Symbian Technical Paper

Designing for Quartz

Martin Tasker, Head of Technical Communications, Symbian Ltd

Summary

Symbian's Quartz design, for a tablet communicator, presents many new opportunities for application development. Successful applications must be designed both for the device, and for its end-users.

This paper covers the design decisions underlying the new Quartz GUI, and shows how the standard guidelines are implemented by the built-in application suite. Finally the author considers how to migrate two of his own applications, the Solo Ships and Battleships examples from *Professional Symbian Programming*.

This paper is based on material prepared for the Wrox Professional Wireless Developers Conference, Amsterdam, July 2000.

Contents

Introduction

Contents

Symbian's previous product was EPOC Release 5. This is the operating system behind the Psion Series 5mx, Psion Series 7, Psion netBook, Ericsson MC218, and Psion Revo. These devices all sport a touch screen of similar size, and a compact but full alphanumeric keyboard.

Symbian's software platform was designed from the beginning to allow the GUI to be replaced so that new products, in quite different categories, could be created. Symbian has already created two reference designs for new devices, and is working on a third. Each design shares the same kernel, the same communications, data management and graphics APIs. At the higher level, however, the GUI for a reference design is tailored for the hardware targetted by that reference design.

Three reference designs are available or being created:

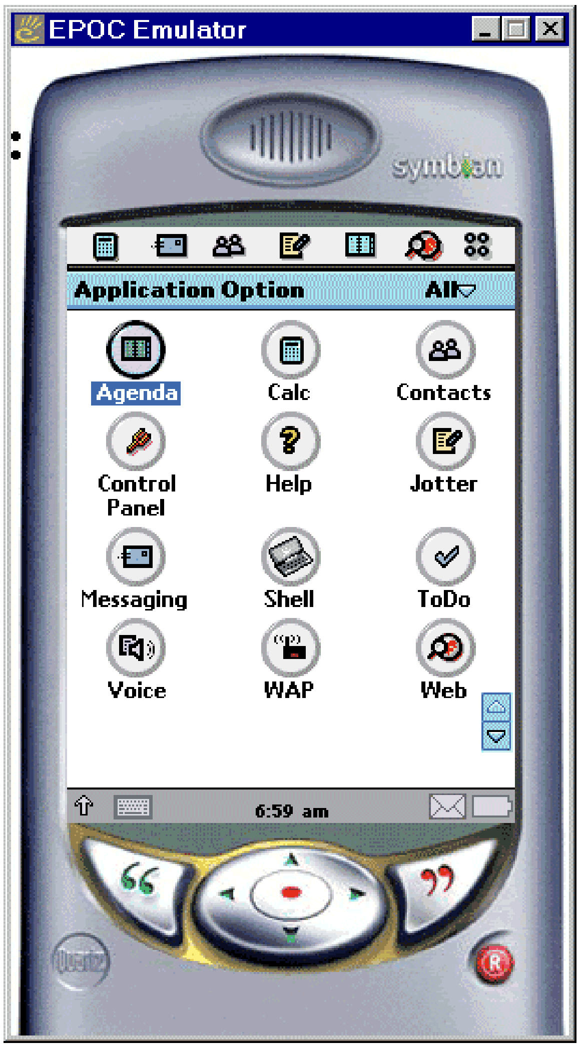
- Quartz, for communicators with tablet form factor and no keyboard
- Crystal, for communicators with keyboard
- Pearl, for smartphones

There is about 80% commonality in the APIs shared by these designs. It's only at the highest and most visible level, namely the GUI, that one design differs from another.

In this paper, I'll focus on Quartz. I'll explain the motivations behind hardware and interaction design choices, and go into detail about the implications for application design.

Here's a picture, showing Symbian's Quartz emulator after launch:

along with other pictures in the paper, this is based in a February 2000 version of Quartz, and needs updating



Basic parameters

Quartz is Symbian's tablet communicator reference design.

It is designed for hardware as follows:

- a quarter-VGA portrait screen — 240 pixels wide, 320 pixels high
- pen operation
- no keyboard
- hardware keys for up and down, and a single, large, Confirm button which works like enter on a PC GUI. Hardware keys for right and left are optional.
- built-in phone, with speaker, microphone, and probably two additional keys (yes/no, dial/hangup or whatever)

You can see all these features on the screenshot above.

The standard pixel size of Quartz hardware is expected to be in range 0.22-0.24mm square, giving a standard screen size of 5.28-5.76cm x 7.04-7.68 cm. EPOC graphics software, on both real Quartz hardware and the Quartz emulator, *always* assume a pixel size of 16/1440inch, or approximately 0.28mm, for the purpose of converting real-world measurements into pixel measurements. On EPOC R5, the standard pixel size was 0.21mm. Graphics software on the emulator and real EPOC R5 devices used this value.

Symbian, manufacturers and developers

It’s important to realise that Quartz is not just a tablet PDA. It’s a *communicator*, which includes a first-class mobile phone, along with everything that a phone implies, namely

- speaker, microphone, antenna
- SIM card
- network subscription
- devices are made by phone manufacturers

At the time of writing, announced Quartz licensees include Ericsson, a phone manufacturer, Sanyo, a consumer electronics company, and Psion+Motorola, who are working jointly on a Quartz-based product. Psion, historically a PDA company, is working with Motorola, historically a phone, semiconductor and communications company, to bring this product to market. Other Quartz licensees may be expected.

