The Extrusion of Software Components to Develop Mobile Internet Software Applications

By

Steve Clemons

A MASTER OF ENGINEERING REPORT

Submitted to the College of Engineering at
Texas Tech University in
Partial Fulfillment of
The Requirements for the
Degree of

MASTER OF ENGINEERING

Approved

Dr. J. Smith

Dr. A. Ertas

Dr. T. T. Maxwell

Dr. M. M. Tanik

June 29, 2001

ACKNOWLEDGEMENTS

I would like to extend a sincere thank you to Dr. Artila Ertas and each of the professors and instructors within the IDEATE program, for sharing their invaluable knowledge with each of use during the past year. I wish their program continued success in the future.

My gratitude also goes to Raytheon Garland, for hosting a program so important to the advancement of the design and integration of software and hardware systems.

Additional thanks goes out to the members of the Tcl newsgroup (comp.lang.tcl), who have provided me a wealth of information, not only for the purpose of this report, but over the last 2 years that I have spent as a Tcl programmer.

Lastly, I would like thank my family, including my wife Malia, daughter Kiana, and son Makai, for their unconditional love and support, and their selflessness and sacrifice during my studies.

TABLE OF CONTENTS

Page

ABSTRACT

The use of the Mobile Internet continues to remain low among Internet users. One major reason is the lack of content in mobile software applications. Small bandwidth, weak CPUs, and low memory availability has prevented Mobile users from enjoying the same programs that they find on their home or work PCs.

While the hardware industry continues to make improvements to wireless communication networks and the mobile devices themselves, the software industry is modifying existing programming languages to help overcome hardware restrictions imposed upon the Mobile Internet.

This report explores the extrusion of attributes of software components to improve performance when used in conjunction with the Mobile Internet. Database, application, and scripting languages are studied. The speed and memory usage differences over the languages' non-extruded applications are determined.

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Introductory Remarks

You're on your way to the airport to pick up your boss, but you're running late. You called the airline ahead of time to check the arrival gate information, gate B-32. As you near the airport, your pager goes off. It's a message from the airline. The gate has changed to A-6. No problem, you pull up just as your boss is leaving the baggage claim.

Passing through the mall's main entrance, you log on to the mall's website with your cell phone. You send out a notice that you're looking for the new Harry Potter novel. Two merchants immediately reply that they are out of stock, a 3rd says they have copies available, while a fourth merchant says they have it in stock and while charge you 10% less than what Merchant 3 is asking.

Young Johnny sits quietly in his bus seat for the ride home. He downloads the newest level to his favorite game from the developer's website to his PDA. He installs the program and spends the rest of the trip trying to get through the maze of creatures.

Is this somebody's idea of the future of the Internet? No, this is the Internet, and the future is now. The Net has gone mobile and these types of services are available or will be in the very near future. So why isn't everyone scraping their desktop for a Palm Pilot? There are many reasons holding consumers back from welcoming the Mobile Internet with open arms.

There might be the infrastructure and the technologies in place, but what will attract consumer interest and demand is going to be content and applications. It is not just any content or applications, but only those that will appeal to end-users, create value for them and meet their personalized needs. Content and applications are crucial to the take-off of Mobile Internet and it is an area which industry players will need to get involved in directly or indirectly.

Developers are dealing with a new environment, one that has greater restrictions and limitations than what has been the norm. Compared to desktop systems, the portable devices that users will be accessing the Mobile Internet on are quite limited, especially memory. A desktop

PC might typically have 128MB RAM (*memory*) and 20GB hard disk (*storage*). A portable wireless information device might have 16MB RAM which serves as both memory and storage. Thus, it is inevitable that end-users will occasionally run out of memory, or storage. Memory allocations, or disk writes, will fail. Creating the content and applications for the Mobile Internet requires a different set of tools than for the wired Internet.

The tools available to programmers involve minimizing memory usage. Some of these include program architecture, secondary storage, compression, the use of small data structures, and careful memory allocation.

Others have taken a different route to memory management. They have started to take existing programming languages and have stripped them of unneeded components to match their targeted platform. Like a pasta company pushing a batch of dough through an extruder, stripping away unwanted dough and coming out with perfect angel hair pasta, software components are being extruded to limit their memory usage.

HISTORY OF WIRELESS

Origins of Wireless Networks.

Wireless networks and software modularization are nothing new. The humble beginning of wireless services goes back to the 19th century, a time when Guglielmo Marconi, "the father of radio" made his mark in the world of wireless technology.

When Marconi started experimenting with radio waves (Hertzian Waves) in 1894, his objective was to produce and detect radio waves over long distances. In 1896, Marconi was successful and obtained a patent and established the Wireless Telegraph and Signal Company Limited, the first radio factory in the world. In 1901, signals were received across the Atlantic and in 1905 the first wireless distress signal was sent using Morse Code.

Wireless technology eventually progressed as an invaluable tool used by the U.S. Military. The Military configured wireless signals to transmit data over a medium that had complex encryption, which makes unauthorized access to network traffic almost impossible. This type of technology was first introduced during World War II when the Army began sending battle plans over enemy lines and when Navy ships instructed their fleets from shore to shore.

Wireless proved so valuable as a secure communications medium many businesses and schools thought it could expand their computing arena by expanding their wired local area networks (LAN) using wireless LANs. The first wireless LAN came together in 1971 when networking technologies met radio communications at the University of Hawaii as a research project called ALOHNET. The bi-directional star topology of the system included seven computers deployed over four islands to communicate with the central computer on the Oahu Island without using phone lines. And so, wireless technology, as we know it, began its journey into every house, classroom, and business around the world.

In 1987, the FCC allowed and encouraged cellular service providers to use alternate technologies in the 800MHz spectrum of radio frequencies. One goal was to employ digital transmissions, a feat already accomplished by the Europeans. The first digital standard tested in the US was Time Division Multiple Access (TDMA), and by 1996 three digital standards were being used by a variety of carriers in North America. TDMA is used by AT&T Wireless, and Southwestern Bell. CDMA, or Code Division Multiple Access, is used by Sprint PCS, GTE, Air Touch, and Bell Atlantic. GSM, or Global System for Mobile Communications, is used by Pacific Bell and Omnipoint. The three systems differ in the way that they process voice signals and encode information, but to the end user they sound virtually identical.

In 1994, the FCC also opened up a spectrum of radio frequencies for Personal Communication Services (PCS) operating at 1900MHz. Unlike cellular services operating at 800MHz that often included both the analog AMPS technology and digital TDMA, CDMA, or GSM, the PCS spectrum was entirely digital. Once the 800MHz and 1900MHz spectrums were established and the various digital systems were in place, advanced features such as voice mail, fax capability, paging, and SMS (Short Messaging Service) became available to cellular users. Also, cell phones could be used as wireless modems for Internet connection from a laptop while on the go. These advanced services and capabilities paved the road for the eventual direct connection between the wireless world and the World Wide Web.

Probably the most important factor in the birth of mobile Internet has been the proliferation of digital cell phones in the last few years. The expanding network of digital cellular and personal communication services (PCS) has created a solid foundation for mobile Internet services. It is estimated that there are more than 50 million Web-enabled cell phones in use. In 1997, Nokia, Motorola, Ericsson and Phone.com came together to create the WAP because they believed that a universal standard is critical to the successful implementation of mobile Internet. Since then, more than 350 companies have joined them in the WAP Forum.

Making a Web site accessible through a mobile device is quite a challenge. So far, only a small portion of the more than a billion Web sites, about 1.5 million, provide any mobile Internet content. As the use of WAP-enabled devices grows, you can expect that many more Web sites will be interested in creating mobile content.

The Mobile Internet

Arrival of the Mobile Internet

After several years of predicting the emergence of the Mobile Internet, this market is finally upon us. Built upon the economic success and broad user base of both the Internet and the mobile communications market, the mobile data and Internet market is expected to be even more dynamic, pervasive and evolutionary. There is now a range of related technologies and platforms to support the successful implementation and management of Mobile Internet services. But that alone is not sufficient to create mass demand. More importantly, there is now a broad and massive base of attractive and relevant content and applications to support, shape and grow this new market.

In the area of standards and infrastructure, there is now a stable set of network technologies, protocols access standards and terminals upon which the Mobile Internet market could build upon. Much of the current mobile network technologies and infrastructure are being leveraged to support mobile data and Internet services. Looking ahead, there is also the progression towards packet data and high bandwidth 3G networks, allowing for the provision of even more bandwidth intensive and interactive services. The much awaited WAP protocol has been introduced, providing a stable and common environment to base and develop the Mobile Internet. Equipment manufacturers have infrastructure equipment in place, as well as an ever increasing range of WAP-enabled terminals to present information to the user in different ways dependent upon their needs and applications.

The explosive content and applications market has created the most interest and rightly so. There might be the infrastructure and the technologies in place, but what will attract consumer interest and demand is going to be content and applications. It is not just any content or applications, but only those that will appeal to end-users, create value for them and meet their personalized needs. Content and applications are crucial to the take-off of Mobile Internet and it is an area which industry players will need to get involved in directly or indirectly.

The Mobile Internet market has undergone tremendous changes in the past year. The structure of the industry has found it necessary to evolve and adapt in order to meet the opportunities and challenges brought about by market changes. Industry players are pushed to expand beyond their traditional market segments to maintain their current revenue streams and generate new ones. The increasingly complex and competitive market has made it necessary for players along the new value chain to form alliances to maximize synergies and economies of scale and reduce risk and service deployment time. To survive and thrive in this new market space, players have to review their position along the value chain and strategies accordingly.

Mobile Internet Development

The market for wireless data usage and growth will be leveraging on the mass market base of both the Internet and mobile service markets. The Internet market is reaching its growth maturity stage and though the growth rate has slowed compared to past years, the number of new users being added is still very substantial, especially in the Asia Pacific region. The US is still the leader in Internet users and that is expected to continue till around 2003 when its penetration will be close to saturation and it will be surpassed by Western Europe.

The mobile market is enjoying strong growth across all markets with Western Europe in the lead, making up 35% of total mobile subscribers. In 2003, Asia Pacific is expected to overtake Western Europe as the biggest mobile market, mainly boosted by the buoyant growth of the Chinese mobile market.

The pace of development in the area of fixed Internet varies from region to region and the same applies to that for wireless data and Internet. This difference in the rate of development can be attributed to factors such as size of population, regulatory and business environment, technology adoption, social and cultural norms. Most wireless data users will be accessing data services from handsets, while others will be accessing through PDAs, laptops and other mobile devices.

Currently, the US has the highest penetration for the Internet but one of the lowest for mobile data. This is due to the sluggish take-up of mobile data services in the country as it is still very much fixed Internet-based at the moment. In Western Europe where Internet penetration is not that high but where there is a booming market for mobile services, the usage of mobile data services such as SMS messaging thrives.

In the next few years, mobile data penetration will be close to mobile user penetration in regions like the US, Western Europe and Japan. It is expected then that all mobile terminals will be
data-enabled and that subscribers will be able to seamlessly access mobile data and Internet services. In all regions, mobile data penetration is expected to exceed Internet penetration, supported by the larger base of mobile users as compared to PC users.

Mobile Content and Applications

As mentioned earlier, the availability of attractive and relevant content and applications will be instrumental to the take-off and success of Mobile Internet. Key to the success and appeal of applications will be the degree to which they can be personalized and to which they meet the unique characteristics of mobile access: timeliness of data provision, accessibility away from the home or office, and relation of information to a user's location. The user has to see a benefit over and above the technology itself.

There will not be a single killer application but a basic quality bundle of information services, personalized by the user, and taking into consideration all available information about the user, including his location. The bundle will be seen by the user as a must have, relying on additional service offerings to differentiate between operators. These applications will only require the transmission of very small quantities of data.

Handhelds have gone beyond their initial function of replacing paper organizers. They've graduated to support an impressive range of important business functions in an enterprise setting. Common applications include: sales force automation, inventory management, inspections, field service automation, construction management, warehouse applications, public construction monitoring, facilities management, local delivery management, local haul fleet truck management, outage management, point of care applications, law enforcement, scientific data collection, and production data capture.

Some of the most widely noted drivers in the growth of mobile computing in the workforce are:

- The need for faster, decentralized decision making

- The need to be closer to customers, prospects, and partners
- The availability of better mobile computing technology
- Mobile workers spend time working, less time commuting
- Mobile workers report less stress, greater job satisfaction
- Mobile workers make fewer healthcare/accident claims
- Using mobile workers opens up a much larger recruiting pool
- A distributed workforce can reduce travel costs
- Mobile workers decrease the need for expensive office space
- The increasingly global economy
- Increased responsiveness to customer service needs

These dynamics are driving rapid growth in the number of mobile workers. The challenge facing upwardly mobile corporations – and particularly their IT departments -- is to provide the same level of access to corporate information to the mobile worker who typically lacks a dedicated high-bandwidth network connection.

Overcoming this challenge is a pre-requisite to realizing the competitive advantages that mobile computing can bring to the modern corporation.

Location based services are going to play a significant role in the mobile data. Eventually, all mobile devices will be data-enabled and access to location and navigation services will be seamless and invisible to the end-user.

Mobile commerce (m-Commerce) is expected to be a major revenue generator. This current lack of confidence in the prospects of m-Commerce in the short term can be attributed to the lackluster performance of the WAP platform and the voice-centric networks now in terms of speed and capability, resulting in m-Commerce being a tedious affair rather than a convenience. Secondly and more importantly, m-Commerce has inherited the payment and security issues that have dogged online e-commerce. Most of the m-Commerce services currently in the market are carried over the SIM Toolkit platform which has better security features than WAP.

m-Commerce applications will mostly mirror online e-Commerce in terms of types of offerings but they will be characterized by factors such easy accessibility, timeliness and location sensitivity to complement and enhance the mobility factor. Providers will need to leverage and optimize these characteristics in creating and promoting their m-Commerce applications in order to attract customers and differentiate themselves. However, differentiation will be short-lived as providers will build up more or less the same array of m-Commerce applications. In order to maintain an edge, providers will have to continually look at ways to widen and deepen their scope of m-Commerce offerings and facilitate easy accessibility and payment systems. Tie-ups with banks and credit card companies are becoming crucial as they will serve to lessen the complexity and payment and security issues involved in m-Commerce transactions.

Mobile Network Technologies

Developments are in place for the advancement of each of the major network technologies to support higher bandwidth data providing a full evolutionary path from messaging services (e.g. SMS) through intermediate network enhancements (e.g. GPRS) through to full 3G (Third Generation) architectures. The data system needs to be able to accommodate these changes, which means that it should interface directly with an IP based packet network, and place no specific requirements on the air interface.

3G is a new radio communications technology that will create a "bit pipe" for providing mobile access to internet-based services. It will enhance and extend mobility in many areas of our lives.

Table 1 illustrates the current and future capabilities of mobile networks. In the near future, mobility won't be an add-on: it will become a fundamental aspect of many services. We'll expect high-speed access to the internet, entertainment, information and electronic commerce (e-commerce) services wherever we are - not just at our desktop computers, home PCs or television sets.

3G services will add an invaluable mobile dimension to services that are already becoming an integral part of modern business life: Internet and Intranet access, video-conferencing, and interactive application sharing.

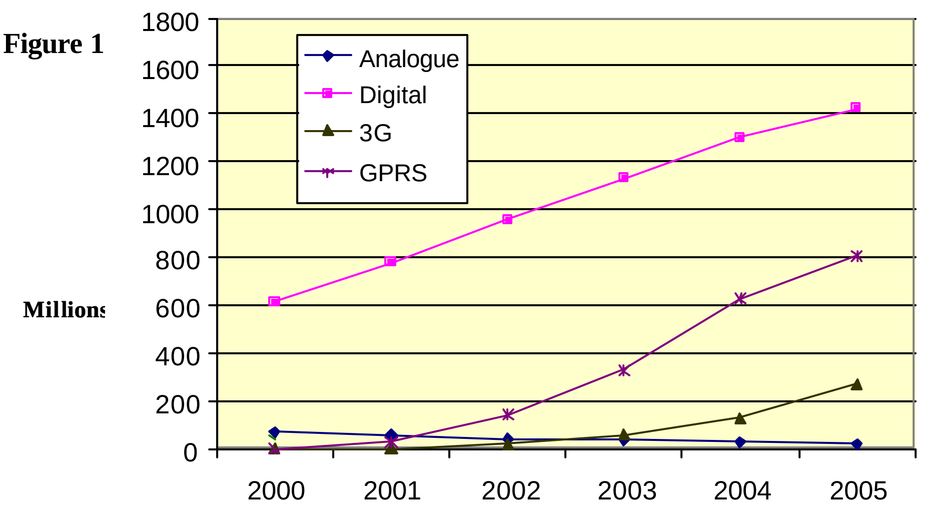
Table 1. Current and Future Wireless Capabilities *(Source: Newsweek)*

2G Wireless	2.5G Wireless	3G Wireless
The technology of most current digital mobile phones	**The best technology now widely available**	**Combines a mobile phone, laptop PC and TV**
Features includes: - Phone calls - Voice mail - Receive simple email messages	**Features includes:** - Phone calls/fax - Voice mail -Send/receive large email messages - Web browsings - Navigation/maps - New updates	**Features includes:** - Phone calls/fax - Glbal roaming - Send/receive large email messages - High-speed Web Navigation/maps Videoconferencing - TV streaming - Electronic agenda meeting reminder.
Speed: 10kb/sec **Time to download a 3min MP3 song:** 31-41 min	**Speed:** 64-144kb/sec **Time to download a 3min MP3 song:** 6-9min	**Speed**: 144kb/sec-2mb/sec **Time to download a 3min MP3 song:** 11sec-1.5min

Mobile Internet based data does not require large amounts of traffic to be communicated for the majority of users. A minority will make use of Internet browsing and intranet access over high bandwidth bearers. The introduction of a 3G network is definitely not a prerequisite for the introduction of valuable data services. The principal success sector for 3G is for Internet and intranet access. Figure 1: below illustrates the forecasts for worldwide mobile subscribers by technology.

Figure 1

ce: ARC Group)



Regardless of mobile network bandwidth capabilities, the devices that content are going to be delivered to still lack the processing speed and memory capabilities to run the high-end applications that users will demand.

Mobile Internet Platforms

Equipment needs to be deployed by the network operator to overcome problems such as limited capabilities within the client terminal, low bandwidths, high latency of mobile networks, noise, and call dropout associated with wireless networks. These problems are typically eliminated through data compression and the intelligent handling and prevention of dropped calls by a proxy server, many of which will reformat and filter data according to the characteristics of the user's terminal. An environment for the management of data via a proxy to the terminal is specified within the WAP standards (the Wireless Application Environment).

Middleware is an intermediate software architecture that translates web content to match the delivery medium and the terminal characteristics. Software agents which can automate some of the user's tasks and simplify their actions to find the information they want will be critical in enabling the personalization of services. These software applications will be located in the terminal, the operator's infrastructure system, and on the Internet.

At the stage where standards were not available to define a suitable protocol for the communication of Internet content to a mobile client proprietary standards had to be developed. These include Phone.com's Handheld Device Markup Language (HDML) and Access Technology of Japan's Compact HTML (C-HTML). The Wireless Application Protocol (WAP) markup language, WML has now come into the market with strong global backing. WML is expected to be the dominant delivery protocol in the short term but is likely to be superseded by XML, which works both for the fixed and Mobile Internet.

The introduction of small footprint versions of Java: Personal Java, Embedded Java, and JavaCard, will enable the integration of software control to handsets. It will be possible to update applications on a smart card, or within the terminal, over the air. Applets can also be incorporated into web content to interact with the phone, or the network. The WAP standard includes Java like elements, in their WMLScript language which can be embedded in WML pages.

Access Technologies, Standards and Terminals

In the short to medium term, the access and terminal market for the provision of Mobile Internet services will be very much defined and driven by the Wireless Application Protocol (WAP). WAP is a global and open communications protocol and application environment that empowers mobile users to gain access easily and interact with information and services utilizing their mobile devices. It is designed to be used by most handheld devices ranging from pagers, phones to communicators and to work over most wireless networks such as GSM, CDMA and TDMA. It can also be built on any operating system in the market today, such as Palm OS, EPOC and Windows CE.

Terminals that might be used for mobile access to Internet based services vary widely in their capabilities. The elements that will have the greatest influence on the formatting of data are the display, memory, processor speed, and user interface. Terminals that will be used range from a basic phone, to a full color VGA display notebook computer. The applications will often be the same though optimzed use will depend on the capability and characteristics of individual terminals.

