with the same priority is done in a round-robin fashion.

[6] Although registry information may be stored on a persistent device, it must be copied to the RAM-based registry before it can be used.

Lower priority threads only runs after higher priority threads have finished, e.g. are idle or waiting. Events, mutexes and critical sections are used for thread synchronisation. Events and mutexes may also be used for inter-process synchronisation. A thread may lose control at any time, except when within a critical section.

4.1.7 Remote Access and Synchronisation

RAPI allows for remote access to a device from a host. This includes viewing file systems, heaps, processes and events.

ActiveSync provides a framework for synchronisation between a device and a host. Synchronisation may be automatic on connect, automatic continuous or manual. ActiveSync is built around a Service Manager and several Service Providers. The Service Manager provides basic synchronisation, including connectivity[7], change detection, conflict resolution as well as mapping and transferring data objects. It is implemented as a part of the Windows CE Services both on the client and the host side.

The service provider is implemented two modules, the desktop provider module and the device provider module, both DLLs. A service provider implements synchronisation tasks specific to an application.

“The service provider interfaces with both the service manager and the application, thereby exposing the application and the application’s data to the service manager. In turn, this interaction enables the service manager to synchronise different types of data from different types of applications. The service provider also facilitates all requests made by the service manager, such as displaying a user interface (UI) or reporting status.” (Taken from the Windows CE Toolkit for Visual C++ documentation).

The basic ActiveSync infrastructure includes providers for built-in databases, including synchronising files and databases. In addition ActiveSync offers backup and restore functionality for connected devices. By implementing and registering new providers application developers can provide application specific transformations, synchronisation rules etc.

4.1.8 Development Environment and Language Support

Microsoft offers two levels of Windows CE development tools. The Windows CE Platform Builder (currently version 3.0) is aimed at device producers, enabling tailoring the OS to a specific platform (processor, peripherals, OS functionality etc). For application development newer versions of the Platform Builder (at least 3.0) include Microsoft’s eMbedded Visual Tools.

The Microsoft eMbedded Visual Tools (currently version 3.0, previously called Windows CE Toolkit) contain all that is necessary to develop CE programs in C++

[7] Based on Windows CE communication services, meaning that most types of communication (LAN, serial, infrared) can be used.

and Visual Basic. It is structured into a generic base, consisting of the compiler and necessary support infrastructure. In addition one or more platform specific SDKs must be installed. The product CD includes SDKs for the most relevant generic platforms, including Pocket PC. SDKs for all the generic platforms are available from Microsoft. Others can be produced using the Platform Builder. Although most of the code can be shared across different platforms, applications must be compiled for a specific platform and a specific processor.

In addition to Microsoft, several other companies build development environments. Sun has produced a beta version of Personal Java for Windows CE. WABA is another Java offering, building a very lightweight core (significantly smaller and less powerful than Personal Java) for building Java programs. Pocket C is and environment for developing C applications. Pocket C applications are not compiled to native code, but to a byte-code format run by the Pocket C interpreter.

Windows CE itself does not provide any scripting environment (other than the Visual Basic environment). Some are available from different sources, see for example http://www.geocities.com/ResearchTriangle/Lab/3533/palm_sw2.html#prog.

4.1.9 Supported CPUs

For application programmers, Windows CE hides all processor details. Like other Windows platform it is also assumed that all numbers are stored in little-endian format. For highly specific low-level programming, for instance special device drivers, processor details may be of importance. Even though the OS itself is largely processor independent, specific applications must be compiled for specific platforms, including specific processor architectures.

From the Toolkit for Visual C++ documentation[8]:
AMD X5, ARM 720T, ARM SA-1100, Hitachi SH4 (16 bit support), IBM PPC 403GC, MIPS 4102, MIPS 4111, MIPS R3910, MIPS R3912, MIPS R4101, Motorola MPC823, NEC VR4111 (16 bit support), NEC VR4300, PPC 821, QED5230, SH3, x86

4.2 EPOC

EPOC is an operating system specifically design for mobile devices, now owned and developed by Symbian. The OS itself is licensed to manufacturers such as Psion, Nokia and Ericsson who produce consumer devices. The current crop of EPOC licensees indicates that EPOC's future will be strongly influenced by the development of mobile phones, smartphones, communicators and other communication centric mobile devices.

EPOC is lightweight general OS optimised for small devices, not favouring specific types of applications. The OS itself is object-oriented, with special constructs for handling multithreading and exception handling.