Quartz devices will be designed so that they can be used as a phone while browsing or entering information using the screen and pen. This could be done using a typical speakerphone approach, or a separate headset using wired or Bluetooth connection to the device.

Quartz is a reference design produced by Symbian. It defines a family of devices. Each device is produced by a manufacturer. Manufacturers can and will differentiate their devices in various ways, so they can compete effectively with each other.

By doing business this way, Symbian can ensure that real innovation is maintained in the industry.

Within certain parameters, the manufacturer can do what they like, including

- the entire hardware design, for optimum combination of cost, size, weight, battery life, display clarity and usability
- the phone, including speaker, microphone, hands-free kit, additional buttons etc
- the *phone app*, ie the application program which allows an end-user to make and receive voice calls, and to control their phone and its interface with the network
- the appearance of some aspects of the Quartz GUI
- the indicators on the status bar at the bottom of the Quartz display: such indicators typically include battery power and signal strength but may include additional indicators if the manufacturer has built in additional hardware facilities
- provide additional hardware and APIs, so that the manufacturer could add extra devices — anything from a GPS receiver to a barcode scanner — and make them available to applications.

As a developer, you can be sure that all APIs included in a Symbian reference design are contained in all products built using that reference design. That means you can write one program which will work on all Quartz devices.

However, since individual Quartz products may contain additional APIs, exposed by the manufacturer in SDK supplements, you can also use these product-specific APIs to produce a product-specific application program. Whether you want to do this will depend very much on the products and your business. Symbian’s business model gives you the choice.

For the rest of this paper, however, I’ll be talking about the Symbian Quartz reference design. I’ll avoid manufacturer-specific issues, though I’ll mention the phone app a few times, since this is something that Symbian *requires* the manufacturer to provide.

End-user

Quartz is not only a different size and shape from the devices supported by EPOC R5: it is aimed at different types of end-user.

Compared with EPOC R5, Quartz is aimed at significantly less technical end-users.

We're moving towards the type of end-user who buys a mobile phone, rather than the type who is really comfortable with a Windows-based PC. You are probably so used to the PC that you have forgotten how intimidating PCs are for average end-users — unless you've recently done some technical support for your parents or grandparents! EPOC R5 used an essentially PC-type paradigm, albeit with improvements to make things easier. Quartz goes much further, as we'll see.

Quartz is like paper: what you see is what you have.

Any data you enter on a Quartz screen is saved when you move away from the view on which you're entering the data — however you move away. There is no PC-like "save data: yes/no/cancel" dialog when you close a program: data is always saved. Data is saved not only when you close a program, but when you switch to a running program. So what you see is what you have, and what you last saw is what you still have — always. This simplifies the user interface, increases the level of user confidence, and provides better insurance against data loss.

Quartz is optimized for browsing and single-tap pen operation.

Without a keyboard, it's very easy to use the pen to browse around information on a Quartz device. Just as web browsers use only single-click navigation, Quartz uses single tap, with the pen, to open up links, programs and items. Windows' double-click is not used, neither is EPOC R5's select-and-open.

Most applications are structured as *list views*, in which all data in an application can be browsed by navigating around a list. Items from the list can be opened by tapping on them with the pen.

Quartz is not browse-*only*. Data can be entered using convenient pointer-friendly controls for dates, numbers and so on. Text can be entered using handwriting recognition or a virtual keyboard. Symbian supplies CIC's Jot handwriting recognition engine with the emulator in the SDK. CIC Jot is competent and comfortable. CIC Jot is not however a part of the reference design: the Quartz device manufacturer must license CIC Jot, or another handwriting recognition engine, for their own device.

Data may also be entered by synchronization with desktop-based data, or by receiving data from other devices using Symbian's software platform, including the Quartz and other reference designs.

Quartz is task oriented, not application-oriented.

The most common task on a mobile phone is to dial a number. In an application-oriented system, you would start with the phone app, look through the contacts database using some shared UI to select a contact, and then dial that person.

In a task-oriented system, you start with the contacts app, select the person you want to talk to, and then tap their phone number. Control then transfers to the phone app so you can dial them.

Likewise, to send an e-mail, you start in the contacts app, select your contact, and then tap their e-mail address. Control transfers to the e-mail app, with a blank message to the contact's e-mail address.

Quartz *does* have applications, but it is structured to make it easy to transfer between them when needed.

Technical necessities are hidden from users — there is no file system, and no task list.

There is a file system, and Quartz implements sophisticated multi-tasking. But these technicalities are hidden from end-users. To Windows users — or even to EPOC R5 users — this may seem revolutionary. But today's mobile phones are already sophisticated multi-tasking systems, and they maintain significant databases, such as the phone book or text message log. Phones do all this without exposing their file system or task list to the end-user.

End-users would not expect to be able to see a task list: they simply need to be able to switch between applications reliably. They do not expect to see the file system: they simply need to get at their data reliably through the application that manages it. All the evidence from Windows, and even EPOC R5, suggests that the file system and task list are intimidating to the great majority of users.

Quartz is much more sophisticated than today's mobile phones, but it delivers this sophistication in a deceptively simple-looking package. Removing the technicalities from the UI, single-tap navigation, and task-oriented application switching, are all a part of this simplification which delivers a much more attractive end-user experience. Now that we've seen the basic principles, let's get down to the details and see how the GUI works.