The present terminal market can be segmented into three markets. The first of these is phones which are segmented by the value and capabilities of the phones. The second market is for organizers and includes PDAs (no keyboard, portrait screen), and handheld PCs (keyboard, landscape screen). The third market is for portable computers including notebook, small notebook, and ultralight notebooks.

These categories will change as the difference between products becomes blurred. At the lowest level there will be the entry level phone, which has a budget price and very limited push information service support. The next level is the value added service mobile (VAS mobile) with a larger display and the ability to support WAP applications and secure e-Commerce. The communicator will become a product category in its own right though it will encompass the PDA whose functionality will become increasingly communications biased, but with a large screen and powerful processor. Finally the handheld PC will become increasingly powerful and will grow to have much of the functionality of today's notebook computers though in a much smaller form factor.

The introduction of Bluetooth's local wireless links between mobile and fixed products will add a new degree of flexibility to the use of mobile data. Devices can make use of personal information stored on other local devices, no wires are needed between computer and mobile phone for data access, and devices can synchronize personal information automatically.

Mobile Internet Industry Structure and Strategies

The structure of the traditional telecoms industry has undergone significant changes with the convergence of industry segments, brought about by the consolidation and expansion of players' roles in line with technological and service evolution. The industry structural evolution looks set to reach greater heights and complexity with the emergence of Internet services delivered over mobile platforms, powered by the growth dynamics of both the Internet and mobile sectors and the market opportunities that they generate.

The emerging Mobile Internet market has brought with it opportunities derived from the mobility of the platform and the varied content and services mirroring and enhancing that of the highly successful fixed Internet. The challenges facing market players are how maximize value and revenue generation from the evolving industry chain and how to remain relevant to the consumers and retain their loyalty, in a cost and time efficient manner. Attributes such as competitive advantage and differentiation have become even more crucial. The resultant market scenario is one where market players are pushed to extend their involvement beyond their traditional service segments and involve themselves in other segments along the industry value chain through acquisitions and partnerships. These movements are done with a view to establish a synergistic and complementary relationship to enhance competitive advantages.

The market evolution brought about by Mobile Internet has altered the position of the network operator in the value chain. Up till now, the operator has been able to rely upon per minute/second billing of voice traffic for revenue. The introduction of data requires a change to this model. Data traffic will not generate large amounts of traffic in the network, though there are opportunities for induced services where the data service provides an alert for further information that is accessible by calling into a voice message server. Data services will initially provide for operator differentiation though retention of these customers will develop to rely upon personalization of services, and a battle for customer ownership.

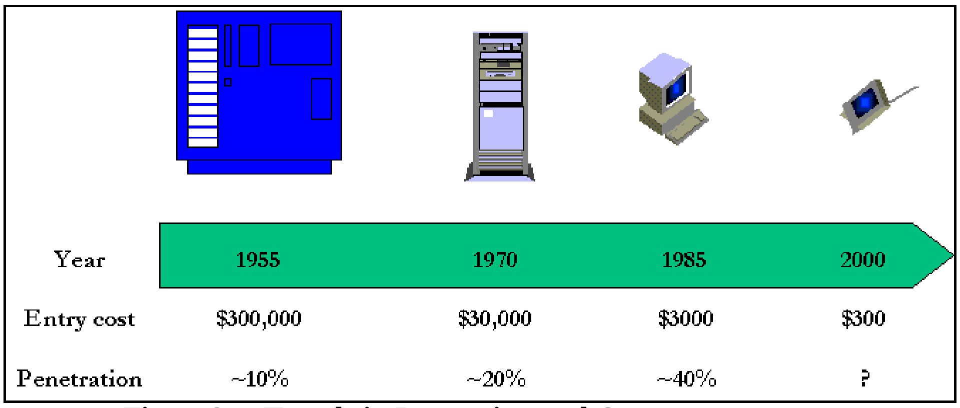
Device Limitations of the Mobile Internet

To discuss limitations, the scope of target small information appliances must first be described. The categories of these devices are often referred to as smart phones, smart communicators, and mobile PDAs. There are some hardware restrictions for these devices. Although low bandwidth is a major limitation of the Mobile Internet, there are other limitations that are device specific. From the hardware point of view, developers have to applications that have:

- Small memory. Typical case: 128-512Kbytes RAM and 512K-1Mbytes ROM
- Low power CPU. Typical case: 1-10 MIPS class CPU for embedded systems
- Small display. Typical case: 50x30 dots, 100x72 dots, and 150x100 dots
- Restricted colors. Typical case: mono-color (black and white)
- Restricted character fonts. Typical case: only single font
- Restricted input method. Typical case: several control buttons and number buttons (0-9)

Trends In Processing And Costs

Most everyone is familiar with the general trend of processing power becoming cheaper (from mainframe to desktop computer). This trend also translates to an increased penetration rate for end-user computing. PDAs are going to be the next big wave where because of reduced cost and smaller form factor, an unprecedented number of end-users will have access to mobile enterprise applications. Figure 2: illustrates then trends in processing and costs with relation to time.



Year	1955	1970	1985	2000
Entry cost	$300,000	$30,000	$3000	$300
Penetration	~10%	~20%	~40%	?

Figure 2: Trends in Processing and Costs *(Source: Oracle)*

MEMORY

The Role Of Memory In The Computer

People in the computer industry commonly use the term "memory" to refer to RAM (Random Access Memory). A computer uses RAM to hold temporary instructions and data needed to complete tasks. This enables the computer's CPU (Central Processing Unit), to access instructions and data stored in memory very quickly.

A good example of this is when the CPU loads an application program - such as a word processing or page layout program - into memory, thereby allowing the application program to work as quickly and efficiently as possible. In practical terms, having the program loaded into memory means that you can get work done more quickly with less time spent waiting for the computer to perform tasks.

The process begins when you enter a command from your keyboard. The CPU interprets the command and instructs the hard drive to load the command or program into memory. Once the data is loaded into memory, the CPU is able to access it much more quickly than if it had to retrieve it from the hard drive.

This process of putting things the CPU needs in a place where it can get at them more quickly is similar to placing various electronic files and documents you're using on the computer into a single file folder or directory. By doing so, you keep all the files you need handy and avoid searching in several places every time you need them.

The Difference Between Memory And Storage

People often confuse the terms memory and storage, especially when describing the amount they have of each. The term memory refers to the amount of RAM installed in the computer, whereas the term storage refers to the capacity of the computer's hard disk. To clarify this common mix-up, it helps to compare your computer to an office that contains a desk and a file cabinet.

Consider a desk-and-file-cabinet metaphor. Imagine what it would be like if every time you wanted to look at a document or folder you had to retrieve it from the file drawer. It would slow you down tremendously, not to mention drive you crazy. With adequate desk space - or memory in this case - you can lay out the documents in use and retrieve information from them immediately, often with just a glance.

Another important difference between memory and storage is that the information stored on a hard disk remains intact even when the computer is turned off. However, any data held in memory is lost when the computer is turned off. In our desk space metaphor, it's as though any files left on the desk at closing time will be thrown away.

Memory And Performance

Adding more memory to a computer system increases its performance. If there isn't enough room in memory for all the information the CPU needs, the computer has to set up what's known as a virtual memory file. In so doing, the CPU reserves space on the hard disk to simulate additional RAM. This process, referred to as "swapping", slows the system down. In an average computer, it takes the CPU approximately 200ns (nanoseconds) to access RAM compared to 12,000,000ns to access the hard drive. To put this into perspective, this is equivalent to what's normally a 3 1/2 minute task taking 4 1/2 months to complete!

Bits And Bytes

Computers speak in a "code" called machine language, which uses only two numerals: 0 and 1. Different combinations of 0s and 1s form what are called binary numbers. These binary numbers form instructions for the chips and microprocessors that drive computing devices - such as computers, printers, hard disk drives, and so on. You may have heard the terms "bit" and "byte." Both of these are units of information that are important to computing. The term bit is short for "binary digit." As the name suggests, a bit represents a single digit in a binary number; a bit is the smallest unit of information used in computing and can have a value of either 1 or a 0. A byte consists of 8 bits. Almost all specifications of a computer's capabilities are represented in bytes. For example, memory capacity, data-transfer rates, and data-storage capacity are all measured in bytes or multiples thereof (such as kilobytes, megabytes, or gigabytes).

The discussion of bits and bytes becomes very relevant when it comes to computing devices and components working together. It is important to understand specifically how bits and bytes form the basis of measuring memory component performance and interaction with other devices like the CPU.

CPU And Memory Requirements

A computer's CPU processes data in 8-bit chunks. Those chunks, as mentioned in the previous section, are commonly referred to as bytes. Because a byte is the fundamental unit of processing, the CPU's processing power is often described in terms of the maximum number of bytes it can process at any given time. For example, Pentium and PowerPC microprocessors currently are 64-bit CPUs, which means they can simultaneously process 64 bits, or 8 bytes, at a time.

Each transaction between the CPU and memory is called a bus cycle. The number of data bits a CPU can transfer during a single bus cycle affects a computer's performance and dictates what type of memory the computer requires. Most desktop computers today use 168-pin DIMMs, which support 64-bit data paths. Earlier 72-pin SIMMs supported 32-bit data paths, and were originally used with 32-bit CPUs. When 32-bit SIMMs were used with 64-bit processors, they had to be installed in pairs, with each pair of modules making up a memory bank. The CPU communicated with the bank of memory as one logical unit.

Mobile Memory Management

Memory management starts with system design. Different OSs use different principles for managing memory. As an application programmer, it is important to understand, and build on, the OS's underlying model. And the end-user's interaction with a machine will be influenced by its memory management strategy.

Table 2 provides a summary of the memory resources and user interaction paradigms of some well known tethered and mobile systems:

Table 2. Mobile Memory Resources *(Source: Symbian, Ltd)*

Desktop PC (Windows, Unix etc)	Conventional multi-tasking, variable number of tasks (processes), each with own address space. If one task runs out of memory then a memory -allocation operation will fail, and some recovery action is needed. Large memory, huge hard disk, virtual memory with disk-based swap file.

EPOC R5	PC-like multi-tasking. But smaller memory, and no hard disk. So programs run out of memory more frequently.
PalmOS	Only one application task simultaneously, possibly a few system tasks also. System tasks pre-allocate RAM, user task and database manager allocate RAM dynamically.
Quartz	Like EPOC R5 — but there is no way for the end-user to close applications, and the file system isn't visible to the end-user.
Basic mobile phone	Like an embedded system, each task gets a fixed RAM budget, and data is stored elsewhere in files with fixed budget.
Windows C E 2.0	Like EPOC R5.
Pocket PC	Like Quartz.

In each of these environments, it is necessary to have a strategy to manage memory. The strategy must have:

- minimal impact on the user: low-memory conditions either do not arise, or have obvious causes and consequences, or are easy to manage
- programmer-friendliness: easy to explain the high-level ideas, easy to work with at detail level

Memory Strategies

Depending on the targeted platform, developers have different scenarios to consider when considering memory use of their programs. It is up to the developer to understand these conditions and build strategies around them.

Desktop PC

In a desktop PC, Low-memory conditions rarely arise, because of practically infinite resources. A "close some programs" dialog is easy enough for users to understand and respond to. Programmers can get careless because allocations so rarely fail. Consequences for end-users are bad when things do get tight: try setting your swap file to 1MB and playing on your PC: it's not pretty.

Palm OS

On the Palm, only a single application can be active at one time, and applications are structured as databases with multiple entries. A failure in the foreground application is easy enough to deal with: it causes at most one entry to be lost. Failure in background is impossible, because applications save data and terminate when sent to background. The negative impact of this is that task switching is slow.

Basic Mobile Phone

On most of the basic mobile phone models, the RAM budget for each task is fixed. By fixing the memory allocated, an out-of-memory(OOM) should never occur. If it does, there is little in the way to recover from it. Efficient programming is used to avoid OOM. Basic mobile phones are normally closed to third-party software, since this would break RAM budgets..

Software Consumption of Memory

The fundamental object in all high-level programming languages is a variable. A variable represents a memory address in RAM that the program reserves for program use. Each variable type occupies an amount of space and is capable of storing a specified range of values. The size of the variable and what values it can store is language dependent. Table 3 illustrates some of these values.

Table 3. Program Variable Values and Sizes *(Source: S. Clemons)*

Type	C Size	C Values	C++ Size	C++ Values	Java Size	Java Values
boolean	N/A	N/A	N/A	N/A	1 bit	0 or 1
byte	N/A	N/A	N/A	N/A	8 bit	-128 to 127
char	8 bit	-127 - +127	8 bit	-127 - +127	16 bits	Unicode
short	16 bits	-32767 to 32767	N/A	N/A	16 bits	-32767 to 32767
int	16 bits	-32767 to 32767	16 bits	-32767 to 32767	32 bits	-2147483647 to 2147483647
long	32 bits	-2147483647 to 2147483647	N/A	N/A	64 bits	2.2*10^-308 to 1.8*10^308
float	32 bits	-2147483647 to 2147483647	32 bits	-2147483647 to 2147483647	32 bits	-2147483647 to 2147483647
double	64 bits	2.2*10^-308 to 1.8*10^308	64 bits	2.2*10^-308 to 1.8*10^308	64 bits	2.2*10^-308 to 1.8*10^308
long double	64 bits	2.2*10^-308 to 1.8*10^308	N/A	N/A	N/A	N/A
void	N/A	N/A	0	0	N/A	N/A

Considering that structures, classes, and components are made up of any number of these variables, or even other structures, classes, or components, it's easy to see how large a program's memory requirements can become.

Memory Management Within A Program

A program starts with an amount of memory allocated for use by it. The program will use this memory in two ways:

- through procedure calls
- through dynamic data types

When a procedure is called, information about the call is placed on the user stack. Space for local variables is also allocated on the user stack. This is organized as a stack which grows when a procedure is called and shrinks when a procedure exits.

When dynamic memory is required it is taken from the ``heap''. Operations on the heap include

- get memory from heap
- release memory to heap
- exchange memory for a different bit

The heap will consist of space, some of which is in use and some of which is free.

This is just one example of the general memory management problem, whether it is memory managed by a program, memory managed by the O/S (which is a program), disk space, etc. One method of keeping track is by a bitmap. More common is by linked list. There may be one linked list containing both used and free memory.

Methods For Retrieving Memory

When a request is made for a piece of memory, it can be taken from the free list in a number of ways. First fit, best fit, worst fit, returning memory, and the buddy system are all methods for retrieving memory.

FIRST FIT

This finds the first hole in the list big enough for the request. This is fast to do in an address sorted list, because it just traverses the list until it finds a large enough one.

BEST FIT

This finds the smallest hole in the list big enough for the request. Surprisingly, best fit is not always the best method, because it results in lots of tiny leftover pieces that are not large enough to be used. This is fast to do in a size sorted list.

WORST FIT

This finds the largest hole in the list. This is fast in a size sorted list, if the largest is first. This is not so good either.

RETURNING MEMORY

When a piece of memory is returned to the free list, it is not just added to the list - if there is already memory on this list either side of it then the pieces should be coalesced into one. This is easy to do in an address sorted list.

BUDDY SYSTEM

This uses a trick on the way numbers are stored - in binary. In this system, memory is only allocated in units that are powers of two. So if 3 bytes are requested you get 4, and if 129 bytes are requested you get 256. This does lead to wasted space (internal fragmentation).

A list of lists of free space is maintained. The first list is the 1 byte blocks, the second the 2 byte blocks, then the 4 byte blocks, etc. When a request is made, the size is rounded up and a search made of the appropriate list. If there is something there it is allocated. If not, a search is made of the next largest, and so on until a block is found that can be used. This search can be done quickly.

A block that is too large is split into two. Each part is known as the ``buddy" of the other. When it is split, it is taken off the free list for its size. One buddy is placed on the free list for the next size down, and the other is used, splitting it again if needed.

For example, suppose a request is made for a 3 byte piece of memory and the smallest free block is 32 bytes. It is split into two 16 byte buddies, one of which is placed on the 16 byte free list. The other is split into two 8 byte buddies, one of which is placed on the 8 byte list. The other is split into two 4 byte buddies, one of which is placed on the 4 byte free list, and the other - finally - is used.

When a piece of memory is released, it is placed back on the appropriate free list. Here now is the trick: two free blocks can only be combined if they are buddies, and buddies have addresses that differ only in 1 bit. Two 1 byte blocks are buddies if they differ in the last bit, two 2 byte blocks are buddies if they differ in the 2nd bit, and so on. So it is very quick to find out if two blocks can be combined.

The advantage of the buddy system is that granting and returning memory are both fast operations. The disadvantage is internal fragmentation.

COMPACTION

Eventually, memory will become ``externally fragmented''. That is, there will be lots of small pieces of memory on the free list(s) that are too small to be useful. When a request comes in for something larger than any of these it will fail even though there may be enough free space in total.

To avoid this happening, memory must be compacted every now and then. This means moving all blocks down to the low end of memory to leave space at the top as one large block. Everything that references the old memory blocks will of course now be pointing to the wrong place. Various code relocation techniques help here.

Simple O/S memory management

The above was general memory management for any situation. Now the tasks that the operating system in particular has to do will be discussed. It has to manage processes. When a process starts, it must allocate a chunk of memory for it to run in. When it terminates, it returns that chunk to free space. The following sections discuss some simple schemes.

NO ALLOCATION

Nothing is done by the O/S. A process has access to all memory and can do what it wants. This is only found on very early machines. In these machines, every program had to do its own I/O, etc, and had to reload the O/S on termination.

SINGLE USER

In this system, the O/S reserves a section for itself, and a process can access the rest. This is the scheme used by MSDOS. Due to a lack of memory protection, a misbehaved MSDOS program can in fact overwrite the O/S. Normally, when MSDOS loads a program it sets aside space for it using information in the header part of an EXE file. The program then asks for memory from MSDOS using the ALLOC_MEMORY system call and releases it back to MSDOS using FREE_ALLLOCATED(Source: ARC Group)_MEM.

MULTIPROGRAMMING

In a multitasking system, many processes will be attempting to use the CPU and memory. It is not possible to just run each of them to conclusion, because for an interactive program (say a shell) this may be for many hours. In addition, while a process is waiting on an input or output device it generally cannot do anything else and so will be idle.

One scheme is to load a number of processes into different parts of memory, and let them use the CPU according to some scheme. The processes could be loaded using a static scheme in which each one gets a fixed size piece of memory, and cannot get more.

With a dynamic memory scheme, processes can ask for more memory as they need it. This can then be given if available from the free list maintained by the O/S.

In Unix, processes have their memory requirements divided up into several parts: text (program code), stack and data areas. The data area is allowed to grow or shrink using the `brk` or `sbrk` system calls. Within a data area the process itself manages its area using the standard C library calls malloc, free and realloc. These calls themselves make use of brk and sbrk if they need to change the size of the data area. This is a two-level mechanism.

software Modularization

Definition of Software Modularization

The Mobile Internet brings with it a challenge to developers to provide content to a wide assortment of devices. Each of these devices has different requirements and constraints. Software modularization is the decomposition of a giving language into a collection of abstract modules that provide specific types of functionality. These modules may be combined with each other and with other modules to create a subsets and extensions of the language

The modularization of a language refers to the task of specifying well-defined sets of elements that can be combined and extended by developers to make it economically feasible to deliver content on a greater number and diversity of platforms.

Modularizing software provides a means for product designers to specify which elements are supported by a device using standard building blocks and standard methods for specifying which building blocks are used. These modules serve as "points of conformance" for the content community. The content community can now target the installed base that supports a certain collection of modules, rather than worry about the installed base that supports this or that permutation of the language. The use of standards is critical for modularized software to be successful on a large scale. It is not economically feasible for content developers to tailor content to each and every permutation of a languages elements. By specifying a standard, either software processes can autonomously tailor content to a device, or the device can automatically load the software required to process a module.

Definition of a Module

A module is a collection of objects that are logically related. Those objects may include constants, data types, variables, and program units (e.g., functions, procedures, etc.). Note that objects in a module need not be physically related. For example, it is quite possible to construct a module using several different source files. Likewise, it is quite possible to have several different modules in the same source file. However, the best modules are physically related as well as logically related; that is, all the objects associated with a module exist in a single source file (or directory if the source file would be too large) and nothing else is present.

Modules contain several different objects including constants, types, variables, and program units (routines). Modules shares many of the attributes with routines; this is not surprising since routines are the major component of a typical module. However, modules have some additional attributes of their own. The following sections describe the attributes of a well-written module.