[8] See also http://www.microsoft.com/windows/embedded/ce/guide/processors/default.asp.

There is currently only one version of this OS available in consumer devices[9]. The Psion 5 and similar devices all use EPOC release 5 (ER5). Several new devices are under development using new or specialised versions.

Ericsson's R380 SmartPhone is based on ER5 with significant changes. Important examples are a proprietary GUI and UNICODE support.

Expected changes in a new version (ER6) include stronger media support as well as general UNICODE support. Other possible extensions will probably include adaptations to voice- and communications centric devices such as Ericsson's R380 and Nokia's Communicator.

4.2.1 Modularisation

EPOC provides a standard application/DLL architecture. Executing applications constitute a process that may utilise dynamic link libraries as needed. In addition EPOC favours a client/server paradigm for creating applications. Examples of bundled servers are the file server, the serial communications server (including telephony), the window server and the font and bitmap server. Applications providing generic functionality to other applications should be implemented as servers. Standard thread-functionality is available, but EPOC provides Active Objects as a more resource friendly and object-oriented abstraction (see *Multitasking Features* for more information).

In addition to the application architecture above, EPOC provides several general-purpose engines. Most engines are libraries providing extended functionality on top of servers. For persistent storage EPOC provide the STORE and DBMS engines for accessing streams and databases respectively (see *Persistent Storage*).

The APPARC engine (application architecture) provides the basic infrastructure for launching applications etc. An important consequence of this architecture is that every application and native file type requires a unique id (UID). UIDs are also used for embedding file types for one application inside another application.

The BAFL engine (basic application framework library) provide, among other generic functions, the necessary functionality for utilising resources such as language dependent texts, UI-components etc.

Three different text-processing engines are available, CHARCONV, LEXICON and ETEXT. CHARCONV provides conversions between various character sets and is integral in EPOC's UNICODE support. LEXICON provides spell-checking functionality for alphabetic languages, with English implemented in the standard server. ETEXT is a general-purpose text component used by databases, word processors etc.

[9] ER3 is available on the Psion 3 family of devices, but Psion no longer supports them.

In addition to these general-purpose engines EPOC provide a comprehensive GUI supporting windows, user input, printing, fonts etc. CONE is a basic GUI-environment built on top of the window server. EIKON is a more comprehensive GUI-environment built on top of CONE.

4.2.2 Memory Architecture

EPOC supports an abstract memory model based on a conventional two-level memory management unit (MMU) with a common page directory for all processes. Memory is allocated in chunks, typically with one private chunk per thread, although chunks may be shared. Each chunk occupies one page directory entry (PDE) and typically has a stack at the bottom and a heap at the top. Chunks may grow upwards (possible requiring more page directory entries), but not downwards. For Java support, EPOC also support chunks that grows downwards.

Context switching on the current EPOC platforms involves moving all the process' PDEs from a user area to an area accessible by the kernel. ER5 supports four fixed process slots (for processes that live in the kernel accessible area) for high-speed context switching. The file server, the serial communications server, the window server and the font and bitmap server occupy these process slots.

4.2.3 Memory Management

A special feature of EPOC is that every thread in principle has its own heap. This means that even if two threads belong to the same process, they have different heaps. This is especially important when code running in one thread tries to access data created in a different thread and thus on different heap. A specific thread may share another's heap either by adopting the other thread's heap as its own or by switching heaps during execution. See *Multitasking Features* for more on EPOC's unique multitasking environment.

To preserve a focus on and support for efficient applications in a restricted environment, EPOC provides several specific types. EPOC strongly suggests (and in most cases require) the use of special EPOC data types instead of native C/C++ data types. Many EPOC types, such as TInt, correspond directly with native C types, such as int.

Most EPOC types follow a specific naming convention. Application developers are encouraged to use the same convention. EPOC provide three different groups of data types, called C, T and R-types.

C-types are classes derived from CBase (as all classes should be for cleanup purposes, see below) and allocated on the heap. They should not be allocated on the stack or as member variables. C-types must be cleaned up when they are no longer needed (CBase provides a virtual destructor, but many classes will need to provide their own). C-types are referenced by a pointer. See cleanup below for more on ownership of C-types. C-types are passed by reference.