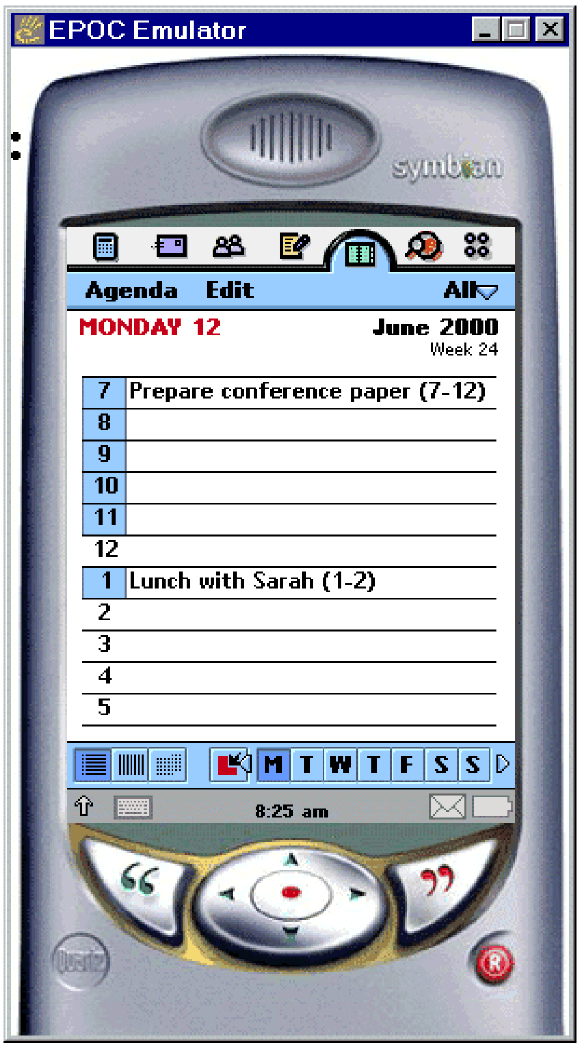
Elements of the GUI

Contents

Now let's look at the GUI in detail, to see how the principles described above are worked out in practice. I'll start with the basic screen layout. Then I'll take you through a couple of applications to get the general idea of how this works in practice. Then, I'll describe the style guidelines for detailed elements such as views, menus, dialogs and so on. Finally I'll go through the remaining applications to show how the big ideas work out in practice, and to highlight a few more detailed design issues.

Screen areas

Here's a view from the Agenda:



MONDAY 12 June 2000
Week 24

7 Prepare conference paper (7-12)
8
9
10
11
12
1 Lunch with Sarah (1-2)
2
3
4
5

You can see that the screen is divided into five areas, top to bottom:

Area name	**Description**
application picker	Controls switching between applications.
	The rightmost icon always invokes the shell (in Quartz, the shell is called the *app launcher*), which allows any application to be launched. The first six icons are customizable by the end-user: the manufacturer may also set the defaults.
	This window is owned by the shell. The same window is used, whatever application is running.
menu bar	Contains the application's menus.
	Each application has its own menu bar. The menu bar may change from view to view within an application. The menu bar is always showing. The leftmost item contains the application's name: it is the quickest way to identify which application is currently showing.
	We'll cover standards for menus below.
client area	The main display area for each application view.
	Also known as *application space*.
button bar	A bar containing convenient navigation controls within the current view (eg the day selectors on the agenda day view's button bar), for switching between peer-level views (eg day, week or month), or containing buttons for other purposes.
	An application view may or may not have a button bar.
status bar	Displays device status information such as battery charge and signal strength, time, and text entry status.
	This window is owned by the shell. Its content may be customized by the manufacturer.

The screen is 240 pixels wide. Each bar — or the client area — is therefore always precisely 240 pixels wide. There is flexibility in the height of the various bars, and more bars — temporary or permanent — may be introduced in the future. So you should make the following assumption about the client area:

The client area is always 240 pixels wide. It is typically between 230-260 pixels deep. This is room for about 13-14 rows of text-based information.

In the Agenda day view above, the client area + button bar are 256 pixels high altogether. The button bar is 25 pixels high, leaving 231 pixels for the client area.

You can design your views for pixel precision in the horizontal dimension, but you should leave some flexibility in the vertical dimension.

One very good way to leave flexibility is to code your view as a list — like the day view above, or like many other list views in Quartz. On the Agenda day view, there are 11 time slots, a two-line title and a button bar: this adds up to 14 rows of information in the client area and button bar combined.

Applications and views

Each application may have more than one view. Different views may be used for navigating through application data, browsing it, showing it in different ways, or editing it. Each application has its own views, and its own ways to navigate around within and between views.

Only one view may be displayed at any one time. The view being displayed is said to be the *active view*.

An application may have multiple views. The view you get when you open the application is its *base view*.

In keeping with Quartz’s paper metaphor, any views that can change data must save changes when the view is deactivated — whether deactivation results from a switch to another view in the same application, or from a switch to another application.

In keeping with Quartz’s task-oriented approach, you can switch directly from a view in one application to a view in another. The underlying API for this is called *direct navigation links* or DNLs. Each Quartz application publishes a list of DNL targets that other applications can switch to. For instance, the phone app allows you to invoke its base view, with a contact ID and phone number that the phone app should dial, if it can, to make a voice call.