Module Attributes

A module is a generic term that describes a set of program related objects (routines as well as data and type objects) that are somehow coupled. Modules should be defined by its actions rather than by source code syntax. Good modules share many of the same attributes as good routines as well as the ability to hide certain details from code outside the module.

Good modules exhibit strong cohesion. That is, a module should offer a (small) group of services that are logically related. For example, a "printer" module might provide all the services one would expect from a printer. The individual routines within the module would provide the individual services.

Good modules exhibit loose coupling. That is, there are only a few, well-defined (visible) interfaces between the module and the outside world. Most data is private, accessible only through accessor functions. Furthermore, the interface should be flexible.

Good modules exhibit information hiding. Code outside the module should only have access to the module through a small set of public routines. All data should be private to that module.

SOFTWARE EXTRUSION

Definition of Software Extrusion

Extrusion is a term usually associated with mechanical engineering. It involves the shaping of a material. Normally, a material is pushed through a two-dimensional opening, varying in form. Remember that thing you stuck Play-Doh in and pushed the lever down and it squeezed out in the shape that you stuck over where the stuff comes out? Same thing.

Webster's Dictionary defines extrusion as:

> **1**. forcing, pressing, or pushing out; or
> **2**. shaping (as metal or plastic) by forcing through a die

The first and only reference I've found referring to extrusion of software was in a 1992 software review by Jeff Alger on Component Workshop and OODLs. Jeff wrote:

> "Component Workshop has a feature called extrusion that, again on paper, does much better. Their claim is that a minimal app like a simple text editor should be about 150K. Now we're talking. Extrusion involves stripping unneeded stuff like the compiler and only spitting out the essential code needed to actually run the application."

Goals of Software Extrusion

The primary design goal for extrusion is to is minimize the static memory footprint and the runtime memory usage of an application. The static memory footprint is comprised of mostly ROM, but also some RAM. It is the memory which is used when application environment classes are preloaded. The runtime memory usage, as the term suggests, is the memory which is consumed at runtime by the dynamically loaded classes, stacks, and heap storage to support dynamic allocation. The end result should be a system with drastically reduced ROM and RAM requirements which has no perceivable degradation in execution speed.

Identifying Top Memory Consumers

During the running of an application, there are some categories that consume more memory than others. These top memory consumer are targeted for potential for memory usage reduction. These include:

- The memory footprint of classes and their various components
    - Preloaded (mostly ROM use)
    - Dynamically loaded
- Stacks
- Heap usage
- Code size

Software extrusion deals with both reducing the memory footprint of classes and other components, and reducing overall code size.

Definition of a Software Component

Before we discuss extrusion of software components any further, it is important to provide a definition of a component. There are many differing definitions of a software component. The software industry has yet to standardize one definition. For the purpose of this report, a software component will be defined as:

"A component is an identifiable piece of software that describes and/or delivers a set of meaningful services that are only used via well-defined interfaces."

Purpose of Software Components

Software components enable practical reuse of software "parts" over multiple applications. An example of a component would be a button in a Java application. Some of the attributes of the button may change from one application to another, such as the label or the command it executes, but the code that creates a button for one application can be reused to create a button for another application, without the programmer having to rewrite the code. There are other units of reuse, such as source code libraries, designs, or architectures. Therefore, to be more specific, software components are binary units of independent production, acquisition, and deployment that interact to form a functioning system.

Software components should always exhibit the following essential characteristics:

- **Identifiable –** Components should have a clear individual identity.
- **Traceable –** Components should retain their identity and be traceable once they have been consumed into another component or application, enabling their replacement

- **Replaceable –** Can be replaced by a new version or another component offering the same service or function with no impact on the consuming application. More accurately, it is the implementation of the component that is replaceable.
- **Accessed Only via Interfaces** – Components are only accessed via defined interfaces, with no dependencies on the physical implementation of the component. More specifically, Applications should only inherit the behavior offered through a specific interface, and not inherit the whole implementation.
- **Accurately Documented Service –** To enable reuse, services offered via an interface must be accurately documented in such as way that it is possible to understand not just how to communicate with them, but *what* service is provided. However, to enable reuse, it should not be necessary to document *how* this is provided.

There are a number of other characteristics that may be desirable, but are optional:

- **Physical Implementation is Hidden** – How the service delivered by a component is actually performed should be hidden from the consumer. The interface should only say *what* is offered, not *how*.
- **Independent** – Components should be independent of the implementation of other components. If not, they are sometimes termed *sub-components*.
- **Encapsulated** – Components are encapsulated and only expose the services they provide via their interfaces. The component forms a boundary around everything that comprises its physical implementation, including sub-components.
- **Can be Reused Dynamically** – Components can be dynamically assembled into applications. Rapid adaptation, application assembly, and end-user consumption is made easier if components can be called dynamically at runtime, rather than statically via a compile and link process.
- **Offer a Generic Service** – They provide a generic service that can be used in unpredictable combinations. Though opportunities for reuse are increased, they may need to be specialized before they can be used.
- **Specialized Only via Designated *Plug Points*** – Specialization of generic services should take place via a specified extension mechanism, rather than by copying and/or changing the physical implementation that reduces replaceability.

- **Certified and Authenticated** – Certification and authentication techniques enable applications to ensure that the correct service is being used. This is a necessity when applications are assembled dynamically and/or third-party services are subscribed to.

Basis For Software Extrusion: Software Architecture

Software extrusion comes from the principles of software architecture. The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.

"Externally visible" properties refer to those assumptions other components can make of a component, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. Software architecture must abstract away some information from the system (otherwise there is no point looking at the architecture, we are simply viewing the entire system) and yet provide enough information to be a basis for analysis, decision making, and hence risk reduction.

Software architecture is a set of concepts and design decisions about the structure and texture of software that must be made prior to concurrent engineering to enable effective satisfaction of architecturally significant explicit functional and quality requirements and implicit requirements of the product family, the problem, and the solution domains.

Implications of Software Architecture

Architecture Defines Components

The architecture embodies information about how the components interact with each other. This means that architecture specifically omits content information about components that does not pertain to their interaction.

Systems Can Be Comprised of More Than One Structure

Not only can systems be comprised of more than one structure, but no one structure holds the irrefutable claim to being *the* architecture. By intention, a software architecture does not specify what architectural components and relationships are. Is a software component an object? A process? A library? A database? A commercial product? It can be any of these things and more.

Every Software System Has An Architecture

Every system can be shown to be composed of components and relations among them.

The Behavior Of Each Component Is Part Of The Architecture

Each component's behavior can be observed or discerned from the point of view of another component. This behavior is what allows components to interact with each other, which is clearly part of the architecture. Hence, most of the box-and-line drawings that are passed off as architectures are in fact not architectures at all. They are simply box-and-line drawings.

Small Memory Software Architecture

The architecture for a system with limited memory must describe policies for memory management and ensure that each component's allocations are feasible in the context of the system as a whole. This means that each individual component must take explicit responsibility for managing its own memory. The design should incorporate small data structures that require minimum memory to store the information the system needs.

Data structures that are appropriate where memory is unrestricted may be far to prodigal where memory is limited. Techniques like compression and using secondary storage can reduce a program's main memory requirements, but both have significant liabilities when used to manage the data a program needs to work on. Many kinds of compression cannot be accessed randomly; if random access is required, the data must be uncompressed first, costing time, and requiring a large amount of buffer memory for the uncompressed data. Data stored on secondary storage is similarly inaccessible, and needs to be copied into main memory buffers before it can be accessed.

Data structure design states that a programmer should choose the smallest structure that supports the operation needed. For any given data set there are many different possible data structures that might support it. First, the program's requirements must be analyzed to determine the information the program needs to store. Second, consider the characteristics of the data – what's its total volume and how will it be accessed. And lastly, choose the data structure that best meets the program's needs.

Applying Extrusion to Software Components

Extrusion of software components takes data structure design one step further. Instead of just choosing a component that meets a need, any part of the component that is determined to be unnecessary for a particular purpose is removed. Using the Play-Doh metaphor again, imagine Play-Doh as a software component that performs the functions of a drop-down list. The drop-down list component has many attributes: size, variable name, text color, background color, mouse event handlers, and possibly many more. If you were developing an application for a Smart Phone, you probably wouldn't need many of the colors or mouse event handlers. Not only are these features unsuitable for a Smart Phone, but they also take up valuable memory while providing no service to the user. So on the opening of your extruder goes the shape of a Smart Phone. You put the drop-down list component in the extruder, push down, and out comes a scaled down version of the component, stripped of unnecessary attributes, and more correctly shaped for a particular role. It's still Play-Doh, but Play-Doh that uses less memory.

application of software EXTRUSION

Languages Applying Extrusion

Although the term “extrusion” may not be used in the software development mainstream, the philosophies that it implies are already being used by some to overcome hardware restrictions of the Mobile Internet. Some of the languages that have had software extrusion principles applied to create new language standards are XHTML, HTML, Java, and Oracle.

XHTML

Extensible Hyper Text Mark Up Language (XHTML 1.0) is the next version of HTML that bridges the gap between the Web's prior easy-going days of HTML (HyperText Markup Language) and the explosive future growth of XML (Extensible Markup Language).

XML is a set of rules allowing spreadsheets, address books, databases and other computer applications to communicate and be understood. Although it has some of the same tags as HTML, their meaning depends on the application XML communicates with. This combination of flexibility and strictness permits a wide range of devices and applications to exchange information with little ambiguity.

An important aspect to remember is that XML code can be embedded within the XHTML document with or without a DTD (document type definition). But the parser (browser) to be able to understand embedded XML requires a DTD. In short a DTD is a document (schema) which defines all the elements (tags) used within an XML document. Hence XML + HTML = XHTML.

How do HTML and XML fit together? HTML is a markup language described in SGML (Standard Generalized Markup Language). XML is a restricted form of SGML, removing many of SGML's more complex features, but preserving most of SGML's power and commonly used features. XHTML is the reformulation of HTML 4.0 as an application of XML. It is the W3C's new version of HTML. The W3C (World Wide Web Consortium) took the logical step of expressing the HTML 4.0 standard in XML instead of using the more complicated SGML.

XHTML provides three major advantages:

- **Transforms** HTML from a stand-alone language, into an XML version.
- **Modularizes** HTML
- **Extends** HTML elements to facilitate creation of robust interfaces.

XHTML Basic

XHTML Basic is a subset of the XHTML document type. It includes the minimal set of modules required to be an XHTML host language document type, and in addition it includes images, forms, basic tables, and object support. It is designed for Web clients that do not support the full set of XHTML features; for example, Web clients such as mobile phones, PDAs, pagers, and settop boxes. XHTML Basic supports basic text, hyperlinks, basic forms, basic tables, images, and some meta information. But since XHTML Basic is modular, you can again add these features to it should your device support them. So if your PDA has Java support, it might be designed to support XHTML Basic plus an XHTML Java module.

XHTML Basic is designed as a common base that may be extended. For example, an event module that is more generic than the traditional HTML 4 event system could be added or it could be extended by additional modules from XHTML Modularization such as the Scripting Module. The goal of XHTML Basic is to serve as a common language supported by various kinds of user agents.

XHTML for Small Information Appliances

HTML 4 is a powerful language for authoring Web content, but its design does not take into consideration issues pertinent to small devices, including the implementation cost (in power, memory, etc.) of the full feature set. Consumer devices with limited resources cannot generally afford to implement the full feature set of HTML 4. Requiring a full-fledged computer for a c-cess to the World Wide Web excludes a large portion of the population from consumer device access of online information and services.

Because there are many ways to subset HTML, there are many almost identical subsets defined by organizations and companies. Without a common base set of features, developing applications for a wide range of Web clients is difficult.

The motivation for XHTML Basic is to provide an XHTML document type that can be shared across communities (e.g. desktop, TV, and mobile phones), and that is rich enough to be used for simple content authoring. New community-wide document types can be defined by extending XHTML Basic in such a way that XHTML Basic documents are in the set of valid documents of the new document type. Thus, an XHTML Basic document can be presented on the maximum number of Web clients.

Background and Requirements

Information appliances are targeted for particular uses. They support the features they need for the functions they are designed to fulfill. The following are examples of different information appliances:

- Mobile phones
- Televisions
- PDAs
- Vending machines
- Pagers
- Car navigation systems
- Mobile game machines
- Digital book readers
- Smart watches

The common features found in these document types include:

- Basic text (including headings, paragraphs, and lists)
- Hyperlinks and links to related documents
- Basic forms
- Basic tables
- Images

- Meta information

This set of HTML features has been the starting point for the design of XHTML Basic. Since many content developers are familiar with these HTML features, they comprise a useful host language that may be combined with markup modules from other. For example, XHTML Basic may be extended with an event module that is more generic than the traditional HTML 4 event system or it could be extended by additional modules from XHTML Modularization, such as the Scripting Module.

It is not the intention of XHTML Basic to limit the functionality of future languages. But since the features in HTML 4 (frames, advanced tables, a fixed set of attribute event handlers, etc.) were developed for a desktop computer type of client, they have proved to be inappropriate for many non-desktop devices. XHTML Basic will be extended and built upon. Extending XHTML from a common and basic set of features, instead of almost identical subsets or the too-large set of functions in HTML 4, will be good for interoperability on the Web, as well as for scalability.

Compared to the rich functionality of HTML 4, XHTML Basic may look like one step back, but in fact, it is two steps forward for clients that do not need what is in HTML 4 and for content developers who get one XHTML subset instead of many.

Modularizing XHTML

XHTML has been divided into modules, so implementers can choose which group of tags their particular application will support while still maintaining compliance. For example, a PalmPilot browser can safely ignore frames and forms and still be considered a standards-complaint browser. Thus making content management much easier for different devices.

XHTML consists of 4 core modules. These core modules are modules that are required to be present in any XHTML Family Conforming Document Type.

- Structural Module
- Text Module
- Hypertext Module
- List Module

Other modules that comprise XHTML are:

- Applet Module
- Base Module
- Basic Forms Module
- Basic Tables Module
- Bi-Directional Text Module
- Client-Side image Map Module
- Edit Module
- Forms Module
- Hypertext Module
- Frame Module
- Image Module
- Intrinsic Events Module
- Legacy Module
- Link Module
- List Module
- Meta information Module
- Name Module
- Object Module
- Presentation Module
- Scripting Module
- Server-Side image Map Module
- Structure Module
- Style Attribute Module
- Style Sheet Module
- Tables Module
- Target Module
- Text Module

Extrusion of XHTML

XHTML goes beyond modularization by removing unneeded attributes from certain elements of XHTML Basic, based on device capabilities. If a attribute has been removed, there is often an existing attribute remaining that has similar capabilities. A comparison of the attributes for

XHTML Standard Tags vs. XHTML Basic Tags, and their assigned modules, is provided as APPENDIX A.

Style Sheets

XHTML Basic does not support the *style* element. Instead, the *link* element can be used to include external style sheets. The *div* and *span* elements and the *class* attribute are supported to hook style information onto the structure. Separation between structure and presentation allows user agents to download the style sheets if they support style sheets; user agents that do not support style sheets can ignore the external stylesheet. The *media* attribute can be used to select the appropriate style sheets.

Script and Events

The *script* and *noscript* elements are not supported by XHTML Basic. Usually small devices have limited memory and CPU power. Execution of script programs may not be supported. Contents should be readable even if scripts are not executed.

Event handler attributes used to invoke script programs are not supported. Events are device dependent. A generic event handling mechanism would be more appropriate than hardwiring the event names in the document type definition.

Presentation

Many simple Web clients cannot display fonts other than monospace. Bi-directional text, bold faced font, and other text extension elements are not supported.

Again, style sheets be used to create a presentation that is appropriate for the device.

Frames

Frames are not supported. Frames depend on a screen interface and may not be applicable to some small appliances like phones, pagers, and watches.

Extruding Even Deeper

XHTML extrudes even further, by removing unneeded attributes from certain elements of XHTML Basic. For example, all of the tags in the standard Text Module, with the exception of the <br> tag, contain the following attributes:

- class
- id
- title
- xml:lang
- onclick
- ondblclick
- onmousedown
- onmouseup
- onmouseover
- onmousemove
- onmouseout
- onkeypress
- onkeydown
- onkeyup
- style

Many hand held devices don't have a mouse, so all of the mouse events were removed. Style sheets were also deemed unnecessary, so the *style* attribute was also removed. What is left of the attributes for the elements in the Text Module for XHTML Basic are 4 attributes:

- class
- id
- title
- xml:lang

By scaling these elements from 15 elements down to only 4, XHTML Basic lowers it's memory usage when Web browsers are displaying pages.

Compact HTML

Compact HTML is similar to XHTML Basic. It is a well-defined subset of HTML 2.0, HTML 3.2, and HTML 4.0 recommendations, designed for small information appliances. Unlike XHTML, it does not integrate any of XML's capabilities.

Compact HTML, sometimes referred to as CHTML (but not to be confused with Compiled HTML), was developed by Tokyo's Access Co. and has been the primary language i-Mode has used. Recently, NTT DoCoMo, owner of i-mode, and some members of the WAP Forum, agreed to adopt XHTML Basic as the future of the Mobile Web development language. Although this may spell the end of Compact HTML, the extrusion techniques used in developing the language are still worth exploring.

Design Principles

Compact HTML was designed to meet the requirements of small information appliances by following four principles:

1. Completely based on the current HTML W3C recommendations. Since Compact HTML is defined as a subset of HTML 2.0, HTML 3.2 and HTML 4.0 specifications, it inherits the flexibility and portability from the standard HTML.
2. Lite Specification. Compact HTML has to be implemented with small memory and low power CPU. Frames and tables which require large memory are excluded from Compact HTML.
3. Can be viewed on a small mono-color display. Compact HTML assumes a small display space of black and white color. However, it does not assume a fixed display space, but it is flexible for the display screen size. Compact HTML also a ssumes single character font.
4. Can be easily operated by the users. Compact HTML is defined so that all the basic operations can be done by a combination of four buttons; *Cursor forward, Cursor backward, Select*, and *Back/Stop*(Return to the previous page). The functions which require two-dimensional focus pointing like "image map" and "table" are excluded from Compact HTML.

Extrusion of Compact HTML

Like XHTML Basic, Compact HTML removes many of HTML’s features, include:

- JPEG image
- Table
- Image map
- Multiple character fonts and styles
- Background color and image
- Frame
- Style sheet

CHTML also removes unnecessary attributes from some of the elements to lower the memory requirement. As an example, HTML 4’s anchor tag, *a*, contains the following attributes:

- charset
- type
- name
- href
- hreflang
- rel
- rev
- accesskey
- shape
- coords
- tabindex
- onfocus
- onblur

In Compact HTML, it is reduced to the following list of attributes:

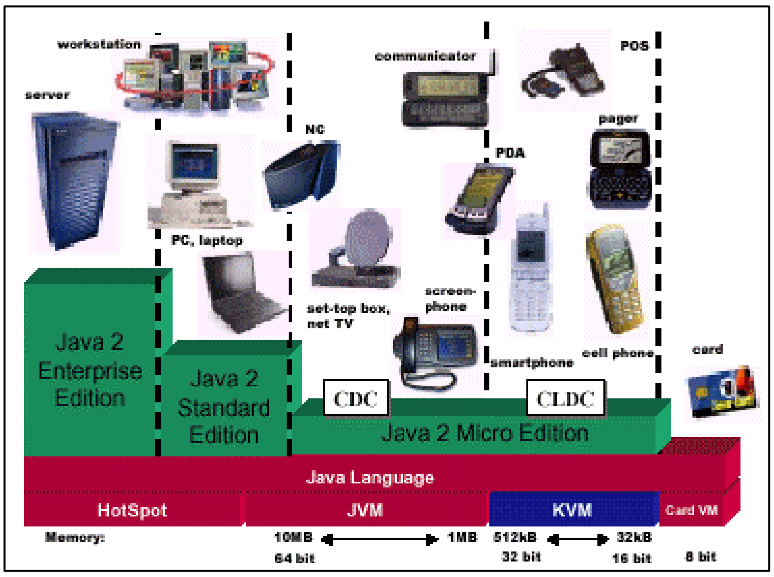
- name
- href

A list of Compact HTML elements and their attributes compared to standard HTML is included as APPENDIX B.

Java 2 Micro Edition (J2ME)

Overview

Depending on the targeted computing platform, Sun has grouped the Java technologies into three editions: Java 2 Micro Edition (J2ME), Java 2 Standard Edition (J2SE) and Java 2 Enterprise Edition (J2EE). Each of these editions has been customized specifically for the platform it is targeting, whether it be a consumer device, a desktop computer or an enterprise network



server (see Figure 3:).