T-types do not have any ownership of other data. Examples are simple types and enumerators like TInt and TAmPm, objects like TBuf<40> that do not contain any references to other objects (i.e. needs no destructor), and objects like TPtrC whose external references does not indicate ownership (i.e. the primary pointer to the other object exists elsewhere). T-types can be passed by value or by reference. Reference is preferred for T-types of more than two machine-words total length. It is till possible (but rare) to pass T-types by pointer or allocate them on the heap.

R-types are special types that usually contain handles to a resource that is maintained elsewhere. R-types are similar to T-types in that they can be class members as well as automatic variables. They are like C-types in that they own other resources. Unlike C-types, however, they do not have destructors. Instead anyone using an R-type must explicitly call its Close() method when cleaning up.

Descriptors are EPOC’s way of handling access to and manipulation of strings and general binary data. A descriptor’s data area is not expandable (except for heap buffers) and must be accessed through the descriptor’s methods. Three types of descriptors, pointer (TPtr, TPtrC), buffer (TBuf<TInt>, TBufC<TInt>) and heap (HBuf, HBufC) are provided. Each type of descriptor comes in two different flavours, on non-modifiable (ending with C, like TPtrC) and one modifiable (not ending with C, like TPtr). In addition to this each descriptor type is available in 8-bit (non-UNICODE and general binary data) and 16-bit (UNICODE) variants. TPtr, for example is defines as three different types, TPtr8, TPtr16 and TPtr. TPtr8 and TPtr16 are the 8 and 16-bit versions respectively. TPtr is mapped to TPtr16 if _UNICODE is defines and TPtr8 if not.

When building for several EPOC platforms it is important to choose the right type of descriptors. 8-bits for general binary data and strings that are 8-bit no matter what, 16-bits for strings that are 16-bit no matter what and non-specified for all other strings.

All the T-type descriptors behave like other T-types. HBuf and HBufC must be explicitly deleted, either by User::Free() or delete, thus behaving like C-types.

In addition to the access-safe descriptors described above, EPOC provide type-safe descriptors called packages. A package is similar to a buffer or pointer descriptor. The functionality added by packages allows for mapping between C-structures and descriptors for reading or writing.

Yet another special feature of EPOC is its exception and cleanup facilities. EPOC does not support C++ exception handling. Leave is EPOC’s way of throwing an exception, e.g. User::Leave(KError). Any method that may leave should end with L (like doExampleL). Any call to a L-method should be protected by a trap-harness either directly or higher up in the call-stack. A trap-harness, for example TRAPD(errorcode,doExampleL) is similar to try in C++. If doExampleL (or any method it calls) leaves, control will be passed directly to the nearest trap-harness in the call-stack (using C++ longjmp). The code immediately following a trap-harness should check the error code and take appropriate action, similar to catch in C++.

EPOC's exception handling have far reaching implications for memory cleanup. Any variables on the stack above the trap-harness will be orphaned. This should ideally only happen to T-types. R-types can safely be orphaned only if they are copies of another R-type still available. The application programmer must clean up all other types.

EPOC provides a cleanup stack for handling cleanup. This is especially important when allocation memory within an objects constructor. Let us assume that the constructor of object A successfully allocates an object B on the heap. If some other part of the constructor fails it will leave orphaning all members of A. B, which should normally be delete by A's constructor will now be lost forever. Debug builds of EPOC will detect all lost data when the application exits, allowing the developer to track down all memory leaks.

EPOC handles the above situation gracefully using the cleanup stack. When B is successfully allocated, it should be pushed to the cleanup stack using CleanupStack::PushL(B). If the rest of the constructor is executed successfully, B should be removed from the cleanup stack using CleanupStack::Pop(). If the constructor leaves, the trap harness will automatically empty the cleanup stack, gracefully removing B.

A trap harness should not be used within a constructor unless explicitly needed. Trapping and then (re-) leaving generates more code and is more time consuming than allowing the leave to fall through to the next trap.