In keeping with Quartz’s browse-mostly usage pattern, most applications use some kind of *list view* as their base view, displaying all the items managed by that application — say, all contacts. You can navigate around the list view, and then open an item for editing or browsing in a *detail view*.

I’ll use the views from the standard Quartz application suite to

- show what applications are provided with Quartz
- list the available DNL targets, which are available to you as a developer
- demonstrate how views work in practice
- provide inspiration for your own application designs
- motivate some other details of the Quartz GUI including standard menu structure

You’ll soon see that, in a real sense, the art of designing a Quartz application is the art of designing its view structure. Once you have the structure, you can fine-tune each view.

Real applications: personal information management

Section Contents

Let's start the tour of standard Quartz applications with three PIM applications

Agenda	Your schedule. Displays day, week and month views. Supports timed and untimed events, and multi-day events.
ToDo	To-do list. Displays items with priorities and whether or not they are done.
Contacts	Contacts database. Stores contacts and information including phone numbers and e-mail addresses, which are used to link to other Quartz applications using DNLs for common tasks such as making a voice call, sending an e-mail or text message, or sending a fax.

Agenda

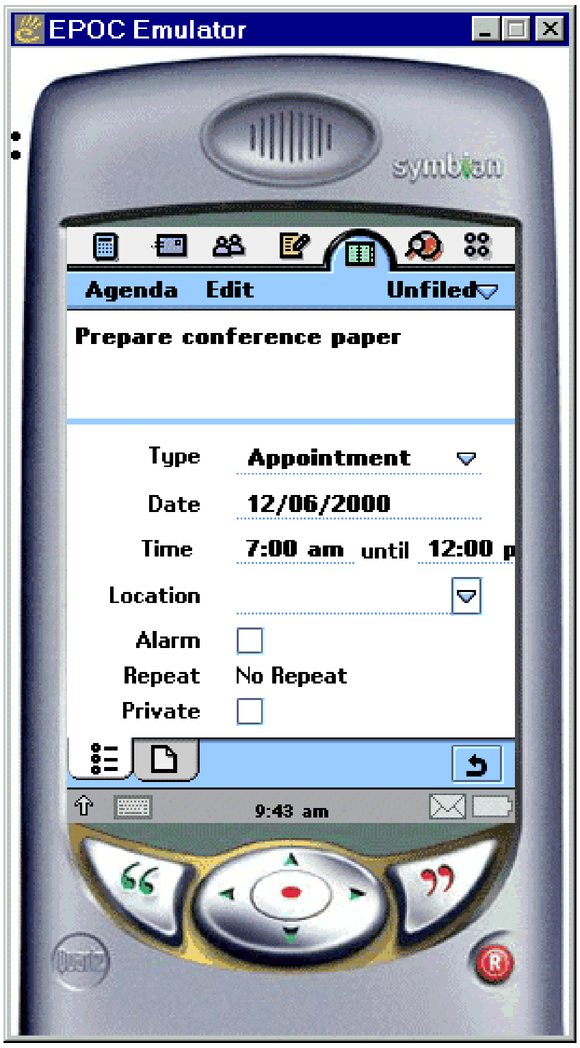
If you're familiar with the EPOC R5 application suite, you'll recognize immediately an important design decision. In EPOC R5, schedule and to-do items were both managed by the Agenda application. The items were stored in a single database, which through the agenda server could be accessed by multiple clients. In Quartz, the same database is used, but it is accessed by two quite distinct applications.

As we have seen, the base view for Agenda is the day view. The button bar on the day view allows you to select week or month views.

Thus it appears to the end-user that Agenda has three base views — day, week and month — all at peer level. There is no nesting between these views.

However, you have to select one of these as the application's base view, which is invoked when the application is started from the app launcher. It happens to be the day view.

If you single-tap on an item in the day view, it immediately opens a detail view which you can browse the item or edit it:



Prepare conference paper

Type Appointment
Date 12/06/2000
Time 7:00 am until 12:00 p
Location
Alarm
Repeat No Repeat
Private

Notice a couple of interesting things here:

- compared with EPOC R5, there are fewer options. For instance, you can't say that an item is tentative, you can't select which alarm sound to use, and although you can attach a memo to an item (see below) you can't attach a picture or anything else.