Figure 3: Java 2 Platform Targets *(Source: Sun)*

The Java 2 Micro Edition consists of the technology, APIs, tools and standards needed to create applications for consumer devices. J2ME specifically targets the consumer space, which covers the range of small commodities such as smart cards and pagers all the way up to the TV set-top boxes. J2ME provides a complete solution for creating dynamically extensible, networked products and applications for the consumer and embedded appliances.

At a high level, J2ME is currently targeted at two categories of products:

- Shared, fixed, connected information devices. Typical examples of devices in this category include TV set-top boxes, Internet TVs, Internet-enabled screenphones, high-end communicators,
- Personal, mobile, connected information devices. Cell phones, pagers and personal organizers are examples of devices in this category. These devices have a large range

of user interface capabilities, memory availability, processing power and persistent, high-bandwidth network connections.

In practice, the line between these two categories is defined more by the memory budget, bandwidth considerations, battery power consumption, and physical screen size of the device, rather than by its specific functionality or type of connectivity of these devices.

J2ME Architecture

J2ME is a technology defined by many parts and specifications. These many parts and specifications help J2ME address the diverse needs of a wide spectrum of consumer products. J2ME provides a range of virtual machine technologies optimized for the different processor types and memory footprints commonly found in the consumer and embedded devices.

For resource-constrained devices, J2ME supports minimal configurations of the Java virtual machine and Java APIs that conisists of just the essential capabilities of each kind of device. These configurations can be extended with new APIs and VMs as new features are added to these devices and new applications are developed for them by the application vendors.

This flexibility and extensibility are supported by J2ME using the following building blocks:

- Java Virtual Machine Layer. This layer is an implementation of a Java Virtual machine that is customized for a particular device's host operating system.
- Configuration Layer. A J2ME configuration defines a minimum platform for a "horizontal" category of devices which has similar memory requirements and processing power. A configuration defines the Java language and virtual machine features and minimum class libraries that a device manufacturer can expect to be available on all devices of the same category.
- Profile Layer A. J2ME device profile is layered on top of a configuration. A profile defines the specific requirements of a certain "vertical" category of devices. The main goal of a profile is to define a standard Java platform for a certain vertical device family and guarantee interoperability within them. Profiles typically include class libraries that are more domain-specific than the class libraries provided in a configuration.

The J2ME architecture currently has two configurations. The Connected Device Configuration technology (CDC) uses the CVM, a full-featured VM that is similar to a virtual machine residing on a desktop system. This configuration is intended for devices with at least a few megabytes of available memory.

For wireless devices and other systems with severely constrained memory environments, J2ME uses the Connected Limited Device Configuration technology (CLDC).

Mobile Information Device Profile (MIDP) is one of the profiles defined in the J2ME architecture which is a set of Java APIs which, together with CLDC provides a complete J2ME application run-time environment targeted at mobile information devices, such as cellular phones and two-way pagers. The MIDP profile address issues such as user interface, persistence storage, networking and application model.

Extruding the Java Virtual Machine: Introduction to KVM

A Java virtual machine is the foundation for Java technology, allowing applications written in the Java programming language to be portable across different hardware environments and operating systems. The virtual machine mediates between the application and the underlying platform, converting the application's bytecodes into machine-level code appropriate for the hardware and operating system being used. In addition to governing the execution of an application's bytecodes, the virtual machine handles related tasks such as managing the system's memory, providing security against malicious code, and managing multiple threads of program execution.

The core of J2ME is it's virtual machine, called the KVM. In order to meet the market need for a very small footprint Java implementation, the KVM was designed to overcome three key technical challenges: reducing the size of the virtual machine and class libraries themselves, reducing the memory utilized by the virtual machine during execution, and allowing for components of the virtual machine to be configured to suit particular devices (for example, by allowing pluggable garbage collection).

KVM is a compact, portable Java virtual machine specifically designed from ground up for small, resource-constrained devices. The high-level design goal for KVM was to create the smallest possible "complete" Java virtual machine that would maintain all the central aspects of the Java programming language, but would run in a resource-constrained device with only a few hundred kilobytes total memory budget.

KVM initially was an acronym for “Kilobyte Virtual Machine”, because of it’s small footprint (the Java 2 Standard Edition Java Virtual Machine has a 32 MB footprint). However, this specification allows for the possibility of running on other virtual machines. This typically applies to digital cellular phones, pagers, personal organizers, and small retail payment terminals.

The actual role of KVM in the target devices can vary significantly. In some implementations, KVM is used on top of an existing software stack to give the device the ability to download and run dynamic, interactive, secure Java content on the device. In other implementations, KVM is used at a lower level to implement the system software and applications of the device in the Java programming language. Several alternative usage models are possible.

Extruding KVM

So how does Sun shrink the memory foot print from it’s Standard Edition virtual machine down to a size realistic for mobile devices? The KVM is designed to support standardized, incremental deployment of Java virtual machine features and Java APIs called for by the Java 2 ME architecture. The KVM specification identifies several optional features which can be omitted from some minimum initial device configurations, depending on the features and capabilities of the device. The J2ME architecture includes a small number of configurations that are specified and standardized by Sun. The KVM specification gives the guidelines that Sun uses in defining configurations. It spells out exactly which features can be omitted from an initial configuration.

The following features defined in The Java Virtual Machine Specification are optional in the KVM architecture. In each case, a feature is designated "optional" because:

1. The applications designed for a particular configuration (or class of device) do not need the feature.
2. Elimination of the feature significantly reduces memory footprint or enables some other cost savings or necessary functionality in the class of device targeted by that configuration.

While these features are optional at the KVM architectural level, they may be required at the KVM implementation level. Each Java 2 ME configuration specifies whether or not each optional feature is included. Any KVM implementation claiming to support a particular configuration must implement all the features required by that configuration.

Features not explicitly identified in the following list are required and must be implemented in all configurations.

- **Large data types: long, float, and double** – many configurations do not need the extended range and precision of the larger data types.
- **Multi-dimension arrays** -- many configurations do not need to support arrays of more that one dimension.
- **Class file verification** -- some specified configurations may not need to support on-device verification of class files. Instead technology is planned to be developed to enable class files to be efficiently verified "off-line" and delivered to the device.
- **Handling of Error classes** -- when the Java virtual machine encounters a serious internal problem, it throws an instance of a subclass of java.lang.Error. However, because there is often no reasonable form of programmatic recovery from these errors, a configuration may specify the KVM to halt with a configuration-defined error indication. Or the configuration may allow device vendors to define device-specific behavior and recovery actions when it encounters such conditions.
- **Threads and event handling** -- some configurations may require a different application execution model from the standard Java technology-based model using the Thread class and standard event handling.
- **Java Native Interface (JNI)** -- many configurations might not need the flexibility of the JNI for the way in which native methods are linked and invoked. A configuration may use a defined alternative, simpler mechanism to invoke native methods.
- **Class loaders** -- many configurations might not need the full flexibility of Java class loaders. A configuration must specify the mechanisms by which classes are located and loaded into the KVM.
- **Finalization** -- many configurations do not need to support object finalization.
- **Maximum size limitations** -- many configurations do not need to support the full range of sizes of internal virtual machine data structures. A configuration may specify a "maximum supported" range for some or all of the following values:

    - The number of classes in a package
    - The number of interfaces implemented by a class
    - The number of fields in a class
    - The number of methods in a class
    - The number of elements in an array
    - The number of bytes of code per method
    - The length of a string in a CONSTANT_UTF8 constant pool entry
    - The maximum amount of stack that a method may use
    - The maximum number of locals that a method may use
- **Start-up --** each configuration must specify how the KVM locates the initial class and method to execute and the expected attributes of that class and method.

The Java 2 Micro Edition specifies a core set of APIs. These are required by all configurations and, therefore, must be supported by all K virtual machines. In addition to this minimum subset, every KVM must also support whatever APIs are specified by its configuration.

The following list contains only the class names. Many of these classes have been significantly subsetted in order to reduce the minimum required functionality to key fields and methods.

Basic Classes from java.lang

`Object, Runtime, System`

These classes are included in the core API because they are fundamental to the operation of the virtual machine. However, because threading is an optional feature, the minimal core Object class does not require the various `wait` and `notify` methods. In addition, many methods of Runtime and System are not required.

Throwable Classes from java.lang

`Throwable, Exception, RuntimeException` and all its subclasses.

These classes are included in the core API because they are also fundamental to the operation of the virtual machine. More complex methods such as `printStackTrace` are not required.

Data Type Classes from java.lang

```
Boolean, Byte, Character, Integer, Short, Void
```

These classes are included in the core API because they are fundamental and generally useful to most programmers writing in the Java language. These classes are subsetted to only the most necessary methods and fields.

String Classes from java.lang

```
String, StringBuffer
```

These classes are included in the core API because they are fundamental and generally useful to most programmers writing in the Java programming language. These classes are subsetted to only the most necessary methods and fields.

Miscellaneous Classes from java.lang

```
Math
```

This class is included in the core API because a few of its methods are generally useful to most programmers writing in the Java programming language.

Miscellaneous Classes from java.util

```
BitSet, Dictionary, Enumeration, Hashtable, Vector
```

These classes are included in the core API because they are generally useful to most programmers writing in the Java programming language.

Table 4 illustrates the J2ME's extrusion of J2SE's java.lang basic classes.

Table 4. Extrusion of Java Classes *(Source: S. Clemons)*

API	Java 2, Standard Edition	Java 2, Micro Edition
Basic classes from java.lang	`Boolean` `Character` `Class` `ClassLoader` `Compiler` `Double` `Float` `Integer` `Long` `Math` `Number` `Object` `Process` `Runtime` `SecurityManager` `String` `StringBuffer` `System` `Thread` `ThreadGroup`	`Object` `Runtime` `System`
Throwable classes from java.lang	`ArithmeticException` `ArrayIndexOutOfBoundsException` `ArrayStoreException` `ClassCastException` `ClassNotFoundException` `CloneNotSupportedException` `Exception` `IllegalAccessException` `IllegalArgumentException` `IllegalMonitorStateException` `IllegalStateException` `IllegalThreadStateException` `IndexOutOfBoundsException` `InstantiationException` `InterruptedException` `NegativeArraySizeException` `NoSuchFieldException` `NoSuchMethodException` `NullPointerException` `NumberFormatException` `RuntimeException` `SecurityException` `StringIndexOutOfBoundsException`	`Throwable` `Exception` `RuntimeException`

In addition to removing many of the classes from the API, J2ME also significantly trims each class itself. Below is the class declaration for an `Object` in J2ME:

```
public class Object {
    public final native Class getClass();
    public native int hashCode();
    public boolean equals(Object obj) {
        return (this == obj);
    }
    public String toString() {
        return     getClass().getName()     +     "@"     +     Inte-
ger.toHexString(hashCode());
    }

    public final native void notify();
    public final native void notifyAll();
    public final native void wait(long timeout) throws InterruptedExcep-
tion;
    public final void wait(long timeout, int nanos) throws InterruptedEx-
ception {
        if (timeout < 0) {
            throw  new  IllegalArgumentException("timeout  value  is  nega-
tive");
        }

        if (nanos < 0 || nanos > 999999) {
            throw new IllegalArgumentException(
                                "nanosecond timeout value out of range");
        }

        if (nanos >= 500000 || (nanos != 0 && timeout == 0)) {
            timeout++;
        }
        wait(timeout);
    }
    public final void wait() throws InterruptedException {
        wait(0);
    }
}
```

In contrast, the Object Class declaration in Java's Standard addition is constructed as such:

```
public class Object {
    private static native void registerNatives();
    static {
        registerNatives();
    }

    public final native Class getClass();
    public native int hashCode();
    public boolean equals(Object obj) {
        return (this == obj);
    }

    protected native Object clone() throws CloneNotSupportedException;

    public String toString() {
        return      getClass().getName()      +      "@"      +      Inte-
ger.toHexString(hashCode());
    }

    public final native void notify();
    public final native void notifyAll();
    public final native void wait(long timeout) throws InterruptedExcep-
tion;

    public final void wait(long timeout, int nanos) throws InterruptedEx-
ception {
        if (timeout < 0) {
            throw  new  IllegalArgumentException("timeout  value  is  nega-
tive");
        }
        if (nanos < 0 || nanos > 999999) {
            throw new IllegalArgumentException(
                 "nanosecond timeout value out of range");
        }
        if (nanos >= 500000 || (nanos != 0 && timeout == 0)) {
             timeout++;
        }
        wait(timeout);
    }

    public final void wait() throws InterruptedException {
        wait(0);
    }
    protected void finalize() throws Throwable { }
}
```

Missing from the J2ME class are the `registerNatives`(), `clone`(), and `finalize`() methods. This doesn't seem like much, but looking at each method further, the memory savings becomes noticeably apparent. The `registerNatives`() method includes:

```
typedef struct {

    char *name;

    char *signature;

    void *fnPtr;

} JNINativeMethod;

 jint RegisterNatives(jclass clazz, const JNINativeMethod *methods, jint
nMethods) {
        return functions->RegisterNatives(this,clazz,methods,nMethods);

    jint (JNICALL *RegisterNatives)

    (JNIEnv *env, jclass clazz, const JNINativeMethod *methods, jint
nMethods);

}
```

Table 5 illustrates the removal of two methods (`clone`() and `finalize`()) from the `Object` class definition.

Table 5. *Object* Class Methods Comparison *(Source: S. Clemons)*

Java 2, Standard Edition	Java 2, Micro Edition
`clone()`	----
`equals(Object obj)`	`equals(Object obj)`
`FINALIZE()`	----
`getClass()`	`getClass()`
`hashCode()`	`hashCode()`
`notify()`	`notify()`
`notifyAll()`	`notifyAll()`
`toString()`	`TOSTRING()`
`wait()`	`wait()`
`wait(long timeout)`	`wait(long timeout)`
`wait(long timeout, int nanos)`	`wait(long timeout, int nanos)`

Configurations

Connected Limited Device Configuration (CLDC)

The configuration for mobile devices or the Connected Limited Device Configuration technology (CLDC technology) defines the Java platforms targeted for small and resource-constrained devices with small memory budgets and low processing power. The CLDC configuration is composed of the KVM and core class libraries that can be used on a variety of devices such as cell phones, two-way pagers, personal organizers, home appliances, and so on.

As mentioned earlier, J2ME can have many different virtual machines. CLDC defines a set of Java virtual machines that can run on the categories of targeted devices and support the profiles layered on top of CLDC. The KVM is just one particular implementation of a Java virtual machine meeting the CLDC specifications.

The primary goals of CLDC is to define a standard Java platform for small, resource constrained, connected devices and to enable third party application developers to easily create applications and content that can be deployed to those devices.

Mobile Information Device Profile (MIDP)

The Mobile Information Device Profile (MIDP) is a set of Java APIs which, together with the Connected Limited Device Configuration (CLDC), provides a complete J2ME application runtime environment targeted at mobile information devices, such as cellular phones and two-way pagers. The MIDP defines the application architecture for these devices and addresses issues such as user interface, persistence storage and networking.

The MID Profile runtime environment allows to dynamically deploy new applications and services on the end user devices. It is designed to work on top of CLDC and the software and hardware requirements of Mobile Information Devices are in addition to those for the broader range of Connected Limited Devices. The APIs defined by MIDP allows an open application

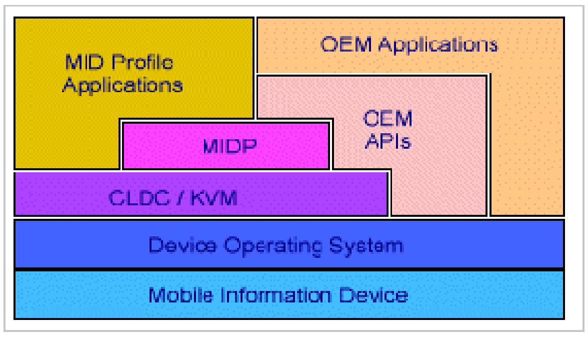


Figure 5.1 MIDP Architecture

development for Mobile Information Devices.

The primary goals of MIDP are to keep the implementation size minimal – must fit in small "footprint", and efficiency – must run on low-end microprocessors with limited heap size and with minimal creation of garbage.

MIDP Applications

The applications written for mobile information devices such as cellular phones and pagers are called MIDlets. Like applets, MIDlets are controlled by the software that runs them. In the case of an applet, the underlying software is a browser or the appletviewer tool and in the case of a MIDlet, the underlying software is the cell phone or two-way pager device implementation that supports the CLDC and MIDP. A MIDlet is a well behaved MIDP application which lives within the resouce constraints which runs and terminates when requested.

All the devices which support MIDP are supposed to have a device-specific Application Management Software which takes care of installing, managing and removing MIDlets interactively.

MIDlets move through a well defined lifecycle consisting of five phases. It is the task of the Application Management Software to move MIDlets through these phases:

- Retrieval - The AMS retrieves the MIDlet from some source and reads the MIDlet into the device's memory. The medium through which the MIDlet is downloaded depends on the device. It could be through a serial cable, an IRDA port, or a wireless network.
- Installation - Once the MIDlet is downloaded, the AMS installs the the MIDlet on the device. During the installation process, the MIDP implementation verifies that the MIDlet does not violate the device's security policies.
- Launching - A MIDlet is launched when a user selects it using the interface provided in the device. At this point, the MIDlet enters the KVM and the lifecycle methods of the MIDlet are invoked.
- Version Management -The AMS keeps track of all the MIDlets that are installed on the device including their version numbers. This information is used to upgrade a MIDlet to its new version.
- Removal -The AMS removes a MIDlet and cleans up the related resources from the memory.

A MIDlet can be in one of the three states after it is launched by the Application Management Software:

- Paused - A MIDlet enters the Paused state once it is created and initialized by the AMS. It can also enter this state when it is Active.
- Active - This state means the MIDlet is running normally. A MIDlet goes to the Active state from the paused state if there are no runtime exceptions during its initialization.
- Destroyed - This state means the MIDlet has released all its resources and is terminated. A MIDlet can reach this state either from the paused state due to a runtime exception during its initialization or from the active state when the user has chosen to close the application.

Memory Savings

By using extrusion techniques, Sun was able to reduce the memory of the application environment from 32 megabytes, down to 128 kilobytes, a very significant reduction. The 128 kilobytes of total memory budget required includes the virtual machine, minimal Java libraries and some heap space for running Java applications. A more typical implementation requires a total memory budget of 256 kilobytes, of which half is used as heap space for applications, 40 to 80 kilobytes is needed for the virtual machine itself, and the rest is reserved for class libraries. The ratio between volatile memory (e.g., DRAM) and non-volatile memory (e.g., ROM or Flash) in the total memory budget varies considerably depending on the implementation. A simple KVM implementation without system class prelinking support needs more volatile memory than a KVM implementation with system classes preloaded into the device.

Oracle Lite

Oracle8*i* was introduced in 1999 as a suite of databases designed for power and ease in Internet development and deployment. One of the versions included in this suite is called Oracle Lite. Oracle8i Lite is an embedded client-side database engine with the ability to sychronize both applications and data with an Oracle Enterprise server across the Web or even with a wireless connection. Optimized for use on small, handheld devices such as PDAs and laptops, Oracle8i Lite allows mobile users to have both a fully functional relational database engine on their machine for offline work and the necessary tools to synhcronize the applications and data with their host server.

Oracle Lite is for developing applications running on distributed or mobile PCs. It provides the strengths and advantages of larger Oracle databases at a fraction of the memory overhead. It is designed explicitly for software developers. Oracle Lite resides on the Oracle database and is automatically downloaded from the server to the client when needed. Networked users access needed applications directly from the server, while users working in the field download Oracle Lite, which then acts as a lightweight proxy for Oracle. The interface remains the same and the user detects no difference between the two scenarios.

Oracle is built upon Open Database Connectivity (ODBC). ODBC is a widely accepted application programming interface (API) for database access. All versions of Oracle use Structured Query Language (SQL) as its database access language.

Oracle Lite SQL

Oracle Lite uses a subset of Oracle's Structured Query Language (SQL). Table 6 illustrates the differences between the types of objects allowed in each version. In addition to limiting the object types, the name identifier for each object has been reduced from a maximum of 128 characters for Oracle, to only 31 characters for Oracle Lite.

Table 6. Oracle vs. Oracle Lite SQL Objects *(Source: Oracle)*

Oracle	Oracle Lite
Clusters	---
Database links	---
Database triggers	---
Dimensions	---
Indexes	Indexes
Java classes, Java resources, Java sources	Java classes, Java resources, Java sources
Packages	---
Profiles	---
Sequences	Sequences
Snapshots	Snapshots
Snapshot logs	---

Stored functions, stored procedures	---
Synonyms	---
Tables	Tables
Table spaces	---
Views	Views

Oracle Lite also has includes a subset of the standard Oracle SQL data types, operators, functions, and commands.