Any object that must be cleaned up must derive from CBase. If this is a compound object, EPOC requires a special two-phase construction. This code example, taken from the EPOC documentation, illustrates the two-phase approach.

```
CCompound* CCompound::NewLC(TInt aRoot,TInt aChild)
  { // NewLC with two stage construct
    CCompound* self=new (ELeave) CCompound;
      // get new, leave if can't
    CleanupStack::PushL(self);
      // push onto cleanup stack
      // (in case self->ConstructL leaves)
    self->ConstructL(aRoot,aChild);
      // use two-stage construct
    return self;
  }

void CCompound::ConstructL(TInt aRoot,TInt aChild)
  { // NB. function may leave
    iRoot = aRoot;
    iChild = CSimple::NewL(aChild);
    iChild->iVal = aChild;
  }

CCompound* CCompound::NewL(TInt aRoot,TInt aChild)
  { // version of NewLC which leaves nothing
    // on the cleanup stack
    CCompound* self=NewLC(aRoot,aChild);
    CleanupStack::Pop();
    return self;
  }
```

NewL creates the new object using NewLC and pops the new object from the cleanup stack. NewLC creates the actual object, pushes it to the cleanup stack and initialises the object using ConstructL.

To simplify checking successful construction of object, EPOC provides an overloaded new-operator (new(ELeave) CCompound) indication that this constructor will leave if it does not successfully allocate memory. In addition, normal C++-constructors are not allowed to leave.

Any thread using a cleanup stack must explicitly create it using CTrapCleanup and delete it when it is not needed. CONE (and thus also EIKON) applications does this automatically for its main thread.

Besides the cleanup stack, for instance when exiting an application, all objects must be explicitly deleted. Therefore it is important to clearly define ownership of objects and make sure that all owners destroy the objects they own. Again, leaks will be detected in debug builds of the operating system.

4.2.4 Persistent Storage

Like Windows CE EPOC uses “persistent” to denote data both in battery backed up RAM and actual persistent devices.

EPOC provides two different types of persistent storage. The stream store provides a file abstraction while the DBMS engine provides a functional interface to a relational database. To support this interface EPOC provides a very simple client-side only implementation of the database interface.

4.2.5 Multitasking Features

EPOC is a multitasking operating systems allowing multiple processes and multiple threads per process. Still, one of the most prominent features of EPOC is Active Objects, an object-oriented multitasking abstraction, and its asynchronous programming facilities.

TRequestStatus objects control asynchronous calls in EPOC. The method TInt RFile::Read(TDes8& aDes) is synchronous meaning that active thread runs through the method, returning with the result later. The method void RFile::Read(TDes8& aDes, TRequestStatus& aStatus), on the other hand, return immediately, while the actual processing takes place along another line of control. The original thread must then issue a User::WaitForRequest() and process the value of aStatus, which should have the same value as the return value of the synchronous version.

Active objects is EPOC way of encapsulating the details of this asynchronous programming model. An active object exposes service methods and dispatches requests to service handlers using the asynchronous mechanisms described above. CActive is the base class of all active objects. CTimer is a simple base class for time-based active objects, i.e. objects that should be called on clock-ticks[10].

[10] For user processes, EPOC provides only a low-resolution clock, with a frequency of 10Hz on a PC based platform (emulator) and 64Hz on an ARM based platform. Higher resolution clocks are only available to low-level processes such as device drivers.

All active objects are controlled by active schedulers. An active scheduler handles all the details of waiting for service completion and dispatching control to the right active object through the active objects RunL method. A thread may have only one active scheduler. The execution of requests and results processing is non-preemptive. A practical consequence of this is that RunL methods should be as short as possible.

Incidentally, these aspects of EPOC are covered extensively in the SDK documentation and in Symbian's technical papers[11].

In addition to the active object abstraction, EPOC supports creating and controlling threads, including mutex and critical section primitives. It is important to remember that in EPOC resources are owned by threads, not by a process. This includes the heap, file stores etc.

4.2.6 Remote Access and Synchronisation

EPOC Connect is the name of Symbian's generic product for remote access and synchronisation. Device manufacturers license this product and sell it under their own name, such as Psion with their PsiWin.

EPOC Connect has several different attributes:

1. An application integrated into the Windows environment facilitating drag and drop file copying, file conversion, synchronisation, backup/restore, device management etc.
2. A COM-based architecture for including new converters, synchronisers etc.
3. A COM-based architecture providing access to EPOC engines from Windows (WINC). The COM-interfaces are similar to the native EPOC interfaces.

For developing converters, synchronisers, etc. utilising EPOC Connect Symbian provides a separate EPOC Connectivity SDK.

4.2.7 Development Environment and Language Support

Symbian's EPOC SDK provides the main EPOC development environment. It provides a separate emulation environment for testing and demonstration purposes (development using Visual C++ 5 or 6). The bundled GNU C++ compiler provides target platform compilation. The OS itself is wholly object-oriented, not modelled after other OS "types" such as WIN32 or POSIX. To ease porting, most stdlib functions are available on top of the native EPOC interface. This means that using standard C functions will impose an overhead. In addition the stdlib implementation preserves state across calls. This means that every thread using the stdlib must explicitly close it when exiting.