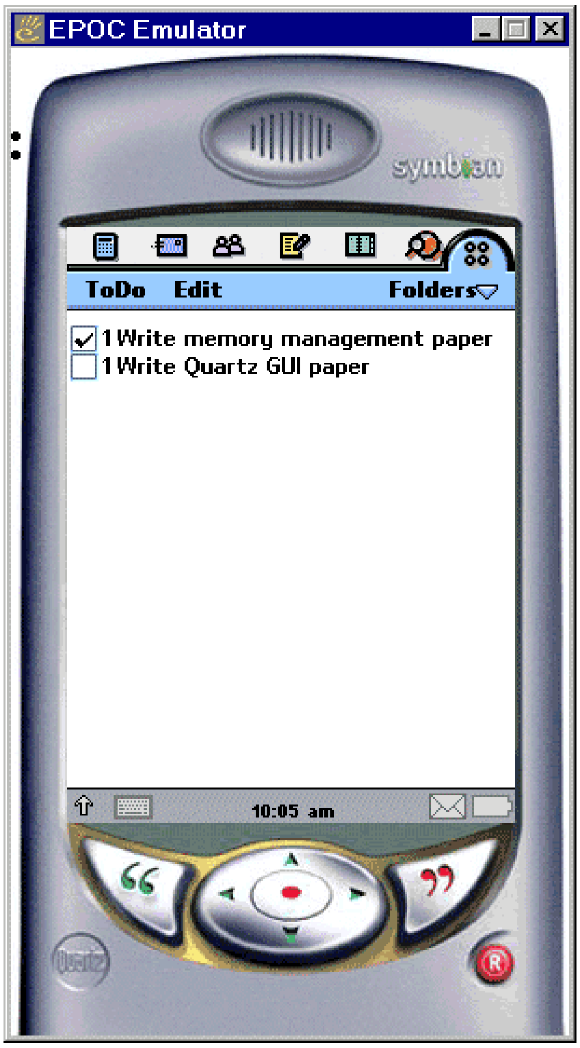
- in some cases, multiple items are placed on a single line, to conserve vertical space — for instance, the start and end times
- the button bar is used for two purposes: tabs for switching between basic information and notes, and buttons such as the curly arrow indicating that you've finished editing or browsing the entry
- one of the most useful features of the Agenda in EPOC R5 (and in its predecessor system, SIBO) was to attach memos to agenda items. You could use a memo to take notes of a meeting. Quartz supports this feature by the notes tab on the button bar. Initially, the notes text is blank. The notes tab is always displayed, whether or not any notes has been entered.
- to quit the detail view and return to the list view (day, week or month), you tap the curly arrow on the button bar, or press the Confirm button. In keeping with Quartz's paper metaphor, any changes you made between opening the item and closing it are saved.

The detail view also opens when you create a new item — either by using `New` from the menus, or by tapping an empty slot on the day, week or month view.

ToDo

Like Agenda, ToDo manipulates a database. ToDo has a single base view, which lists all the items to be done. As usual, if you single-tap an item, it opens, allowing you to edit the details of the item.

One very common operation with to-do items is to check an item off when it has been done. To support this, ToDo's list view includes a check box against every item. If you tap the check box, it checks the item (or unchecks it, if it was already checked). If you tap anywhere else on the line, the item opens as normal.