Oracle Lite Memory Savings

Through extrusion and modularity, Oracle Lite in its lightest configuration, with simply the kernel deployed on EPOC or Palm OS, consumes only 50 kilobytes of memory, quite a savings over the 1 megabyte footprint of the standard version. In its full configuration, with Java services, Oracle Lite never exceeds 750 KB.

Tools to Aid Extrusion

CC/PP

CC/PP stands for *Composite Capabilities/Preferences Profiles*, and is a way to specify what exactly a user agent (web browser) is capable of doing. This allows for sophisticated content negotiation techniques between web servers and clients, to produce optimized XML-based markup for display and use on a wide variety of web user agents.

As the number and variety of devices connected to the Internet grows, there is a corresponding increase in the need to deliver content that is tailored to the capabilities of different devices. Some limited techniques, such as HTTP '`accept`' headers and HTML `<alt>` tags, already exist. As part of a framework for content adaptation and contextualization, a general purpose profile format is required that can describe the capabilities of a user agent and preferences of its user. CC/PP is designed to be such a format.

CC/PP is designed to work with a wide variety of web-enabled devices, from PDAs to desktop machines to laptops to WAP phones to phone browsers to web television units to specialized browsers for users with disabilities. Proxies may also be used to provide markup transformation, transmission, or caching services for CC/PP-enabled clients and servers.

RDF

CC/PP is based on RDF, the Resource Description Framework, which was designed by the W3C as a general purpose metadata description language. RDF provides the framework with the basic tools for both vocabulary extensibility, via XML namespaces, and interoperability. There is a specification that describes how to encode RDF using XML, and another that defines an RDF schema description language using RDF. RDF was designed to describe the metadata or machine understandable properties of the Web. RDF is a natural choice for the CC/PP framework since user agent profiles are metadata intended primarily for communication between user agents and resource data providers. The metadata provided by CC/PP contains information about users and devices, instead of documents. This metadata includes:

- A framework in RDF for client capabilities information, such as screen size, user interaction design, software platform, etc.
- Capabilities can be referenced in profiles as URL:s
- The framework becomes extensible when you use RDF

Basic RDF Model

The foundation of RDF is a model for representing named properties and property values. The RDF model draws on principles from various data representation communities. RDF properties may be thought of as attributes of resources and in this sense correspond to traditional attribute-value pairs. RDF properties also represent relationships between resources and an RDF model can therefore resemble an entity-relationship diagram. (More precisely, RDF Schemas which are themselves instances of RDF data models are ER diagrams.) In object-oriented design terminology, resources correspond to objects and properties correspond to instance variables.

The RDF data model is a syntax-neutral way of representing RDF expressions. The data model representation is used to evaluate equivalence in meaning. Two RDF expressions are equivalent if and only if their data model representations are the same. This definition of equivalence permits some syntactic variation in expression without altering the meaning.

The basic data model consists of three object types:

- Resources - All things being described by RDF expressions are called resources. A resource may be an entire Web page. A resource may be a part of a Web page; e.g. a specific HTML or XML element within the document source. A resource may also be a whole collection of pages; e.g. an entire Web site. A resource may also be an object that is not directly accessible via the Web; e.g. a printed book. Resources are always named by URIs plus optional anchor id:s. Anything can have a URI; the extensibility of URIs allows the introduction of identifiers for any entity imaginable.
- Properties - A property is a specific aspect, characteristic, attribute, or relation used to describe a resource. Each property has a specific meaning, defines its permitted values, the types of resources it can describe, and its relationship with other properties.
- Statements - A specific resource together with a named property plus the value of that property for that resource is an RDF statement. These three individual parts of a statement are called, respectively, the subject, the predicate, and the object. The object of a statement (i.e., the property value) can be another resource or it can be a literal; i.e., a resource (specified by a URI) or a simple string or other primitive datatype defined by XML. In RDF terms, a literal may have content that is XML markup but is not further evaluated by the RDF processor.

CC/PP Profile

A CC/PP profile is a description of device capabilities (physical and programmatic) in terms of a number of "CC/PP attributes" for each component, as well as the user's specified preferences within the user agent's set of options, and specific qualities about the user that can affect content processing and display, such as physical location.. The values are then used by a server to determine the most appropriate form of a resource to deliver to a client. It is structured to allow a client and/or proxy to describe their capabilities by reference to a standard profile, accessible to an origin server or other sender of resource data, and a smaller set of features that are in addition to or different than the standard profile. A set of CC/PP attribute names, permissible values and associated meanings constitute a CC/PP vocabulary.

The description of each component is a sub-tree whose branches are the capabilities or preferences associated with that component. Though RDF makes modeling a wide range of data structures possible, including arbitrary graphs, complex data models are usually best avoided for profile attribute values. A capability can often be described using a small number of

CC/PP attributes, each having a simple, atomic value. Where more complex values are needed, these can be constructed as RDF subgraphs. One useful useful case for complex attribute values is to represent alternative values; e.g. a browser may support multiple versions of HTML.

The two primary ways in which a profile might be used are **selection** and **transformation**.

Selection is the process by which the originating server chooses an appropriate representation of requested web content from a finite set of existing representations. For example, the site might have three versions of a given page: a "rich XHTML with Java and ECMAscript" version for visual browsers, a "textual XHTML" version for non-visual browsers and older browsers, and a WML version for WAP phones. From the capabilities and preferences described in the CC/PP profile, the server would select the best match and send that back to the user agent.

Transformation, on the other hand, assumes that there is no finite set of representations, but rather than content is created on the fly, based on the properties expressed by the user agent profile. The content would be stored in an XML-compatible format and then transformed into an appropriate language (or modules thereof) that could be understood and optimized for the user agent, such as XHTML or WML.

CC/PP expresses the user agent capabilities and how the user wants to use them. XHTML document profiles express the required functionalities for what the author perceives as optimal rendering, and how the author wants them to be used.

The requirements emphasize three aspects: Flexibility, extensibility, and distribution. The framework must be flexible, since it is unreasonable to predict all the different types of devices that will be used in the future, or the ways those devices will be used. It must be extensible for the same reasons: It should not be hard to add and test new descriptions. And it must be distributed, since relying on a central registry might make it inflexible.

CC/PP Vocabulary

A CC/PP profile describes client capabilities in terms of a number of "CC/PP attributes", or "features". Each of these features is identified by a name in the form of a URI. A collection of such names used to describe a client is called a "vocabulary".

CC/PP defines a small, core set of features that are applicable to wide range of user agents, and which provide a broad indication of a clients capabilities. This is called the "core vocabulary". It is expected that any CC/PP processor will recognize all names in the core vocabulary, together with an arbitrary number of additional names drawn from one or more "extension vocabularies".

When using names from the core vocabulary or an extension vocabulary, it is important that all system components (clients, servers, proxies, etc.) that generate or interpret the names all apply a common meaning to the same name. It is preferable that different components use the same name to refer to the same feature, even when they are part of different applications, as this improves the chances of effective interworking across applications that use capability information.

Within an RDF expression describing a device, a vocabulary name appears as the label on a graph edge linking a resource to a value for the named attribute. The attribute value may be a simple string value, or another resource, with its own attributes representing the component parts of a composite value.

application of EXTRUSION to tcl

What Is TCL?

Tcl is an acronym for Tool Command Language. It is an interpreted language with programming features, available across platforms running Unix, Windows and the Apple Macintosh operating system. Interpreted languages, also called scripting languages, are compiled at runtime as opposed to being precompiled.

There are many other scripting languages besides Tcl, including JavaScript, Visual Basic, Perl, and others. As a group, all of the scripting languages tend to be used for integration applications, and all offer significant benefits over system programming languages.

Each scripting language has particular strengths. For example, JavaScript is known for its smooth integration with Web browsers, Visual Basic for its easy-to-learn development environment, and Perl for its string-handling capabilities. Tcl's greatest strength is its versatility: it can be embedded in applications or used standalone, it has outstanding GUI capabilities, and it can easily be connected to nearly any other application or protocol. Tcl was designed from the start to be used for many different purposes in many different situations, and the tremendous diversity of Tcl applications demonstrates that it has met this design goal.

In contrast, most other scripting languages were designed for a narrower set of tasks. They perform well for those specific tasks but they aren't used for as many different things as Tcl. For example, JavaScript is the obvious choice to use for simple scripting in a browser, but it is rarely used for anything outside the browser. Visual Basic provides excellent facilities for creating Windows GUIs, but it isn't suitable for integrating Windows desktops with Unix servers. Perl's string handling makes it an excellent choice for system administration tasks, report generation, and Web scripting, but it doesn't have native GUI capabilities and it isn't as easily embeddable as Tcl.

The Tk Widget

Tk, the associated toolkit, is an easy and efficient way of developing window based applications. Tk is an X11 (X Window System) toolkit based on Tcl, providing a "graphical scripting language." It is a C language extension of Tcl, and it can be used for easily creating Motif-like graphical user interfaces under the X Window System, often without writing C code. In contrast, writing in C is required to use the X Window System through other interfaces.

Tk provides a set of Tcl commands that create and manipulate *widgets*. A widget is a window in a graphical user interface that has a particular appearance and behavior. Widget types include buttons, scrollbars, menus, and text windows.

Widgets have classes and instances, but are not fully object oriented. It is not possible to subclass a widget class and use inheritance. Instead, widgets are very flexible and can be configured in many different ways to tune their appearance. The resource database can store configuration information that is shared by many widgets, and new classes can be introduced to group resources. Widget behavior is shared by using binding tags that group bindings. Instead of building class hierarchies, Tk uses composition to assemble widgets with shared behavior and attributes.

Application tasks are split into modules and any new application specific task is written and compiled as C or C++ program and exported as a new Tcl command. Then a Tcl script, consisting of a series of existing and new Tcl commands, is composed to make the overall application. Therefore, several Tcl based applications could be made to work together to create or extend into a new application.

Tk provides a higher level application programming interface for developing interactive widgets-based applications, particularly for those who wish to concentrate on the functionality of their application and have no need to gain indepth programming expertise in the underlying window system and/or verbose toolkits such as OSF/Motif.

Tcl and the Web

It is interesting to reflect on the fact that Tcl and the Web were invented at around the same time, circa 1988. Some of the early tools were built using Tcl/Tk, such as tkWWW, which was one of the first graphical browsers also to incorporate editing features. A “feature” of the Web is that the main delay in delivering a document across the Internet is the connection setup time and bandwidth constraints, so the programs used to generate dynamic Web pages are not performance-critical. Thus, Tcl and Perl quickly came to be used as languages for scripting CGI applications, since programming ease and convenience far outweigh any performance advantage.

Tcl provides a dramatically easier way to build integration applications ranging from simple graphical user interfaces to complex financial, Web, and management applications. This means that all you need to do is to write a script, insert it into your HTML, and you're ready to roll.

Scripting languages are a simple way to add functionality to any Web page. They allow you to write simple programs directly inside your HTML. These programs then get executed on the user's browser or on the server before the page gets to the user. By using scripting, you can validate the data in client forms, store user preferences on the server and build a Web page on the fly as the user requests it, and much more.

Client-Side Tcl Applications

Client-side scripting is executed locally on the user's browser. It can increase the performance of a Web site, because a level of processing can be performed on the browser without having to make a round-trip to the server. One of the most common uses for client-side scripting is form validation, where you can check the user input before passing it onto the server.

The main advantages to client-side scripting are as follows:

- Performance
- By processing user input locally, you avoid the need for a round-trip to the server.
- Interactivity
- Using scripting you can build multimedia style interactivity into your pages.
- Controlling ActiveX objects and applets
- You can manipulate controls and applets through client-side scripting.

Tcl Plug-In

Plug-ins are external programs that are loaded into Netscape Navigator on demand to extend its capabilities to display Web content that Navigator itself does not know how to format and render. Before plug-ins, non-html Web content had to be displayed in local viewers outside the Web browser. The Tcl plug-in is a viewer for Tcl and Tk applications, called Tclets. It allows you to embed the Tclets in a Web page and to display Tk user interfaces in windows in the main browser window with normal HTML.

When the pages are viewed with Netscape Navigator and the plug-in, the Tclets are executed to provide custom user interfaces and simple animations inside the browser. Plug-in scripts have access to all of the power of Tcl and Tk, except for a few features removed for security reasons. For example, plug-in scripts can use the powerful Tk text and canvas widgets, they can create Tcl timer events for simple animations, and they can use Tcl's extensive library of string manipulation commands.

There are currently no Tcl plug-ins for wireless browsers.

Tcl And Integration

Businesses and engineering teams today are often faced with the problem of making diverse collections of resources work together. These programming tasks are called *integration applications*. The resources managed by integration applications can take many forms, such as:

- **Components** implemented using frameworks such as ActiveX or JavaBeans.
- **Devices** such as manufacturing equipment or test equipment.
- **Applications** such as Web servers, enterprise applications, and computer-aided design tools.
- **Data sources** such as databases, live news feeds, and Web sites.
- **Data formats** such as HTML and XML.
- **Protocols** such as CORBA, DCOM, and HTTP.

Typically integration applications involve communicating between resources such as these, co-ordinating their operation, and extending their basic capabilities with additional functions.

Drivers For Integration Applications

Integration applications have existed for many years but their importance has risen dramatically over the last 5-10 years and will continue to rise in the years ahead. The need for integration applications is being driven by some of the most important trends in the software industry, such as online information management, networked control of devices, network aggregation, and component frameworks and protocols.

Online Information Management

More and more of the world's information is being stored and accessed online; managing and presenting this information is an integration application. For example, high-end Web sites such as those at AOL's Digital City, CNET, and Travelocity create customized Web pages for their visitors by integrating information from different sources such as databases, live news feeds, and travel reservation systems. Or, Wall Street trading firms must filter and integrate information from a variety of different sources in order to provide traders with the information they need.

Networked Control Of Devices

Over the last decade computers have been embedded in many of the world's devices. These embedded processors typically provide remote control interfaces, and with the rise of the computer networking it has become easy to reach them and manage them remotely. Managing collections of networked devices is an integration application. For example, in factory automation the various pieces of equipment on a factory floor must be integrated with each other and with databases to control and track production. Or, in automated testing the devices are diverse pieces of test equipment, which must be coordinated to exercise a test device and extract performance information.

Network Aggregation

The rise of the Internet, combined with enterprise consolidation, is creating large and heterogeneous networks of hardware and software systems. Making these systems work well together is an integration application. For example, different departments of hospitals have historically developed their own (different) computer systems. In the past there was no attempt to communicate between the departments so the differences were not noticed. Now, however, hospitals are integrating all of their systems in order to coordinate and better manage the flow of information. As a result, they must now deal with the differences in the departmental systems. Another example is corporate growth by mergers and acquisitions. A company may initially have computer systems that are uniform, but in a merger it is likely to inherit systems that are different. After the merger, all of these systems must be integrated to work together.

Component Frameworks And Protocols

Several component frameworks, such as ActiveX and JavaBeans, have become popular in recent years. These frameworks encourage the creation of reusable software components. Making components work together is an integration problem. The rise of the Internet has also led to numerous protocols for communicating between components and applications, such as CORBA and DCOM. Using these protocols is also an integration task. In addition, it is becoming increasingly important to tie together different protocols and component frameworks, which is an even more challenging integration task.

Why Traditional Programming Approaches Don't Work

Integration applications have characteristics quite different from traditional programming tasks. In traditional programming the main problem is to create data structures and algorithms from scratch. System programming languages such as C, C++, and Java work well for these tasks.

In integration applications, data structures and algorithms are not usually very important: they are encapsulated in the components being integrated. The most difficult challenges in integration applications have to do with linkage and coordination:

- How to deal with the variety of things being integrated?
- How to make them all work together?
- How to evolve the applications rapidly in response to changing needs?

Integration applications often incorporate business rules and processes. As a result, they tend to be ill-structured and evolve rapidly. It is difficult to plan integration applications in advance, and they must often be modified in the field as needs change.

Furthermore, the people who write integration applications are often not sophisticated programmers. Their primary area of expertise is typically in a particular domain such as factory management, financial services, or Web content. They use programming as part of their job but don't consider themselves to be first and foremost programmers.

Because of these characteristics, system programming languages such as C, C++, or even Java are poorly suited to integration tasks. The main problem with system programming languages is inflexibility, which stems from their compilation and strong typing. Compilation freezes the functionality of an application at the time it is compiled, making it difficult to modify or extend the behavior of the application in the field. Strong typing forces programs to be carefully planned in advance and makes it difficult for them to be used in ways that weren't planned. As a result, system programming languages cannot accommodate the variety and rapid evolution that are typical in integration applications. System programming languages are also too complex and difficult to learn for many of the people who build integration applications.

Thus it is expensive to build integration applications with system programming languages, and the resulting applications tend to be brittle and inflexible.

The Power Of Tcl

In contrast, Tcl provides a superb platform for creating integration applications. Tcl's power comes from two basic features.

First, Tcl makes it easy to connect to any of the things you need to integrate. If you need to connect any X to any Y, it is easy to create one Tcl extension that connects to X, another that connects to Y, and use Tcl as the intermediary between them. Dozens of free extensions are already available for database access, network management, and many other purposes.

Second, with Tcl it is easy to write scripts that manage the connections in powerful ways. In contrast to system programming languages, Tcl is interpreted and typeless. The interpreted nature of Tcl makes it easy to modify and extend applications on the fly and evolve them rapidly. By being typeless and string-oriented, Tcl hides the differences between components and makes it easy to move information between them.

The combination of these two features allows integration applications to be developed much more efficiently with Tcl than with system programming languages such as C++ or Java, measured either in development time or in lines of code. Furthermore, the applications created with Tcl are more powerful and flexible.

Tcl Architecture

Tcl has a simple structure. Each line starts out with a command, such as `button` and a number of arguments. Each command is interpreted by an interpreter, which is a library of C procedures. This function is responsible for handling all the arguments.

Interpreters

Since Tcl is an interpreted language, to run a Tcl program, you normally pass the script file to the Tcl interpreter, **wish**, for example:

```
 wish hello.tcl
```

The interpreter is Tcl's central data structure. An interpreter consists of a set of command bindings, a set of variable values, and a few other miscellaneous pieces of state. Each Tcl command is interpreted in the context of a particular interpreter. Some Tcl-based applications will maintain multiple interpreters simultaneously, each associated with a different widget or portion of the application. Interpreters are relatively lightweight structures. They can be created and deleted quickly, so application programmers should feel free to use multiple interpreters if that simplifies the application. Eventually Tcl will provide a mechanism for sending Tcl commands and results back and forth between interpreters, even if the interpreters are managed by different processes.

You can also use **wish** in interactive mode and type in commands at the command line.

There's another standard Tcl interpreter, **tclsh**, which only understands the Tcl language. **tclsh** does not have any of the Tk user interface commands, so you cannot create graphical programs in **tclsh**.

Some Tcl freeware applications extend the Tcl language by adding new commands written as C functions. If such is the case, you need to compile the application instead of just passing its Tcl code to the **wish** interpreter. This application program, from a Tcl perspective, is really a new version of the **wish** interpreter, which the new C commands linked in. Of course, the application program may be a lot more than merely a Tcl interpreter.

Data Types

Tcl supports only one type of data: strings. All commands, all arguments to commands, all command results, and all variable values are strings. Where commands require numeric arguments or return numeric results, the arguments and results are passed as strings. Many commands expect their string arguments to have certain formats, but this interpretation is up to the individual commands. For example, arguments often contain Tcl command strings, which may get executed as part of the commands. The easiest way to understand the Tcl interpreter is to remember that everything is just an operation on a string. In many cases Tcl constructs will look similar to more structured constructs from other languages. However, the Tcl constructs are not structured at all; they are just strings of characters, and this gives them a different behavior than the structures they may look like.

Although the exact interpretation of a Tcl string depends on who is doing the interpretation, there are three common forms that strings take: commands, expressions, and lists.

Commands

A Tcl command string consists of one or more commands separated by newline characters or semi-colons. Each command consists of a collection of fields separated by white space (spaces or tabs). The first field must be the name of a command, and the additional fields, if any, are arguments that will be passed to that command. For example, the command

```
set a 22
```

has three fields: the first, `set`, is the name of a Tcl command, and the last two, `a` and `22`, will be passed as arguments to the `set` command. The command name may refer either to a built-in Tcl command, an application-specific command bound in with the library procedure `Tcl_CreateCommand`, or a command procedure defined with the `proc` built-in command. Arguments are passed literally as text strings. Individual commands may interpret those strings in any fashion they wish. The `set` command, for example, will treat its first argument as the name of a variable and its second argument as a string value to assign to that variable. For other commands, arguments may be interpreted as integers, lists, file names, or Tcl commands.