In addition to the C++ environment EPOC provides a full Java environment. This is integrated into the existing EPOC platform. Current implementations still suffer from bugs, but fixes are expected.

[11] See for instance http://www.symbian.com/technology/papers/active/active.html.

To complement C++ and Java, EPOC offers its own scripting language, OPL. This can for instance be used to automate common tasks etc.

4.2.8 Supported CPUs (ER5)

MARM: ARM 7100, StrongArm
MTHUMB: not yet supported, will build for ARM Thumb
MCORE: not yet supported, will build for Motorola M340

4.3 Palm OS

Palm OS is a lightweight operating systems especially designed for small devices. It is currently owned and controlled by Palm, Inc.

Unlike Windows CE and EPOC, Palm OS does not attempt to provide a generic OS for mobile platforms. Instead the focus is on the creation of small, specialised and extremely efficient application such as calendar, contacts database etc. Palm OS does not provide support for threads or more than one running process at a time. However, the small applications together with the application launch architecture provide the user with the feeling of having several applications alive and running.

The extremely narrow focus of the Palm platform is a limiting factor. Still, Palm devices constitute the majority of PPC-type devices, indeed the majority of all PDA-type platforms. Rumour has it that Palm is seeking to license its user interface to be used on top of EPOC (or other suitable operating systems). As with the other platforms, device manufacturers and market impact will have profound influence on the future development of the platform.

4.3.1 Modularisation

The only modularisation features provided by Palm OS are application and device driver. No support for dynamically linked libraries exists. This coupled with the extremely tight memory requirements for most Palm OS devices means that the application programmer should focus on using the OS-features as much as possible.

Device drivers are small, specialised programs controlled by the OS' communication manager. Serial device drivers provide a byte-oriented hardware abstraction layer concealing details of different UARTs etc. Virtual device drivers provide a block oriented interface, usually on to of a serial device driver.

4.3.2 Memory Architecture

The memory architecture in Palm OS is structured around storage cards. Each storage card may have ROM, RAM and flash memory. A 256 MB address space is set aside for each storage card. All available memory on a storage card is divided into heaps. RAM is divided into storage RAM and dynamic RAM. Dynamic RAM consists of a single heap. Storage RAM consists of one or more dynamic heaps.

If more than one storage card is available, only the dynamic heap of the first storage card is used. All other potential dynamic heaps are ignored. This, and the fact that a single heap is limited in size, mean that heap memory is an extremely scarce resource in Palm OS devices. Do not use heap space unless you really, really need to. Use the "edit in place" functionality of database to minimise memory usage.

Palm OS uses 32-bit addresses, with 8, 16, and 32 bit data. Current implementations use a 16-bit external data bus. The Palm OS documentation explicitly states that the different OS interfaces should be the design target of any Palm OS application, not specific details about word lengths, addresses, memory sizes, etc.

4.3.3 Memory Management

Palm OS provides only low-level memory management functions. Most is left to the application developer.

Memory is allocated in chunks using the memory manager. A chunk is part of a heap, either a ROM heap, dynamic heap or storage heap. The memory manager should be used for all allocation of chunks. Current implementations limit the size of a single chunk to 64KB. Chunks used to hold storage data is handled as a record in a database. Databases are accessed through the data manager. Cards are treated as separate, meaning that heaps, chunks and databases are limited to a specific card.

Chunks can be non-movable (fastest, but limits flexibility) or movable (most flexible, preferred). Non-movable chunks are referenced by a direct pointer. A handle in a master pointer table is used to reference movable chunks. Handles are not persistent and should not be stored in a database.

4.3.4 Persistent Storage

Databases are the closest Palm OS comes to a file abstraction. For access to large blocks of data, Palm OS provides a file streaming API, derived from stdio, on top of databases. The file streaming API uses a double-buffering architecture which significant performance impacts. Record-intensive applications tend to obtain better performance from the Data Manager.

Palm OS does not provide a conventional file system. It provides a flat name-space for storing applications (.prc-files) and database (.db-files). All access to stored data is through the data manager, the resource manager or the file streaming API.