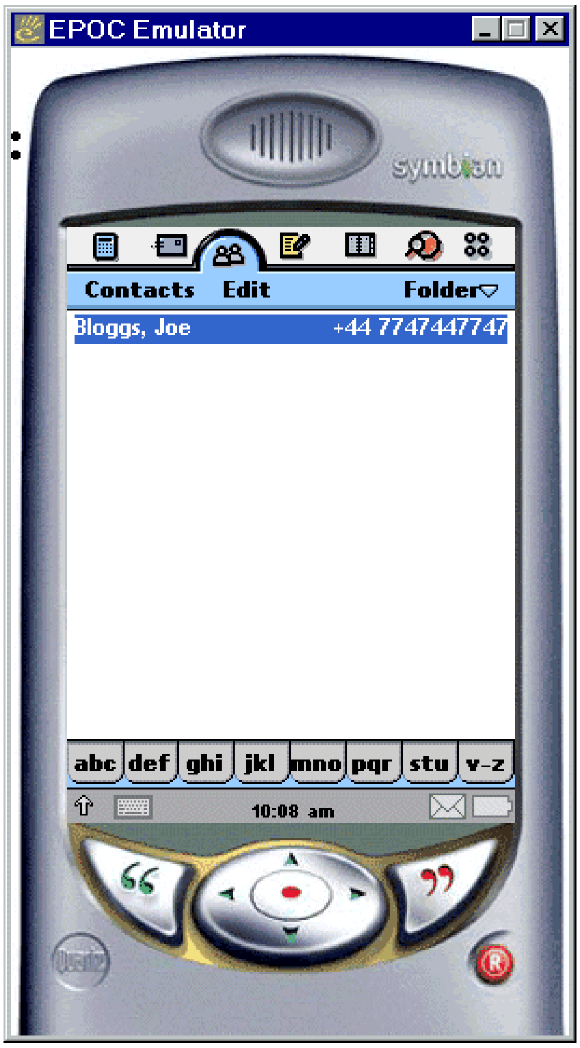


✓ 1 Write memory management paper
☐ 1 Write Quartz GUI paper

Contacts

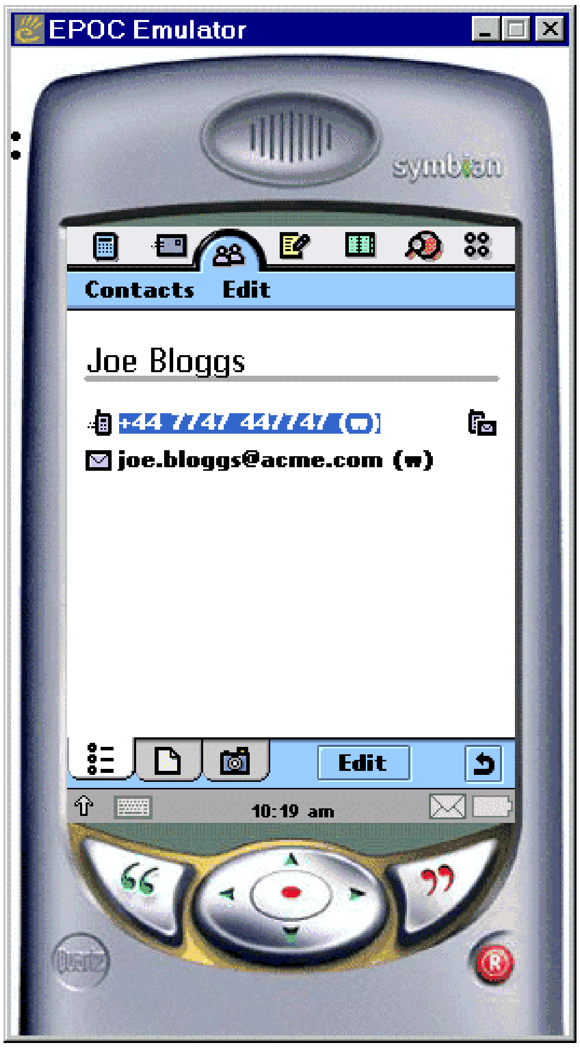
Like Agenda, Contacts manipulates a database and allows items to be opened for editing. In a communicator device, Contacts is a key application, since communication is typically with a contact whose details are in the contacts database.

Contacts' list view includes tabs for selecting items alphabetically:



An interesting touch here is the button bar, which contains a set of alphabetical tabs, which you can use to rapidly select part of the list containing the contact you're looking for.

When you tap an item, you open it. Unlike Agenda, which has a single view for both browsing and editing, a Contact is opened initially for browsing only:



This shows all the contact information available for the contact — in this case, just work mobile number, and work e-mail.

If you tap on the e-mail address, Quartz will open the messaging app and open an e-mail to that address.

If you tap on the phone number, Quartz will likewise open the phone app and, if possible, dial a voice call. If you wanted to send a text message, you must tap on the message icon to the right of the mobile phone number: the messaging app will be used instead of the phone app.

As usual, the button bar provides tabs for additional information, and a curly arrow to return you to the list view. For a contact, the tabs allow you to attach notes, and a picture — perhaps a photo.

An `Edit` button takes you to a detail view which allows you to edit the contact's details. This distinction between edit and browse wasn't necessary with Agenda or ToDo. But with Contacts, there are so many possible fields that it is impracticable to display them all at once. Most contacts would use only a few fields anyway, and you only want to browse the fields that are non-blank. The edit view, however, allows you to enter data into any of the allowable fields.

Standard guidelines

Section Contents

Now that we have seen how some real Quartz applications are designed, we can understand the standard guidelines for designing Quartz applications.

List views

Quartz relies heavily on list views, such as those used to display the schedule, contacts and to-do databases. In a list view, you tap once with the pen to open a detail view of an existing item.

You can use the pen to navigate up and down. A pair of scroll buttons floats above the view if necessary — see the picture of the emulator after launch, above in this paper. Tapping on a scroll button scrolls a whole page — with a pen-driven device, there's little point in scrolling by a single line.

You can also use the up, down and Confirm keys to navigate up and down the list, and to open items.

List navigation can be optimized where necessary: for instance, the tabs in the Contacts database allow you to quickly move to regions of the alphabet.

A single tap selects and opens an item. Devices such as the check box on the left of a to-do item can be used to implement a behaviour other than opening — for instance, checking a to-do item.

The single-tap system has some deep implications:

- the *only* action that can be performed on an item, in the absence of devices such as the check box on to-do items, is to open it
- actions such as delete, cut, copy, or send must therefore be done from the detail view, not the list view
- multiple selection is not possible from the list view

These implications are in keeping with Quartz's browse-mostly usage pattern. Browsing is optimized: certain other functions, as a result, are less optimal than they might have been if a different UI paradigm were used.

Where other operations are obviously very frequently needed, special devices can be used, for example

- the check box to the left of ToDo items
- the SMS message icon to the right of a contact's mobile phone number
- a selection area to the left of a message, to allow multiple selection without opening — it is thought that users will often want to use multiple selection when organizing their messages, for instance, when downloading them from an IMAP server to a local folder

Devices alongside list items should be used when necessary. But don't over-use them: aim to keep the design of list views simple and uncluttered.

Detail views

A detail view displays an individual item, and allows you to edit it.

Quartz's paper paradigm has a deep impact on the design of detail views:

- when the detail view is deactivated — whether by returning to the list view, or by switching to another application — the data currently being displayed is saved
- there is only one exit button: the Confirm button or the curly arrow that you can tap with the pen. When you exit, data is saved.
- because changes are always saved on exit, an `Undo` menu item is provided which (as a minimum level of functionality) restores the data to the state it was in when the detail view was last activated. In more sophisticated detail views, undo may undo a single operation at a time.

The operation of list views — in particular, the fact that they don't allow any operation except open — also affects the design of detail views, because all operations except open must be offered from the detail view instead of the list view. This includes delete, cut, copy and send.

Detail views can use multiple tabs to show different parts of the data — such as the information, notes and picture tabs for a contact.

In some cases, it's inconvenient to provide integreated browse and edit functionality in the detail view. In this case existing items should normally be opened in browse view, and the browse view should provide a button to edit, if possible. New items should clearly be opened in edit view.

Managing large databases

If an application can grow to manage many items, its lists can get too long to manipulate without some help. Several standard approaches are provided by Quartz to help here.