Expressions

Several commands, such as `expr`, `for`, and `if`, treat one or more of their arguments as expressions and call the Tcl expression processors (`Tcl_ExprLong`, `Tcl_ExprBoolean`, etc.) to evaluate them. The operators permitted in Tcl expressions are a subset of the operators permitted in C expressions, and they have the same meaning and precedence as the corresponding C operators. Expressions almost always yield numeric results (integer or floating-point values).

A Tcl expression consists of a combination of operands, operators, and parentheses. White space may be used between the operands and operators and parentheses; it is ignored by the expression processor. Where possible, operands are interpreted as integer values. Integer values may be specified in decimal (the normal case), in octal (if the first character of the operand is 0), or in hexadecimal (if the first two characters of the operand are 0x). If an operand does not have one of the integer formats given above, then it is treated as a floating-point number if that is possible. Floating-point numbers may be specified in any of the ways accepted by an ANSI-compliant C compiler. If no numeric interpretation is possible, then an operand is left as a string (and only a limited set of operators may be applied to it).

Lists

A list is just a string with a list-like structure consisting of fields separated by white space. For example, the string

```
    Al Sue Anne John
```

is a list with four elements or fields. Lists have the same basic structure as command strings, except that a newline character in a list is treated as a field separator just like space or tab.

Tcl Memory Usage

Running on 32-bit Windows 98, I used a couple of tools in determining memory sizes. To determine Tcl and Tk's kernel size, I used a freeware program called PrcView. To determine Tcl program sizes, I used Tcl's `memory` command, accessible when compiling with the `TCL_MEM_DEBUG` flag turned on. This allows for memory information to be viewed at the command line from the Tcl shell. This feature doesn't work for Tk, since output directed to stdout/stderr is lost under Win32 when Tk is in control.

There is a source code patch called **tclWA.patch** that fixes this problem, but I couldn't get it to work, so I just used PrcView to determine Tk widget memory usage. I also used the `memory` command, and a technique borrowed from Vincent Wartelle, to verify data type sizes.

Kernel Size

Running on Windows 98, a 32-bit machine, the Tcl 8.2 kernel has a memory footprint of about 348 kilobytes. The Tk toolkit takes up another 385 kilobytes of memory. Other versions vary in footprint size. With debugging features turned off and excluding Tk, the Tcl 8.0.3 kernel memory footprint is about 300k. Tcl 8.3.0 is about 450k and Tcl 8.3.2 is near 600k. Another version of Tcl that will be discussed later, Tiny Tcl, occupies about 60 kilobytes of memory.

Since the domain of this report is the Mobile Internet, I'll use the Tcl/Tk 2.0 plugin kernels for the remainder of my memory determinations. With debugging off, the plugin has a footprint of 343 kbytes for Tcl and 386 kbytes for Tk. Allowing debugging features adds about another 3 kbytes to each.

Byte Alignment

Before discussing data type sizes, it's important to understand byte alignment, since this directly affects the size of the data type. 32 bit microprocessors typically organize memory as shown in Table 7. Memory is accessed by performing 32 bit bus cycles. 32 bit bus cycles can however be performed at addresses that are divisible by 4. (32 bit microprocessors do not use the address lines A1 and A0 for addressing memory).

The reasons for not permitting misaligned long word reads and writes are not difficult to see. For example, an aligned long word X would be written as X0, X1, X2 and X3. Thus the microprocessor can read the complete long word in a single bus cycle. If the same microprocessor now attempts to access a long word at address 0x000D, it will have to read bytes Y0, Y1, Y2 and Y3. Notice that this read cannot be performed in a single 32 bit bus cycle. The microprocessor will have to issue two different reads at address 0x100C and 0x1010 to read the complete long word. Thus it takes twice the time to read a misaligned long word.

Table 7. Byte Alignment *(Source EventHelix.com)*

	Byte 0	Byte 1	Byte 2	Byte 3
0x1000				
0x1004	X0	X1	X2	X3
0x1008				
0x100C		Y0	Y1	Y2
0x1010	Y3			

Compiler Byte Padding

Compilers have to follow the byte alignment restrictions defined by the target microprocessors. This means that compilers have to add pad bytes into user defined structures so that the structure does not violate any restrictions imposed by the target microprocessor.

Data Type Size

Since Tcl is a string based language, each new object will use a minimum of 24 bytes of memory.

each object =

any new string, number, date, value → 24 bytes + content size

any pre-existing string, data, value → 4 bytes (pointer only)

content size =

depends on encoding and data type :

one or two bytes per char for string values

may be 0 for a number (integer/double), if it is never used as a string

(therefore included in the core tcl object)

each list entry =

4 bytes + object size of the content

- Each different thing in a Tcl program will cost 24 bytes

- Variables and hash-tables are costly:

  52 bytes overhead for each variable,

  52 bytes overhead for each hash table key

- Lists are not costly:

  4 bytes overhead for each element.

On a 32 bit machine where alignment is 4 byte boundary and the types have the following sizes,

```
long    4 bytes

int     4 bytes

char *  4 bytes

double  8 bytes

void *  4 bytes

sizeof (Tcl_Obj) = 4 + 4 + 4 + 4 + MAX (4, 8, 4, 4 + 4)

= 24 bytes
```

Well, it takes a certain amount of space to store the hash entry (four words plus the size of the key; about 20 bytes on a 32-bit machine) and more to store the variable (each entry in an array is an independent variable that can support its own traces, etc.) which adds another 8 words or 32 bytes. This gives about 52 bytes per array member.

Benchmark Tests

```
C:\Program Files\Tcl\tclplugin2.0>tclsh80

% memory info

total mallocs                  2873
total frees                       1611

current packets allocated         1262

current bytes allocated          83579

maximum packets allocated         1311

maximum bytes allocated          96094

% for {set i 0} {$i < 10000} {incr i} {

set t($i) abcd

}

% memory info

total mallocs                    63016

total frees                      41737

current packets allocated        21279

current bytes allocated         629318

maximum packets allocated        21291

maximum bytes allocated         629649
```

To determine the number of bytes per key, the number of bytes allocated before the function is subtracted from the number of bytes after, then divided by the number of iteration:

629318 – 83579 = 545739 / 10000 is approximately 54.6 bytes per key

1. hashtable with empty values

memory info

```
current bytes allocated       152681

...

% for {set i 0} {$i &lt; 10000 } { incr i } {

set t($i)

}

% memory info

current bytes allocated       698453

...

698453 - 152681 = 545772

approx 54 bytes per key.
```

2. hashtable with constant value

```
memory info

current bytes allocated       152550

...

% for {set i 0} {$i &lt; 10000 } { incr i } {

set t($i) abcd

}

% memory info

current bytes allocated       698363
```

```
...

698363 - 152550 = 545813

approx 54 bytes per key.
```

3. hashtable with variable value

```
memory info

current bytes allocated       152550

...

% for {set i 0} {$i &lt; 10000 } { incr i } {

set t($i) &quot;abcd_$i&quot;

}

% memory info

current bytes allocated      1037220

...

1037220  - 152550 = 884670

approx 89 bytes per key.
```

4. empty global variables

```
% memory info

current bytes allocated       152550

...
```

```
% for { set i 1 } { $i &lt;= 10000 } { incr i } {

set ::a[set i] &quot;&quot;

}

% memory info

current bytes allocated      729761

...

729761 - 152550 = 577211

approx 57 bytes per variable
```

5. global variables with the same value

```
% memory info

...

current bytes allocated       152550

% for { set i 1 } { $i &lt;= 10000 } { incr i } {

set ::a[set i] &quot;abcd&quot;

}

% memory info

...

current bytes allocated       708202

708202 -  152550 = 555652

approx. 55 bytes per variable.
```

6. global variables with different values

```
% memory info

...

current bytes allocated       152550

% for { set i 1 } { $i &lt;= 10000 } { incr i } {

set ::a[set i] &quot;abcd_$i&quot;

}

% memory info

...

current bytes allocated     1047070

1047070 -152550 = 894520

approx 89 bytes per variable.
```

7. empty list entries

```
% memory info

...

maximum bytes allocated       152550

% for {set i 1 } { $i &lt;= 10000 } { incr i } {

lappend l &quot;&quot;

}

% memory info

...

current bytes allocated       202179
```

```
        202179 - 152550 = 49629

        approx 5 bytes per list entry.
```

6. identic list entries

```
        % memory info

        ...

        current bytes allocated       152550

        % for {set i 1 } { $i &lt;= 10000 } { incr i } {

        lappend ::l abcd

        }

        % memory info

        ...

        current bytes allocated       202215

        202215 - 152550 = 49665

        approx 5 bytes per list entry.
```

7. different list entries

```
        % memory info

        ...

        current bytes allocated       152550

        % for {set i 1 } { $i &lt;= 10000 } { incr i } {

        lappend ::l &quot;abcd_$i&quot;

        }
```

```
% memory info

...

current bytes allocated     541083

541083 - 152550 = 428533

approx 43 bytes per list entry.
```

Extrusion of Tcl/Tk

Because of the power, portability, and easy use of Tcl, there has been much discussion and desire among developers for a smaller version of Tcl that can run on a PDA. As this report was being completed, two notable versions of Tcl that have been released, targeting small memory devices. Available documentation is scarce and I haven't had the opportunity to fully explore the source codes, Tiny Tcl and Palm Tcl do appear to apply extrusion principles. There are also at least two other versions of Tcl for the Palm OS being currently developed, but yet to be released.

Tiny Tcl

Tiny Tcl was announced for release on May 7, 2001 by Karl Lehenbauer through the comp.lang.tcl newsgroup. Tiny Tcl 6.8 is a rommable, minimal Tcl for embedded applications. Derived from the venerable Tcl 6.7 release, Tiny Tcl 6.8 has a solid Tcl feature set, excluding newer capabilities of Tcl 7 and 8 such as the bytecode compiler, namespaces, sockets, and async event handling, among others.

Excluding C library functions, Tiny Tcl compiles down to less than 60 Kbytes on most machines, far smaller than any Tcl 7 or Tcl 8 derivatives. On an embedded DOS system with 640K of RAM, programs of up to several thousand lines of code can be executed.

Tiny Tcl should not be confused with TinyTcl, released by John-Claude Whippler in 1999. TinyTcl wasn't written in Tcl. It was C++ code that emulated Tcl's behavior.

Palm Tcl

Two days after Tiny Tcl was released, Palm Tcl Version 0.1 was announced on the same newsgroup. Developed by Ashok P. Nadkarni, it's intended for creating applications for handheld devices running the Palm OS operating system. It's limitations include no shared library, inability to handle finds, it does not support full Palm OS API, can only read / write databases created through Palm Tcl, and has no visual design tool.

Palm Tcl is based on Tcl 7.6 but has some significant differences. Some commands have been removed because they are not relevant to the Palm OS environment. These commands are – `cd exit file flush gets glob load open puts pwd read seek tell`.

Dynamic memory and stack space is extremely limited in Palm OS devices. Some commands had to be removed because of their memory usage. Some (memory -optimized) form of these commands may be added in future versions. These commands are – `clock history regexp regsub`.

Some commands have not been implemented to reduce the Palm Tcl code footprint. These are – `case interp package`. The following commands have not been implemented but will be in a future release – `after close eof flush socket vwait`. The following new commands have been added to support Palm OS features – `dm form fatal memstat`.

Although Palm OS does provide low level windowing and graphics routines, Palm Tcl is focused on applications that are based on Palm OS *forms.* A Palm OS form is basically a container window with a title and menu that collects together various user interface elements, such as buttons and text fields, that are functionally related. An application usually consists of a number of these forms that the user navigates through using buttons or menu selections.

Tk

Although Palm Tcl replaces the Tk GUI capabilities by using Palm's *forms*, there still is a desire and need to have a smaller version of Tk that can provide it's own GUI capabilities on other devices. There is work being done to improve cross-platform GUI differences, such as TkGS (the Tk Graphics System), but nothing that targets memory limited devices directly.

Extruding Tk

A Sample Application

To test the extrusion benefit for Tk, I used a simple calculator application (see Figure 4:) written by Sydney R. Polk of Sun Microsystems. The source code is included as APPENDIX C. The reasons for selecting this program are that it has practical use on a PDA, and because it contains two widgets, label and button, that can be modified to illustrate the differences between standard Tk and a modified version of Tk.

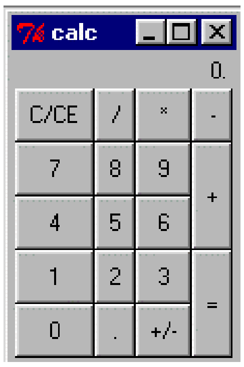


Figure 4: Tcl/Tk Calculator Script *(Source: S. Clemons)*

Creating the Calculator

BUILDING A WINDOW

All Tk applications are created in a window. Tk uses a token of type `tk_Window` to represent each window. When a new window is created, Tk returns a `tk_Window` token. The structure is defined in tclInit.h and is included as APPENDIX D.

THE GRID

The `grid` is a geometry manager that arranges widgets on a grid with variable-sized and rows. The Tk_GridCmd procedure in tkGrid.c is invoked to process the `grid` command. The `label` and `button` widgets will be place in the window using the `grid` geometry manager.

THE *LABEL* AND *BUTTON* WIDGETS

The Tk widget, label, is a simple widget. It appears as a colored rectangular region and might have a 3d border. A label can also display a string or a bitmap. The Tk widget, button, is a similar to the label widget, but it also responds to mouse events. Table 8 Lists their modifiable attributes.

Table 8. Tk `label` and `button` Widget Attributes *(Source: S. Clemons)*

Attribute	Function	`label`	`button`
`activeBackground`	Background color when the mouse is over the button	x	x
`activeForeground`	Text color when mouse is over button		x
`anchor`	Relative position of text	x	x
`background`	Specifies the normal background color to use when displaying the widget.	x	x
`bitmap`	Specifies a bitmap to display in the widget	x	x
`borderwidth`	The width of the 3-D border to draw around the outside of the widget	x	x
`command`	Tcl command to invoke		x
`cursor`	Specifies the mouse cursor to be used for the widget	x	x
`default`	Display a default or normal button		x
`direction`	Offset direction for posting menus		x
`disabledForeground`	Specifies foreground color to use when drawing a disabled element		x
`font`	Specifies the font to use when drawing text inside the widget	x	x
`foreground`	Specifies the normal foreground color to use when displaying the widget	x	x
`height`	Height, in lines of text, or screen units for images	x	x
`highlightBack-ground`	Specifies the color to display when the widget does not have the input focus	x	x
`highlightColor`	Specifies the color to use around the widget when it has the input focus	x	x
`highlightThickness`	The width of the highlight border around the outside of the widget when it has the input focus	x	x
`image`	Specifies an image to display in the widget	x	x
`indicatorOn`	Boolean that controls if the indicator is displayed: used with `checkbutton`, `radiobutton`, or `menubutton`		x
`justify`	how the text lines line up with each other	x	x
`menu`	Menu posted when `menubutton` is clicked		x
`offValue`	Variable value when `checkbutton` is not selected		x
`onValue`	Variable value when `checkbutton` is selected		x
`padx`	How much extra space to request for the widget in the X-direction	x	x
`pady`	How much extra space to request for the widget in the Y-direction	x	x
`relief`	Specifies the 3-D effect desired for the widget	x	x
`selectColor`	Color for selector. `checkbutton` or `radiobutton`		x
`selectImage`	Alternate graphic for selector. `checkbutton` or `ra-diobutton`		x
`state`	Enabled or deactivated		x
`takeFocus`	Determines whether the window accepts the focus during keyboard traversal (e.g., Tab and Shift-Tab).	x	x
`text`	Specifies a string to be displayed inside the widget	x	x

`textVariable`	Tcl variable that has the value of the text	x	x
`underline`	Specifies the integer index of a character to underline in the widget	x	x
`value`	Value for Tcl variable when `radiobutton` is selected		x
`variable`	Text variable associated with a `checkbutton` or `radiobutton`		x
`width`	Width in characters for text, or screen units for image	x	x
`wrapLength`	Maximum character length before text is wrapped	x	x

In Tcl8.0, both the button and label commands are part of the TkCmd structure in tkWindow.c and are implemented through tkButton.c. The widgets' attributes are defined in the tkButton structure in tkButton.h:

```
typedef struct {
    Tk_Window      tkwin;
    Display       *display;
    Tcl_Interp    *interp;
    Tcl_Command    widgetCmd;
    int            type;

    Tk_3DBorder    activeBorder;
    XColor        *activeFg;
    GC             activeTextGC;
    Tk_Anchor      anchor;
    Pixmap         bitmap;
    int            borderWidth;
    char          *command;
    GC             copyGC;
    Tk_Cursor      cursor;
    Tk_Uid         defaultState;
    Xcolor        *disabledFg;
    GC             disabledGC;
    int            flags;
    Pixmap         gray;
    int            height;
    char          *heightString;
    Tk_3Dborder    highlightBorder;
    XColor        *highlightColorPtr;
    int            highlightWidth;
    Tk_Image       image;
    char          *imageString;
    int            indicatorDiameter;
    int            indicatorOn;
    int            indicatorSpace;
    int            inset;
    Tk_Justify     justify;
    Tk_3Dborder    normalBorder;
    XColor        *normalFg;
    GC             normalTextGC;
    char          *offValue;
    char          *onValue;
    int            padX, padY;
```

```
        Int            relief;
        Tk_3Dborder    selectBorder;
        Tk_Image       selectImage;
        char          *selectImageString;
        char          *selVarName;
        Tk_Uid         state;
        char          *takeFocus;
        char          *text;
        int            textHeight;
        Tk_TextLayout  textLayout;
        int            textWidth;
        char          *textVarName;
        Tk_Font        tkfont;
        int            underline;
        int            width;
        char          *widthString;
        int            wrapLength;

    } TkButton;
```

The widgets in the

Size of The Calculator In Memory

Without any extrusion performed on the Tk package, calc.tcl used

403,320 (wish running calc.tcl) — 396,320 bytes (wish only) = 7 kbytes

Applying Extrusion To The Calculator

To reduce calc.tcl's memory usage, it is first determined which attributes are not necessary for wireless applications. When a program implements a button widget, memory is allocated for the entire structure. So attributes that are not used just waste space in memory. When considering developing an application for a hand-held device, the TkButton structure can be rewritten to remove unnecessary elements.

First, I decided to remove all attributes associated with a mouse, this included activeBackground, and activeForeground, which specify a background or text color to display when the mouse is over the button. Since my program wasn't going to use a mouse, there was no need for the cursor attribute, which selects the type of cursor to display when the mouse is moved over the widget. I also removed the command attribute, which executes a Tcl command when a button is pressed, because keyboard strokes can be binded to the button instead.

I also decided to remove all color related attributes, which included `foreground`, `highlightBackground`, `highlightColor`, `highlightThickness`, and `selectColor`. Keeping the `bitmap` option, I was able to remove `image` and `selectImage` and still retain custom graphic capabilities.

Once I determined which attributes to remove, I simply commented them out of the source and header files. The source files for Tcl and Tk are free and readily available for download from numerous sources on the Internet. The source files came with a make file to aid in the compilation process. The make file (makefile.vc) is executed with Microsoft's Visual C++ 5.0. There was quite a bit of modification required to the locations of various executable files and libraries in the make file to get the program's to build correctly.

Since a Tk `button` and `label` share the same attributes, there structures are stored in the tkButton.h and tkButton.c files. Compilation revealed that there were also some necessary modifications to be made to a couple of Windows operating system specific files – tkWinButton and tkWinWm.c (Window's window manager). Once those modifications were made, a new version of Tcl and Tk were successfully compiled.

RESULTS and discussions

The Results

Running the calculator script with the modifications made to the `button` attributes revealed no differences, as far as the end-user is concerned, with the application. The buttons and labels performed as they did with the unmodified version of Tk.

With no changes made to the calc.tcl script, a small memory savings was achieved simply by removing unnecessary attributes from the Tk `button` widget. Unexpectedly, not only did the application require less memory, but the **wish** shell also allocated 240 bytes less memory when executed,, although the total allocated memory remained unchanged. After reviewing data types sizes in section 0, this actually makes sense. Since each data type is 24 bytes plus the size of the contents (0 in this case), removing the 10 attributes should total 240 bytes. The tk80.dll dynamic loaded library also was reduced by 4096 bytes.