Current Palm OS devices have very limited flash-memory support. The only available offering is a springboard module providing 8 MB of memory. ROM-modules are also available, providing an external medium for carrying applications and other static data.

4.3.5 Multitasking Features

Palm OS does not support threads or multitasking. Only one application process may be active at any time. The only execution environment is the system event loop. This part of the OS kernel catches all interrupts and events and delivers them to the correct driver, kernel module or application.

Palm OS' application launching architecture provides the user with the notion of having several applications running by implementing a crude form of task switching. When the OS event handler detects and event for another application than the one that is currently active, the active application receives an appStopEvent. This event signals that application should store whatever state it needs in order to be restarted in its current state. The application must then stop. The OS starts a new application with a specific launch code. Depending on the launch code this newly started application may retrieved its old state, giving the impression of having been dormant.

4.3.6 Remote Access and Synchronisation

Unlike EPOC and Windows CE, Palm OS does not provide any remote access mechanisms other than low-level network and serial (including infrared) communication.

Synchronisation is provided by the HotSync infrastructure. Synchronisation is manual, activated by the HotSync-button on the device. Hitting the button provides both database synchronisation (integration of data into desktop applications) and backup (copy of databases on PC).

The HotSync architecture uses special plug-ins called Conduits for moving data between desktop applications and Palm databases. HotSync uses the application name to determine which conduit to use for synchronisation. When synchronisation starts (and finishes), application registered for HotSync notifications (start or end) are launched one at a time using a special launch code.

4.3.7 Development Environment and Language Support

Palm's primary development environment is based on C using the Codewarrior integrated development environment. Standard C routines are supported but must be linked into every program. Using Palm specific routines instead is recommended. It is also possible to develop Palm OS applications using the GNU gcc/g++ compiler.

In contrast to EPOC and especially Windows CE, the Palm platform have been heartily embraced by the free software community, meaning that both applications and software development tools are readily available, for instance a Pocket C environment similar to that for Windows CE, and WABA, a lightweight Java environment.

The platform itself does not provide any scripting facilities, but some standalone products are available.

4.3.8 Supported CPUs

Motorola 68K

4.4 VxWorks

VxWorks is a real-time multitasking operating system for embedded systems. It is provided by Wind River Systems. The operating system itself can be tailored to include only the components necessary for a given application, resulting in a very small and highly efficient environment.

4.4.1 Modularisation

VxWorks' focus on embedded systems affects the modularisation available. Normal target platforms usually employ a single monolithic application. VxWorks provide a single process that may have one or more threads. Code executing within a thread is called a task.

4.4.2 State Management

VxWorks do not provide a state management model similar to the one described under *Common Aspects*. When the system is turned of, it is really off.

4.4.3 Memory Architecture

VxWorks provides a single, linear memory space. To ease development of memory demanding applications virtual memory is supported. To achieve the necessary real-time requirements, no paging or swapping is performed during run-time. The standard virtual memory architecture does not support memory page locking or memory protection. The VxVMI option provides memory page locking, exception table locking, as well as an architecture independent MMU interface.

A RAM disk driver is available, but this does not support memory-mapped files. Applications requiring memory mapped files or other forms of caching must implement it themselves.

4.4.4 Memory Management

From the VxWorks documentation: “Memory Allocation: VxWorks supplies a memory management facility useful for dynamically allocating, freeing, and reallocating blocks of memory from a memory pool. Blocks of arbitrary size can be allocated, and you can specify the size of the memory pool. This memory scheme is built on a much more general mechanism that allows VxWorks to manage several separate memory pools.”

VxWorks provide standard C and C++ memory management functions. In addition libraries for handling linear and ring buffers, linked lists etc. are part of the standard package. The application developer has the full responsibility for protecting, deleting and controlling memory, data types etc. The operating system kernel assumes that all tasks are well behaved and correctly implemented. To help developers achieve this, Wind River Systems provide testing and evaluation tools and environments.

4.4.5 Persistent Storage

VxWorks provide an abstract Local File System that may be realised using different actual file systems such as FAT, NTFS etc. The OS also supports development of drivers for a wide range of storage types, including flash memory, hard drives and CD-rom. Again, memory-mapped files is not part of the basic OS capabilities.

4.4.6 Multitasking Features

VxWorks is built around the wind kernel. This optimised kernel provides priority based multitasking (round-robin can be selected) with strong priority control functionality. Comprehensive thread support in the form of tasks is provided through the task library (taskLib).