- a find facility allows an application to search through all items, and displays items matching the search. From the find results dialog, you can open individual items.
- a folders facility allows you to associate a folder with each item. You can display either items in `All` folders (the default), or in specific folders. Each application comes with three standard folder names: `Business`, `Personal` and `Unfiled`. New items, by default, are `Unfiled`. End-users can change or delete these names, or add more folders.

You can also use *ad hoc* methods to suit the application data, such as the alphabetical tabs on the Contacts list view.

It's important that these sophisticated features don't intrude on the basic operation when the database size is small. That's why the default behaviour for folders is to create `Unfiled` items and to show items in `All` folders: it means that users don't have to learn about folders, and they don't get in the way.

Quartz's folders can be implemented using the file system — a directory for each folder. But, typically, they are not. Instead, each entry in a database is given an ID, and the IDs are associated with folder names. The folder selection engine then simply checks IDs. The way an application implements folders is a choice which is open to the developer. However, folders do not — and must not — expose the file system directly to end-users. Most users find a full file system intimidating: it *does* intrude on basic operation of the system when you only have simple data, and therefore violates a basic usability guideline.

Standard menu structure

Each Quartz view should have only three top-level menus: the application menu, the `Edit` menu, and the folders menu. The folders menu is optional. Cascaded menus, or more menu panes, are supported in the Quartz GUI, but their use is strongly discouraged.

Since there are so few options, you have the opportunity — indeed, the requirement — to simplify your application design. This requires some hard work, but is immensely liberating for both developer and end-users.

The application menu should always use the application name. Since the menu bar is always visible, the application menu name is *the* quick way to distinguish which application is running.

A Quartz application with a list view / detail view structure should use the following standard menu assignments:

List view:

Application	`Edit`	*Folder*

`New`
`Find`
	`Undo delete`
	----- 	
	`Zoom`
	`Preferences`
	`Help`
		`All`
		`Business`
		`Personal`
		`Unfiled`
		----- 	
		`Edit folders`

Detail view:

Application	`Edit`	*Folder*

`New`	`Undo changes`	`Business`
`Find`	`Cut`	`Personal`
`Send as`	`Copy`	`Unfiled`
`Beam`	`Paste`	-----
-----	-----	`Edit folders`
`Delete` *item type*	`Zoom`	
	`Preferences`	
	`Help`	

You can add application-specific items to these menus also. Note the capitalization convention. Note that `...` ellipses are not used to indicate items that invoke a dialog.

There is no `Close` option. `Close` is not required by Quartz applications, and should not be provided.

When a view is deactivated, it should save its data. When a view is activated, it should restore from what was previously saved. End-users cannot distinguish between application launch and switching to an application — or between switching away from an application and closing it. That leaves Quartz free to close applications when it needs to (which usually means, in order to free up some memory). There is no need for the end-user to close applications explicitly.

It is often useful to add a `Close` option to debug builds, so that as a developer you can close the application quickly when running on the emulator. You don't have to put this on the menus: you canprovide a hotkey such as ctrl+E, as used by EPOC R5 applications.

`New` and `Find` are available in both list and detail view — because they are always useful, and they do not refer to the current item.

`Zoom`, `Preferences` and `Help` are available from both views also. Depending on the applicationn, you may wish to code these items as view-specific or application-wide. Quartz has a standard zoom settings dialog, supporting three levels of zoom — small, medium and large. Use this dialog if possible and interpret it according to the requirements of your application.

It is very easy to *do* things in Quartz, so it's a good idea to make them easy to *undo* also. A typical edit view would provide `Undo changes` to restore the item to its state when the view was last activated. A typical list view would provide `Undo delete` to undo a previous deletion. If there is nothing to undo at present, dim the `Undo` menu item. More sophisticated applications may use more forms of undo.

`Send as` (a message), `Beam` (over infrared), `Delete`, `Cut` and `Copy` are all item-specific, and are therefore available only in the detail view. `Delete` and `Cut` should return to the list view when they have finished. `Paste` pastes into the current item. Note the additional text after `Delete`: `Delete contact` tells the end-user the scope of their delete operation, which is important.

The folder menu on a list view includes `All`, then the currently available folder names, and finally, after a divider, `Edit folders`. The options above the divider select which items are displayed in the list view. `Edit folders` allows you to edit the available folder names, or add new ones, or delete old ones. Usually, however, It does not allow you to assign items to a specific folder. For that, you need the folder menu in the detail view: options include all currently defined folder names and `Edit folders`. There is no `All` option (because an item can't be in all folders at once!) and no divider (because `Edit folders` allows you among other things to assign the item to a folder, which is just what the other options do).

Direct navigation links

We saw above that, if you tap on a contact's e-mail address, Quartz takes you into the Messaging application, and opens an edit view for an e-mail to the contact you came from.

This is an example of Quartz's *direct navigation link* scheme, which is used for task-oriented application suite design.

Some applications use DNLs, others provide DNL targets. An application which uses a DNL must specify

- the application ID which it wants to link to
- the view ID
- a command type
- some command parameters

An application which provides DNL targets must export the view IDs, command types and command parameter formats in a C++ header file. Applications which use these DNL targets must build using that header file. So, an application's targets are fixed at its build time, and an application using DNLs must know all available and relevant targets at its build time. This amounts to a fairly static system: over time, more support for dynamic linkages may be added.