Prior to any extrusion to the Tk package, calc.tcl used 98,328 bytes of memory from the heap:

1,442,144 bytes (wish running calc.tcl) – 1,343,816 bytes (wish only) = 98,328 bytes

After extrusion, calc.tcl used only 93,768 bytes, a savings of 4560 bytes:

1,437,344 bytes (wish running calc.tcl) – 1,343,576 bytes (wish only) = 93,768 bytes

This number is also consistent with expected results. Since the calculator script includes 18 buttons and 1 label, and each widget had 10 attributes removed, a total memory savings of 4560 bytes should result:

10 attributes x 24 bytes saved each = 240 bytes per widget x 19 widgets = 4560 bytes total

Discussion

It is obvious that a more thorough extrusion of the Tk widgets, targeted at particular mobile devices, can result in significant memory usage reduction. Enough so that a GUI tool kit can be made available to provide feature rich applications to mobile users.

There is currently no governing body of the Tcl/Tk language. Being an open source language, the community relies on its own developers to provide new versions, enhancements, and bug fixes for it. The lack of structure does hamper the progression of Tcl/Tk, but also awards the freedom for users to develop the language as they see fit.

As stated before, there is an active interest in a version of Tk that can be ported to various devices. A thorough knowledge of the targeted device, it's capabilities and limitations, is required to provide a tool that is truly useful to both the developer and the end-user. My knowledge is limited in that respect, but the information gathered in this report will be made available to the Tcl community so that perhaps those with competence can use the principles of extrusion to finish the work started here.

conclusions and recommendations

A Return To Simpler Days

There was a time, not so long ago, that memory costs and restrictions forced programmers to develop very tight code - programs that managed memory extremely well. As memory prices have dropped, so has the need to write memory conscious software. Feature intense applications are being developed with little or no regard for memory consumption. Many of these features are not ever used. With the dawn of the Mobile Internet, we have come full cycle. Mobile users still demand application quality similar to that they have grown accustomed to on their desktops. Anything less, and many feel it's not worth the trouble. The responsibility now lies on the developer to provide similar applications in a much more limited environment.

For developers to provide mobile users with value-added applications, we must relearn the lost art of memory management. But not even that will be enough. Application languages like XHTML Basic, CHTML, J2ME, Oracle Lite, and yes, even Tiny Tcl, have shown that software engineers can provide alternative solutions to the memory crisis. By applying extrusion principles and removing unnecessary components and reducing unneeded attributes from those that comprise an application, smaller applications with similar features are being produced for mobile users.

Extrusion provides a viable method to develop content-rich software applications for mobile devices without increasing hardware capabilities. Costs are limited by increasing developer productivity by negating the need to learn new languages or rewrite existing applications.

An across the board solution for all languages is not easily implemented, though. The problem is a catch-22. The Mobile Internet hasn't take off because the applications are so limited, and corporations are not quick to spend money developing new languages for a domain that doesn't have many users. But in the not too distant future, that will change. Languages like J2ME and Oracle Lite are providing quality applications to the mobile environment. This, coupled with improved hardware and increased bandwidth over wireless networks, will attract huge numbers of users to the Mobile Internet.

The big companies like Sun and Oracle have the personnel and financial resources to take on such endeavors. J2ME and Oracle Lite may not always be the best solution for a particular mobile application, but they may be the only ones. For those developers that are most comfortable and productive working with other languages, such as C++, Visual Basic, Perl, Python, Lisp, and SmallTalk, the solution is a grass-roots approach.

We have to provide our own efforts to extrude our language of choice into one that can be ported to mobile devices. Learn about memory management, small memory architecture, and component-based engineering. Join a news group of your respective language to keep up to date on individual efforts in your domain. Look closely at the components that will comprise a given application and then apply the principles of extrusion to reduce memory consumption. With knowledge and effort, the software engineers of today will be able to provide the application features that end-users demand today, not tomorrow.

APPENDIX A
XHTML Standard TAGS vs xHTML Basic Tags

Table 9. XHTML Standard Tags vs. XHTML Basic Tags *(Source: S. Clemons)*

XTML Tag	Description	XHTML Attributes	XHTML Basic Attributes	Module
<a>	Defines an anchor	accesskey charset class href hreflang id rel rev tabindex title type xml:lang onclick ondblclick onmousedown onmouseu onmouseover onmousemove onmouseout onkeypress onkeydown onkeyup style	accesskey charset class href hreflang id rel rev tabindex title type xml:lang	Client-Side Image Map Hypertext Intrinsic Name Target
<abbr>	Defines an abbreviation	class id title xml:lang onclick ondblclick onmousedown onmouseu onmouseover onmousemove onmouseout onkeypress onkeydown onkeyup style	class id title xml:lang	Text
<acro-nym>	Defines an acronym	class id title xml:lang onclick ondblclick onmousedown onmouseu onmouseover onmousemove onmouseout onkeypress	class id title xml:lang	Text

		onkeydown onkeyup style		
<applet>				Applet Name
<ad- dress>	Defines an address element	class dir id lang onclick ondblclick onkeydown onkeypress onkeyup onmousedown onmousemove onmouseout onmouseover onmouseup style title xml:lang	class id title xml:lang	Text
<area>	Defines an area inside an image map			Client-Side Image Map Intrinsic Target
<b>	Defines bold text			Presentation
<base>	Defines the base URL or target for the anchors in the Web document.		href	Base Target
<bdo>	Defines or alters the default algo-rithm used for the language and display direction.			Bi-Directional Text
<big>	Defines big text			Presentation
<blockqu ote>	Defines a long quotation	class id title xml:lang onclick ondblclick onmousedown onmouseu onmouseover onmousemove onmouseout onkeypress on- keydown onke- yup style cite	cite class id title xml:lang	Text
<body>	Defines the body element		class id title xml:lang	Intrinsic Structure
<br>	Inserts a single line break	class id title	class id title	Text
<button>	Defines a push button			Forms Intrinsic
<cap- tion>	Defines a table caption		align class	Basic Tables Tables

			id title xml:lang	
<cite>	Defines a citation	class id title xml:lang onclick ondblclick onmousedown onmouseu onmouseover onmousemove onmouseout onkeypress on- keydown onke- yup style	class id title xml:lang	Text
<code>	Defines computer code text	class id title xml:lang onclick ondblclick onmousedown onmouseu onmouseover onmousemove onmouseout onkeypress on- keydown onke- yup style	class id title xml:lang	Text
<col>	Defines attributes for table col- umns			Tables
<col- group>	Defines groups of table columns			Tables
<dd>	Defines a definition description		class id title xml:lang	List
<del>	Defines deleted text			Edit
<dfn>	Defines a definition term	class id title xml:lang onclick ondblclick onmousedown onmouseu onmouseover onmousemove onmouseout onkeypress on- keydown onke- yup style	class id title xml:lang	Text
<div>	Defines a section in a document	class id	class id	Text

		title xml:lang onclick ondblclick onmousedown onmouseu onmouseover onmousemove onmouseout onkeypress on- keydown onke- yup style	title xml:lang	
<dl>	Defines a definition list		class id title xml:lang	List
<dt>	Defines a definition term		class id title xml:lang	List
<em>	Defines emphasized text	class id title xml:lang onclick ondblclick onmousedown onmouseu onmouseover onmousemove onmouseout onkeypress on- keydown onke- yup style	class id title xml:lang	Text
<field- set>	Defines a fieldset			Forms
<form>	Defines a form		action class enctype id method title xml:lang	Basic Forms Forms Intrinsic Name Target
<frame>	Defines a sub window (a frame)			Name
<frame- set>	Defines a set of frames			Intrinsic
<h1>	Defines header 1	class id title xml:lang onclick ondblclick onmousedown onmouseu onmouseover onmousemove onmouseout onkeypress on-	class id title xml:lang	Text

		keydown onke- yup style		
<h2>	Defines header 2	class id title xml:lang onclick ondblclick onmousedown onmouseu onmouseover onmousemove onmouseout onkeypress on- keydown onke- yup style	class id title xml:lang	Text
<h3>	Defines header 3		class id title xml:lang	Text
<h4>	Defines header 4	class id title xml:lang onclick ondblclick onmousedown onmouseu onmouseover onmousemove onmouseout onkeypress on- keydown onke- yup style	class id title xml:lang	Text
<h5>	Defines header 5	class id title xml:lang onclick ondblclick onmousedown onmouseu onmouseover onmousemove onmouseout onkeypress on- keydown onke- yup style	class id title xml:lang	Text
<h6>	Defines header 6	class id title xml:lang onclick ondblclick onmousedown onmouseu onmouseover	class id title xml:lang	Text

		onmousemove onmouseout onkeypress on-keydown onke-yup style		
<head>	Defines information about the document		xml:lang	Structure
<hr>	Defines a horizontal rule			Presentation
<html>	Defines an html document		version xml:lang xmlns	Structure
<i>	Defines italic text			
<iframe>	Defines an inline sub window (frame)			IFrame Name
<img>	Defines an image		alt class height id longdesc src title width xml:lang	Client -Side Image Map Image Name Server-Side Image Map
<input>	Defines an input field		checked class id maxlength name size src title type value xml:lang	Basic Forms Client -Side Image Map Forms Intrinsic Server-Side Image Map
<ins>	Defines inserted text			Edit
<kbd>	Defines keyboard text		class id title xml:lang	Text
<label>	Defines a label		accesskey class for id title xml:lang	Basic Forms Forms Intrinsic
<legend>	Defines a title in a fieldset			Forms
<li>	Defines a list item		class id title xml:lang	List
<link>	Defines a resource reference		charset class href hreflang	Link Target

			id media rel rev title type xml:lang	
<map>	Defines an image map			Client-Side Image Map Name
<meta>	Defines meta information		content http-equiv name scheme xml:lang	Meta Information
<no-frames>	Defines a noframe section			
<no-script>	Defines a noscript section			Scripting
<object>	Defines an embedded object			Client-Side Image Map Object
<ol>	Defines an ordered list		class id title xml:lang	List
<optgroup>	Defines an option group			Forms
<option>	Defines an item in a list box		class id selected title value xml:lang	Basic Forms Forms
<p>	Defines a paragraph		class id title xml:lang	Text
<param>	Defines a parameter for an object			Applet Object
<pre>	Defines preformatted text		class id title xml:lang xml:space	Text
<q>	Defines a short quotation		cite class id title xml:lang	Text
<samp>	Defines sample computer code		class id title xml:lang	Text
<script>	Defines a script			Scripting
<select>	Defines a selectable list		class id multiple name size title xml:lang	Basic Forms Forms Intrinsic

<small>	Defines small text			Presentation
<span>	Defines a section in a document		class id title xml:lang	Text
<strong>	Defines strong text		class id title xml:lang	Text
<style>	Defines a style definition			Style Sheet
<sub>	Defines subscripted text			Presentation
<sup>	Defines superscripted text			
<table>	Defines a table		border cellpadding cellspacing class id title width xml:lang	Basic Tables Tables
<tbody>	Defines a table body			Tables
<td>	Defines a table cell	abbr axis headers scope rowspan colspan Number cellhalign; cell-valign	abbr align axis class colspan headers id rowspan scope title valign xml:lang	Basic Tables Tables
<tex-tarea>	Defines a text area		class cols id name rows title xml:lang	Basic Forms Forms Intrinsic
<tfoot>	Defines a fixed table footer			Tables
<th>	Defines a table header		abbr align axis class colspan headers id rowspan scope title valign xml:lang	Basic Tables Tables
<thead>	Defines a fixed table header			Tables

<title>	Defines the document title		xml:lang	Structure
<tr>	Defines a table row		align class id title valign xml:lang	Basic Tables Tables
<tt>	Defines teletype text			Presentation
<ul>	Defines an unordered list		class id title xml:lang	List
<var>	Defines a variable		class id title xml:lang	Text

APPENDIX B

Compact HTML Tag List

Table Legend:
--HTML(2.0:HTML2.0, 3.2:HTML3.2, 4.0:HTML4.0)
–CH Compact HTML

Table 10. Compact HTML Tag List *(Source: W3C)*

No	Elements	Attributes	HTML	CH	Comments
1	!-	-	2.0	CH	--
2	!DOCTYPE	-	2.0	CH	--
3	&xxx;	-	2.0	CH	-- &amp;,&copy;,&gt;,&lt;,&quot;,&reg ;,&nbsp; --&#0;• `&#127;
4	A	name= href="URL" rel= rev= title= urn=(deleted from HTML3.2) methods=(deleted from HTML3.2)	2.0	CH CH - - - - -	--
5	ABBR	-	4.0	-	--
6	ACRONYM	-	4.0	-	--
7	ADDRESS	-	2.0	-	--Only one font.
8	APPLET	-	3.2	-	--(Deprecated element in HTML4.0)
9	AREA	shape= coords= href="URL" alt= nohref	3.2	-	--
10	B	-	2.0	-	--Only one font.
11	BASE	href="URL"	2.0	CH	--
12	BASEFONT	size=	3.2	-	--Only one font. --(Deprecated element in HTML4.0)
13	BDO	-	4.0	-	--
14	BIG	-	3.2	-	--Only one font.
15	BLOCKQUOTE	-	3.2	CH	--
16	BODY	- bgcolor= background= text= link= vlink= alink=	2.0 3.2 3.2 3.2 3.2 3.2 3.2	CH - - - - - -	--Non-white colors are drawn as black.
17	BR	- clear=all/left/right	2.0 3.2	CH CH	--
18	BUTTON	-	4.0	-	--
19	CAPTION	-	3.2	-	--

20	CENTER	-	3.2	CH	--(Deprecated element in HTML4.0)
21	CITE	-	2.0	-	--Only one font.
22	CODE	-	2.0	-	--Only one font.
23	COL	-	4.0	-	--
24	COLGROUP	-	4.0	-	--
25	DD	-	2.0	CH	--
26	DEL	-	4.0	-	--
27	DFN	-	3.2	-	--
28	DIR	- compact	2.0	CH -	--(Deprecated element in HTML4.0)
29	DIV	- align=left/center/right	3.2	CH CH	--
30	DL	- compact	2.0	CH -	--
31	DT	-	2.0	CH	--
32	EM	-	2.0	-	--Only one font.
33	FIELDSET	-	4.0	-	--
34	FONT	size=n size=+n/-n color=	3.2	- - -	--Only one font. --(Deprecated element in HTML4.0)
35	FORM	action= method=get/post enctype=	2.0	CH CH CH	--
36	FRAME	-	4.0	-	--(Frameset DTD)
37	FRAMESET	-	4.0	-	--(Frameset DTD)
38	HEAD	-	2.0	CH	--
39	Hn	- align=left/center/right	2.0 3.2	CH CH	--
40	HR	- align=left/center/right size= width= noshade	2.0 3.2 3.2 3.2 3.2	CH CH CH CH CH	--
41	HTML	- version=	2.0 3.2	CH CH	--version="C-HTML 1.0".
42	I	-	2.0	-	--Only one font.
43	IFRAME	-	4.0	-	--(Frameset DTD)
44	IMG	src= align=top/middle/bottom align=left/right width= height= hspace= vspace= alt= border= usemap= ismap=	2.0 2.0 3.2 3.2 3.2 3.2 3.2 2.0 3.2 3.2 2.0	CH CH CH CH CH CH CH CH CH - -	--Large images compressed automatically.
45	INPUT	type=text name= size= maxlength= value=	2.0	CH CH CH CH CH	--Max character buffer 512 bytes.

46	INS	-	4.0	-	--
47	ISINDEX	- prompt=	2.0 3.2	- -	--(Deprecated element in HTML4.0)
48	KBD	-	2.0	-	--Only one font.
49	LABEL	-	4.0	-	--
50	LEGEND	-	4.0	-	--
51	LI	- type=1/A/a/I/i type=circle/disc/square value=	2.0 3.2 3.2 3.2	CH - - -	--
52	LINK	href="URL" rel= rev= urn= methods= title= id=	2.0	-	--
53	LISTING	-	2.0	-	--Only one font. --(Obsolete element in HTML4.0)
54	MAP	name=	3.2	-	--
55	MENU	- compact	2.0	CH -	--(Deprecated element in HTML4.0)
56	META	name= http-equiv= content=	2.0	CH	--http-equiv="refresh" only.
57	NEXTID	n=	2.0	-	--Deleted from HTML3.2.
58	NOFRAMES	-	4.0	-	--(Frameset DTD)
59	NOSCRIPT	-	4.0	-	--
60	OBJECT	-	4.0	-	--
61	OL	- type=1/A/a/I/i start= compact	2.0 3.2 3.2 2.0	CH - - -	--
62	OPTGROUP	-	4.0	-	--
63	OPTION	- selected value=	2.0	CH CH -	--
64	P	- align=left/center/right	2.0 3.2	CH CH	--
65	PARAM	-	4.0	-	--
66	PLAINTEXT	-	2.0	CH	--(Obsolete element in HTML4.0)
67	PRE	- width=	2.0 3.2	CH -	--
68	Q	-	4.0	-	--
69	S	-	2.0	-	--(Deprecated element in HTML4.0)
70	SAMP	-	2.0	-	--Only one font.
71	SCRIPT	-	3.2	-	--
72	SELECT	name= size= multiple	2.0	CH CH CH	--Max character buffer 4 Kbytes.
73	SMALL	-	3.2	-	--Only one font.
74	SPAN	-	4.0	-	--
75	STRIKE	-	2.0	-	--(Deprecated element in HTML4.0)

76	STRONG	-	2.0	-	--Only one font.
77	STYLE	-	2.0	-	--
78	SUB	-	3.2	-	--
79	SUP	-	3.2	-	--
80	TABLE	- align=left/center/right etc. border= width= cellspacing= cellpadding=	3.2	-	--
81	TBODY	-	4.0	-	--
82	TD	- align=left/center/right valign=top/middle/bottom/baselin e rowspan= colspan= width= height= nowrap	3.2	-	--
83	TEXTAREA	name= rows= cols=	2.0	CH CH CH	--Max character buffer 512 bytes.
84	TFOOT	-	4.0	-	--
85	TH	- align=left/center/right valign=top/middle/bottom/baselin e rowspan= colspan= width= height= nowrap	3.2	-	--
86	THEAD	-	4.0	-	--
87	TITLE	-	2.0	CH	--
88	TR	- align=left/center/right valign=top/middle/bottom/baselin e	3.2	-	--
89	TT	-	2.0	-	--Only one font.
90	U	-	3.2	-	--(Deprecated element in HTML4.0)
91	UL	- type=disk/circle/square compact	2.0 3.2 2.0	CH - -	--
92	VAR	-	2.0	-	--Only one font.
93	XMP	-	2.0	-	--Only one font. --(Obsolete element in HTML4.0)