A wide variety of inter-process communication facilities is offered. This includes message queues (both wind and POSIX), pipes, socket/RPC, and UNIX-style signals. Shared memory objects are provided through the VxMP option. A wide range of semaphores is also available, including binary semaphores, mutual-exclusion semaphores and POSIX semaphores.

In addition to the inter-process facilities VxWorks provides strong interrupt handling. This includes allowing task to disable interrupts during specific code segments using interrupt locks. Task-locks is a milder for of lock protecting the executing task from most context switches, except hardware interrupts.

4.4.7 Remote Access and Synchronisation

VxWorks does not provide any explicit synchronisation architecture.

4.4.8 Development Environment and Language Support

Development of VxWorks applications is based on UNIX-style C and C++ development, including full ANSI C and POSIX support. Compilers include GNU gcc/g++ as well as specialised components as part of the Tornado Development environment.

4.4.9 Supported CPUs

Motorola MC680x0
Sun SPARC, SPARClite
Intel i960
Intel x86
MIPS R3000, R4000, R4650
PowerPC
ARM
(VxSIM)

5 Summary

This section gives a brief summary of this report.

5.1 Future Trends

The general trend is a move to include more and more functionality into mobile devices. It may seem that a general consolidation is emerging, focusing on a limited number of physical formats, such as mobile phones (smallest), smartphones and palmsized PDAs (good screen, no keyboard), communicators and handhelds (good screen and keyboard). Larger sizes will ultimately compete with notebooks using traditional operating systems.

Currently, Palm OS has the largest market share, with Windows CE trying to establish itself as the only serious alternative for advanced applications, and Symbian playing on its powerful partners within the mobile phone industry.

Microsoft is in definitive control of Windows CE, having no clear (or at least very shifting) strategy other than wanting to dominate the market. Device manufacturers (licensees) are bound by special conditions (marketing, labelling etc). Expensive licensing gives expensive devices. Expensive development environment has hindered independent development (most applications must be paid for). Not so good GUI (esp. Palmsize) have lead to bad user image. A new image (Pocket PC), much cheaper toolkits, stronger kernel etc are all designed to improve Microsoft's mobile profile, hoping to gain on Palm (the current mass-market leader) and EPOC (the so far uncontested mobile phone champion – without any available devices!).

For EPOC, where the Symbian – the OS developer is controlled by the most influential licensees the co-development of devices and operating systems seem stronger. Still, the lack of new available devices makes it difficult to judge EPOC's future expectations.

For Palm OS one company in control of OS and most devices, although there are some licensees. There have been rumours that the Palm OS as such to be abandoned and GUI built on top of EPOC. Other rumours state that Palm OS is superior in its focus on small, efficient applications. It Palm OS is to keep its dominating market share Palm must counter the best of the other contenders while keeping a competitive edge inventing their own, new features.

For special purpose, especially critical systems, traditional embedded systems like VxWorks still hold the competitive edge. Here, an interesting development will be how the marked based on Microsoft's embedded offerings, both Windows CE and Embedded NT, will develop.

5.2 Aspect Summary

In practice, the different operating systems described have some common features, but with significant differences. Most can be programmed using C or C++ and provide some sort of stdlib-support. The most important differences are EPOC's memory management and Palm OS application architecture.

Windows CE is a general purpose operating providing a environment very similar to other WIN32 operating systems. It does not have an explicit focus on resource constraints or real-time features. EPOC is also a general-purpose operating system with special focus on resource constraints such as multitasking and memory management. Palm OS, on the other hand, is designed for special purpose applications, small, simple and effective. Palm OS has no explicit focus on resource constraint, but only low-level memory management support forces every application developer to handle most resource problems himself. VxWorks is a special purpose operating system focusing on real-time applications.

6 References

Microsoft Windows CE Developer’s Kit (documentation for Windows CE Toolkit for Visual C++ 6.0).
Microsoft eMbedded Visual Tools 3.0

EPOC 5 SDK
EPOC Connectivity SDK
Programming Psion Computers
http://www.symbian.com/technology/papers/papers.html

Palm 3.5 SDK Doc (Palm OS Companion, Reference)

VxWorks Programmer’s Guide 5.4
VxWorks Reference Guide 5.4

CF+ and CompactFlash Specification Revision 1.4
IBM MicroDrive Technical Specifcation
Lexar Media CF Technical Specification
Sandisk CF Technical Specification
Kingston CF Technical Specification