APPENDIX C

Tcl calculator source code

```
# calc - A simple calculator for use as a plug-in demo.
# Author: Steve Clemons
# This is a modification of Sydney Polk's (of Sun Microsystems)
# calc.tcl script.  This script replaces the command commands with
# keyboard bindings to remove mouse capabilities.
#
# The global variable state is used to keep track of what the user has
# done. It has several fields: Whether or not the last button was a
# CE, the current value of the expression, the current value of the
# entry window.

proc doClear {} {
    global state

    set state(entry) "0."
    set state(dot) 0
    if {$state(entrystarted) == 0} {
        set state(result) "0."
        set state(operation) ""
    }
    set state(entrystarted) 0
}

proc doDot {} {
    global state

    set state(dot) 1
}

proc doAppend {what} {
    global state

    if {$state(entrystarted) == 0} {
        set state(entry) "0."
    }

    if {[string compare $state(operation) ""] == 0} {
        set state(result) "0."
    }

    if {!($what == 0 && [string compare $state(entry) "0."] == 0)} {
        set state(entrystarted) 1
        if {$state(dot) == 1} {
            set state(entry) [format "%s%s" $state(entry) $what]
        } else {
            regexp {([-0123456789]+).} $state(entry) foo integer

        if {[string compare $integer "0"] == 0} {
            set state(entry) [format "%s." $what]
        } else {
            set state(entry) [format "%s%s." $integer $what]
        }
        }
    }
}

proc doOperation {what} {
    global state

    if {[string compare $state(operation) ""] != 0} {
        doEqual
    }
    set state(operation) $what
    set state(result) $state(entry)
    set state(entrystarted) 0
    set state(dot) 0
}

proc doEqual {} {
    global state

    if {[string compare $state(operation) ""] != 0} {
        set state(result) \
                [expr "$state(result) $state(operation) $state(entry)"]
    } else {
        set state(result) $state(entry)
    }
    set state(entry) $state(result)
    set state(entrystarted) 0
    set state(operation) ""
    set state(dot) 0
}

proc doSign {} {
    global state

    if {[string compare $state(entry) "0."] != 0} {
        set sign ""
        set abs ""
        regexp {([-]?)([0-9.]+)} $state(entry) foo sign abs
        if {[string compare $sign "-"] == 0} {
            set state(entry) $abs
        } else {
            set state(entry) [format "-%s" $abs]
        }
        set state(entrystarted) 1
    }
}

set state(result) "0."
set state(entry) "0."
set state(operation) ""
set state(dot) 0
set state(entrystarted) 0

label .label -textvariable state(entry) -justify right -anchor e

set i 48
foreach {number} {0 1 2 3 4 5 6 7 8 9} {
    set buttons($number) [button .$number -text $number]
    bind . <KeyPress-$number> "doAppend $number"
}

set buttons(clear) [button .clear -text C/CE -padx 1]
bind . <KeyPress-c> {doClear}
foreach {label operation} {div / mult * minus - plus +} {
    set buttons($label) [button .$label -text $operation]
    bind . <KeyPress-$operation> "doOperation $operation"
}

set buttons(dot) [button .dot -text .]
bind . <KeyPress-.> {doDot}
set buttons(sign) [button .sign -text +/- -padx 1]
bind . <KeyPress-m> {doSign}

set buttons(equal) [button .equal -text =]
bind . <KeyPress-=> {doEqual}

grid .label -column 0 -row 0 -columnspan 4 -sticky news
grid $buttons(clear) $buttons(div) $buttons(mult) $buttons(minus) \
      –sticky news
grid $buttons(7) $buttons(8) $buttons(9) -sticky news
grid $buttons(4) $buttons(5) $buttons(6) -sticky news
grid $buttons(1) $buttons(2) $buttons(3) -sticky news
grid $buttons(0) $buttons(dot) $buttons(sign) -sticky news
grid $buttons(plus) -column 3 -row 2 -rowspan 2 -sticky news
grid $buttons(equal) -column 3 -row 4 -rowspan 2 -sticky news
```

APPENDIX D

TK_WINDOW STRUCTURE

```
typedef struct TkWindow {

    Display *display;                     /* Display containing window. */
    TkDisplay *dispPtr;                  /* Tk's information about display
                                                          * for window. */
    int screenNum;              /* Index of screen for window, among all
                                                   * those for dispPtr. */
    Visual *visual;        /* Visual to use for window.  If not default,
                            * MUST be set before X window is created. */
    int depth;                                 /* Number of bits/pixel. */
    Window window;              /* X's id for window.   NULL means window
                              * hasn't actually been created yet, or it's
                                                        * been deleted. */
    struct TkWindow *childList;       /* First in list of child windows,
                                     * or NULL if no children.  List is in
                                * stacking order, lowest window first.*/
    struct TkWindow *lastChildPtr;
                               /* Last in list of child windows (highest
                                     * in stacking order), or NULL if no
                                                          * children. */
    struct TkWindow *parentPtr;     /* Pointer to parent window (logical
                               * parent, not necessarily X parent).  NULL
                               * means either this is the main window, or
                                   * the window's parent has already been
                                                           * deleted. */
    struct TkWindow *nextPtr;  /* Next higher sibling (in stacking order)
                          * in list of children with same parent.  NULL
                                                   * means end of list. */
    TkMainInfo *mainPtr;            /* Information shared by all windows
                                      * associated with a particular main
                                     * window.  NULL means this window is
                                     * a rogue that isn't associated with
                                      * any application (at present, this
                                     * only happens for the dummy windows
                                    * used for "send" communication).  */
    /*
     * Name and type information for the window:
     */
    char *pathName;                 /* Path name of window (concatenation
                                   * of all names between this window and
                                    * its top-level ancestor).  This is a
                                               * pointer into an entry in
                                   * mainPtr->nameTable.  NULL means that
                                      * the window hasn't been completely
                                                        * created yet. */
    Tk_Uid nameUid;              /* Name of the window within its parent
                                          * (unique within the parent). */
    Tk_Uid classUid;          /* Class of the window.  NULL means window
                                      * hasn't been given a class yet. */
    /*
     * Geometry and other attributes of window.  This information
     * may not be updated on the server immediately;  stuff that
     * hasn't been reflected in the server yet is called "dirty".
     * At present, information can be dirty only if the window
     * hasn't yet been created.
     */

    XWindowChanges changes;             /* Geometry and other info about
                                                     * window. */
    unsigned int dirtyChanges;      /* Bits indicate fields of "changes"
                                             * that are dirty. */
    XSetWindowAttributes atts;       /* Current attributes of window. */
    unsigned long dirtyAtts;           /* Bits indicate fields of "atts"
                                             * that are dirty. */

    unsigned int flags;           /* Various flag values:  these are all
                               * defined in tk.h (confusing, but they're
                               * needed there for some query macros). */
    /*
     * Information kept by the event manager (tkEvent.c):
     */

    TkEventHandler *handlerList;/* First in list of event handlers
                                          * declared for this window, or
                                                  * NULL if none. */
#ifdef TK_USE_INPUT_METHODS
    XIC inputContext;           /* Input context (for input methods). */
#endif /* TK_USE_INPUT_METHODS */

    /*
     * Information used for event bindings (see "bind" and "bindtags"
     * commands in tkCmds.c):
     */

    ClientData *tagPtr;     /* Points to array of tags used for bindings
                                * on this window.  Each tag is a Tk_Uid.
                                    * Malloc'ed.  NULL means no tags. */
    int numTags;                        /* Number of tags at *tagPtr. */

    /*
     * Information used by tkOption.c to manage options for the
     * window.
     */

    int optionLevel;                /* -1 means no option information is
                                     * currently cached for this window.
                                     * Otherwise this gives the level in
                                     * the option stack at which info is
                                                     * cached. */
    /*
     * Information used by tkSelect.c to manage the selection.
     */

    struct TkSelHandler *selHandlerList;
                                        /* First in list of handlers for
                                    * returning the selection in various
                                                      * forms. */

    /*
```

```
     * Information used by tkGeometry.c for geometry management.
     */

    Tk_GeomMgr *geomMgrPtr;    /* Information about geometry manager for
                                                  * this window. */
    ClientData geomData; /* Argument for geometry manager procedures. */
    int reqWidth, reqHeight;            /* Arguments from last call to
                                         * Tk_GeometryRequest, or 0's if
                                        * Tk_GeometryRequest hasn't been
                                                            * called. */
    int internalBorderWidth;       /* Width of internal border of window
                               * (0 means no internal border).  Geometry
                           * managers should not normally place children
                                              * on top of the border. */

    /*
     * Information maintained by tkWm.c for window manager communication.
     */

    struct TkWmInfo *wmInfoPtr;       /* For top-level windows (and also
                                   * for special Unix menubar and wrapper
                                     * windows), points to structure with
                                     * wm-related info (see tkWm.c).  For
                                        * other windows, this is NULL. */

    /*
     * Information used by widget classes.
     */

    TkClassProcs *classProcsPtr;
    ClientData instanceData;

    /*
     * Platform specific information private to each port.
     */
    struct TkWindowPrivate *privatePtr;

}TkWindow;
```

APPENDIX E

GLOSSARY

appliance - Runs applications and is a visual interface between the user and the network. There are several classes of user appliances—the desktop workstation, laptop, palmtop, pen-based computer, Personal Digital Assistant (PDA), and pager.

application layer - Establishes communications with other users and provides services such as file transfer and electronic mail to the end users of the network.

application process - An entity, either human or software, that uses the services offered by the application layer of the OSI reference model.

attribute - A property or characteristic.

CC/PP - Composite Capabilities/Preferences Profiles. A way to specify what exactly a user agent (web browser) is capable of doing.

application software - Accomplishes the functions users require, such as database access, electronic mail, and menu prompts. Therefore, application software directly satisfies network requirements, particularly user requirements.

bandwidth - Specifies the amount of the frequency spectrum that is usable for data transfer. In other words, it identifies the maximum data rate a signal can attain on the medium without encountering significant attenuation (loss of power).

baud rate - The number of pulses of a signal that occur in one second. Thus, baud rate is the speed the digital signal pulses travel.

bit rate - The transmission rate of binary symbols ("0" and "1"). Bit rate is equal to the total number of bits transmitted in one second.

Broadband - A signal that has undergone a shift in frequency. Normally with LANs, a broadband signal is analog.

connectivity - A path for communications signals to flow through. Connectivity exists between a pair of nodes if the destination node can correctly receive data from the source node at a specified minimum data rate.

Garbage collection (GC) - The automatic recycling of dynamically allocated memory. Garbage collection is performed by a garbage collector which recycles memory that it can prove will never be used again. Systems and languages which use garbage collection can be described as garbage-collected. Also known as automatic memory management

Global Positioning System (GPS) - A worldwide, satellite-based radio navigation system providing three-dimensional position, velocity and time information to users having GPS receivers anywhere on or near the surface of the Earth.

GSM - Global System for Mobile Communications. A second-generation digital cellular radio standard developed in Europe but widely adopted around the world.

Handheld Markup Language (HDML). A markup language optimized for use in wireless hand held devices.

Hypertext Markup Language (HTML)- A standard used on the Internet World Wide Web for defining hypertext links between documents.

integration testing- Type of testing that verifies the interfaces between network components as the components are installed. The installation crew should integrate components into the network one-by-one and perform integration testing when necessary to ensure proper gradual integration of components.

middleware- An intermediate software component located on the wired network between the wireless appliance and the application or data residing on the wired network. Middleware provides appropriate interfaces between the appliance and the host application or server database.

mobility- Ability to continually move from one location to another.

narrowband system- A wireless system that uses dedicated frequencies assigned by the FCC licenses. The advantage of narrowband systems is that if interference occurs, the FCC will intervene and issue an order for the interfering source to cease operations. This is especially important when operating wireless MANs in areas having a great deal of other operating radio-based systems.

Open Database Connectivity (ODBC)- A standard database interface enabling interoperability between application software and multi-vendor ODBC-compliant databases.

PCS- See Personal Communications Services.

performance modeling- The use of simulation software to predict network behavior, allowing you to perform capacity planning. Simulation allows you to model the network and impose varying levels of utilization to observe the effects. Performance Monitoring Addresses performance of a network during normal operations. Performance monitoring includes real-time monitoring, where metrics are collected and compared against thresholds that can set off alarms; recent-past monitoring, where metrics are collected and analyzed for trends that may lead to performance problems; and historical data analysis, where metrics are collected and stored for later analysis.

Personal Communications Services (PCS)- A spectrum allocation located at 1.9 GHz, a new wireless communications technology offering wireless access to the World Wide Web, wireless e-mail, wireless voice mail, and cellular telephone service.

portability- Defines network connectivity that can be easily established, used, then dismantled.

SQL- See Structured Query Language.

Structured Query Language (SQL)- An international standard for defining and accessing relational databases.

telecommuting- The concept of electronically stretching an office to a person's home.

transceiver- A device for transmitting and receiving packets between the computer and the medium.

WAP - Wireless Access Protocol is the technology that links wireless devices (mobile phones, pages, PDA's, etc) to the Internet. WAP provides the capability to translate information downloaded from the Internet into a format that mobile devices can understand.

WML - Wireless Markup Language. Relatively new (1999) programming language similar to HTML, but with less functionality and a much stricter format. Supports limited graphics (monochrome bitmaps but no animated GIF's or streaming video).

WMLScript - The wireless version of JavaScript that lets programmers add functionality to their WML pages. Currently, WMLScript must be saved in a different file than plain WML, so a single card may have several files associated with it.

XML - Extensible Markup Language. A markup language that can be extended by
defining new data types.

References

Architecture

B. Appleton. Patterns and Software: Essential Concepts and Terminology. 5 May 2001 <http://www.enteract.com/~bradapp/docs/patterns-intro.html>.

L. Bass, P. Clements, and R. Kazman. Software Architecture in Practice. Addison-Wesley, 1997.

D. Connolly. Issues in the Development of Distributed Hypermedia Applications. 13 Oct 1999 <http://www.w3.org/OOP/HyperMediaDev>.

Jazayeri, Ran, and Van der Linden. Software Architecture for Product Families: Principles and Practice. Addison-Wesley, 2000.

P. Hoshka. The Architecture Domain. 20 Nov 2000 <http://www.w3.org/OOP/HyperMediaDev>.

J. Noble and C. Weir. Small Memory Software, Patterns For Systems With Limited Memory. Addision-Wesley, 2001.

Nokia. Mobile Internet Technical Architecture Overview. White Paper, Nokia Corp. 2000 <http://www.nokia.com/press/background/pdf/mita.pdf>.

S. Schach. Classical And Object-Oriented Software Engineering. WCB/McGraw-Hill, 1999.

Component-Based Software

S. Clemens. Component Software, Beyond Object-Oriented Programming. Addison-Wesley, 1999.

Object Services and Consulting, Inc. Component Software Glossary. 12 Jan 2001 <http://www.objs.com/survey/ComponentwareGlossary.htm>.

D. Sprott and L. Wilkes. Component Based Development, Using Componentised Software. White Paper, The Forum for Component Based Development and Integration, May 1999 <http://www.cbdiforum.com/report.php3>.

HTML, CHTML, XHTML

M. Altheim and S. McCarron,. XHTML 1.1 - Module-based XHTML. 31 May 2001 <http://www.w3.org/TR/2001/REC-xhtml11-20010531>

M. Claben. XHTML Tag Reference. 19 Mar 2001
<http://www.webreference.com/xml/reference/xhtml.html>.

Compact HTML For Information Appliances. Web Review. 21 Aug 1998
<http://www.webreview.com/1998/08_21/webauthors/08_21_98_1.shtml>.

Compact HTML Tag List. Home.Mono.Com. 21 Apr 2001
<http://home.monyo.com/htmllint/tagslist.cgi?HTMLVersion=Compact-HTML>.

T. Kamada. Compact HTML For Small Information Appliances. 9 Feb 1998
<http://www.w3.org/TR/1998/NOTE-compactHTML-19980209/>.

J. Kyrnin. HTML Tag Library. 28 Apr 2001
<http://html.about.com/compute/html/library/tags/bl_index.htm>.

M. Nic. XHTML Basic Reference, With Examples. 2 May 2001
<http://zvon.org/xxl/xhtmlBasicReference/Output/index.html>

Nokia. Advantages of XHTML for Wireless Data. White Paper, Nokia Corp. 2001
<http://www.nokia.com/press/background/pdf/mar011.pdf>.

Refsnes Data. The Complete XHTML 1.0 Reference. 2 May 2001
<http://www.w3schools.com/xhtml/xhtml_reference.asp>.

Java

G. Cornell and C. Horstmann. Core Java 2. Volumes I and II. Sun Microsystems, 2000.

B. Day. "Develop Wireless Applications Using J2ME." JavaWorld, Feb 25 2001
<http://www.javaworld.com/javaworld/javaone00/j1-00-j2me.html>.

T. Ekman and A. Nilsson. Deterministic Java in Tiny Embedded Systems. Lund University, Sweden, White Paper, 2001
<http://www.artes.uu.se/events/gsconf01/papers/article.pdf>.

M. Hardee. Why Wireless Needs Java Technology. 10 Jul 2000
<http://java.sun.com/features/2000/07/wireless.html>.

T. Lindholm and F. Yellin. The Java™ Virtual Machine Specification. 18 Feb 2001
<http://java.sun.com/docs/books/vmspec/html/VMSpecTOC.doc.html>.

MIT Telemedia, Networks And Systems Group. Package java.lang. 18 Feb 2001
<http://tns-www.lcs.mit.edu/manuals/java-1.1.1/api/Package-java.lang.html>.

Sun Microsystems. J2ME Technology For Creating Mobile Devices. White Paper, Sun Microsystems, Inc., May 19, 2000 <http://java.sun.com/products/cldc/wp/KVMwp.pdf>.

J. White, “Big Plans For J2ME.” JavaPro, Vol 5, No. 5, May 2001, pp. 20-32.

Memory

A. Clem. How Memory Works. 15 May 2001 <http://xtronics.com/memory/how_memory-works.htm>.

EventHelix. Byte Alignment and Ordering. 15 May 2001 <http://www.eventhelix.com/RealtimeMantra/ByteAlignmentAndOrdering.htm>.

Kingston Technology. What is Memory? 15 May 2001 <http://www.kingston.com/tools/umg/newumg01a.asp>.

C. Levin. “Mobile Memory”. PC Magazine Online. Dec 19, 1995 <http://www.zdnet.com/pcmag/issues/1422/pcm00016.htm>.

J. Newmarch. System Software Memory Overview. Nov 19, 1997 <http://jan.netcomp.monash.edu.au/ssw/memory/overview.html>.

Symbian, Ltd. Memory Management in Quartz v6.0. Technical Paper, Symbian, Ltd. 2000 <http://www.symbiandevnet.com/techlib/techcomms/techpapers/papers/v6/quartz/>.

J. Tyson. How Memory Works. 8 May, 2001 <http://www.howstuffworks.com/computer-memory.htm>.

Xanalys. The Memory Management Reference. 8 May 2001 <http://www.xanalys.com/software_tools/mm/>.

Mobile

D. Connolly. Mobile Code. 9 Dec 1996 <http://www.w3.org/MobileCode/>.

Ericson, Inc. Mobile Applications Initiative. 20 May 2000 <http://www.mobileapplicationsinitiative.com>.

Mobile Computing Application Development Tools and Strategies. Mobile Info. 11 Apr, 2001 <http://www.mobileinfo.com/application_dev.htm>.

Mobile Computing Software Standards. Managing Change. 12 Jan, 2001 <http://www.managingchange.com/mediums%5Cmobile%5Csoftware.htm>.

Motorola. Wireless Application Development. White Paper, Motorola, Inc. 2000 <http://www.motorola.com/MIMS/MSPG/spin/library_files/wad.pdf>.

3G Newsroom. 4 Apr, 2001 <http://www.3gnewsroom.com/>.

Synchrologic. The Future of Enterprise Mobile Computing. White Paper, Synchrologic, Inc. 2000 < http://www.synchrologic.com>.

Synchrologic. The Future of Enterprise Mobile Computing. White Paper, Synchrologic, Inc. 2000 < http://www.synchrologic.com>.

Wireless Telecom Glossary. Cellular Network Perspectives. 3 Jan, 2001 <http://www.cnp-wireless.com/glossary.html>

Oracle

GSI. Oracle Lite Object Kernel API Reference, Release 4.0. GSI. 12 Apr, 2001 <http://www-wnt.gsi.de/oragsidoc/doc_8i_lite/olokref/html/apitoc.htm>.

Oracle Mobile. Oracle Corporation. 10 Apr, 2001 <http://www.oracle.com/mobile/>.

T. Dyck, “Oracle8i Lite Takes Fresh Web Apps on the Road”, eWEEK October 11, 1999 <http://www.zdnet.com/products/stories/reviews/0,4161,2349507,00.html>.

Tcl

S. Ball, Web Tcl Complete, McGraw-Hill, 1999.

European Southern Observatory, Data Management Division. Tcl and C++ Utilities Package Programer’s Guide. 1998 < http://archive.eso.org/skycat/docs/tclutil/>.

J. Ousterhout, Tcl and the Tk Toolkit, Addison-Wesley. 1994.

J. Radajewski. The Beowulf Monitoring System. 2 Jun 2001. <http://www.sci.usq.edu.au/staff/jacek/bWatch/>.

SourceForge. The Tiny Tcl Home Page. 13 Jun 2001 <http://tinytcl.sourceforge.net>.

Tcl Developer Exchange. The Tcl Developer Site. 3 Mar 2001 <http://www.scriptics.com>.

The Tcl Newsgroup <comp.lang.tcl>.

K. Waclena. Introduction To The Tcl Programming Language. 3 Mar 2001 <http://www.lib.uchicago.edu/keith/courses/tcl/>.

B. Welch, Practical Programming in Tcl and Tk, Prentice Hall PTR 3rd ed., 2000.

Miscellaneous

J. Alger. “Software Review - Component Workshop and OODLs” MacTech. Jul 92 <http://www.mactech.com/articles/frameworks/6_4/Component_Workshop_Alger.html>.

CC/PP. 30 Mar 2001 <http://www.ccpp.org>.

Client and Server Side Scripting. 24 Apr 2001. <http://www2.fyrisskolan.uppsala.se>.