Real applications: finishing the tour

Section Contents

it's time to take up the tour again. I'll proceed quickly through the standard application suite, highlighting the main design issues in each application.

Browsers

The Web and WAP browsers have a simple structure: the base view is a list of bookmarks. Open a bookmark, and the page is displayed.

Contrast conventional web browsers in which the action on opening the browser is to take the user to their home page.

A feature of the Quartz browser is that pages can be cached for off-line reading. A page's cache status is indicated in the list view.

The web browser does not allow `file:` URLs: that would be a back door to the file system.

Words and pictures

The Jotter application doubles as both Word and Sketch in EPOC R5. It manipulates a database of rich text objects — which, in EPOC, may include pictures.

The title given in the list view is the first line of the item's text. The detail view contains two panes — one for notes, which is text only, and one for the sketch, which allows a single bitmap to be edited.

This is less general than the full power of Word, embedded Sketch objects, the file system or the Data application available in EPOC R5. But it's much easier to use, it's an appropriate level of sophistication for a system with no keyboard, it hides the file system, and it's consistent in its approach with Agenda (notes on a detail view pane) and Contacts (notes and sketch on detail view panes).

Messaging

As with EPOC R5's Email application, Messaging supports e-mail, fax and SMS messaging. Multiple e-mail accounts are supported. Only a single SMS account is needed, and a single fax account — because the phone is built into the Quartz device.

Messaging is an important application whose usability will be a key factor in determining the success of Quartz as a platform. Messaging bends the usual interface design rules to provide an optimized user experience:

- instead of a single list view displaying all messaging, the base view is a list of accounts — initially, just fax and SMS. You can also add any number of e-mail accounts.
- folders in Messaging correspond with folders on the IMAP server, and may be nested
- since multiple selection is important when manipulating messages, a check area to the left of each item, in list views, allows

multiple items to be selected

In keeping with Quartz's task orientation, Messaging defines DNL targets to allow messages of any type to be composed, given the appropriate information from the Contacts application.

Simple apps

Voice, Time and Calc are applications which will be familiar to EPOC R5 users.

Voice is a sound recorder. In theory, voice is a perfect candidate for the list view/detail view approach taken by most Quartz applications. However, voice recordings — even with GSM standard compression — occupy a lot of data. In a move which may prove controversial, Symbian's interaction designers have made the list less obvious, in order to deter extended use of the voice recorder. As a result, Voice is a single-view application with, for Quartz, an unusual system for navigating between recordings.

Time is the world time clock. In EPOC R5, Time also included a world map. In Quartz, the world database is included, so you can conveniently change your time settings when travelling. But due the small screen size, no bitmap display of the world map is provided. Also, Quartz only supports three alarms instead of the six provided by the Psion Revo, or the eight provided by other EPOC R5 models. (You can, of course, associate an alarm with any Agenda item — you don't need a slot in Time to do that).

Calc is a very simple, but effective, desktop calculator. Quartz Calc doesn't include the scientific mode available in EPOC R5 Calc.

Neither of these simple applications uses a folder menu. Time has `Time` and `Edit` menus. Calc uses only an `Edit` menu — the only standard Quartz application that doesn't use its own name for the first menu pane.

Phone

Quartz is a communicator — an information-centric device with built-in voice telephony. As such, a built-in phone is integral to any Quartz device.

The hardware and software for driving the phone is a matter for the manufacturer. A GSM phone, for instance, will include RF circuitry, GSM signalling stack, voice coder and decoder, and data stack for SMS, data and fax. In the very near future, the data stack will also support GPRS. SIM (subscriber information module) management must also be included. EPOC provides some high-level APIs into these features, including

- the telephony server, which supports data calls on phones
- the external interface of the phone application, which requires the application to support a DNL target to enable a specified phone number to be dialled — with optional information about the contacts database entry from which that phone number was taken

The manufacturer must provide the relevant implementations for these APIs.

System apps

The Quartz shell is quite different from that of EPOC R5. EPOC R5's shell (also called the System application) enabled you to launch either files or applications, and to launch the control panel.

The Quartz shell, like all other Quartz components, hides the file system. The shell is really an *app launcher*. The view of available applications can be displayed either as icons (as seen in the screenshot at the start of this paper) or as a text-based list. The option to switch between these two modes is not governed by a button bar (like Agenda day, week and month views) but by menu options (`View as icons` vs `View as list`). This reflects the fact that you change only rarely between the two views, and it isn't worth taking up the vertical space required by a button bar.

The Control Panel is a regular Quartz application. The Control Panel displays a list of Control Panel applets: you tap on an entry in the list to open the applet. A Control Panel applet can be a full Quartz application: typically, however, it displays only a settings dialog.

The Control Panel raises an interesting design issue: should settings be controlled from within an application (`Edit > Preferences`) or a control panel applet? Good style would suggest that

- the application be supplied with sensible defaults
- settings which are easily understood and manipulated by the user be located in the application
- settings which are obscure be located in a Control Panel applet
- if there are insufficient obscure settings to justify a Control Panel applet, use application preferences alone

In any case, you should minimize the number of options you offer to your users. Try to auto-adjust for everything. Only surface things as options if they are known to arouse strong — and incompatible — aesthetic feelings in the user community, or if they are necessary to put the user in control of choices with a significant financial impact.

Dialogs

Section Contents

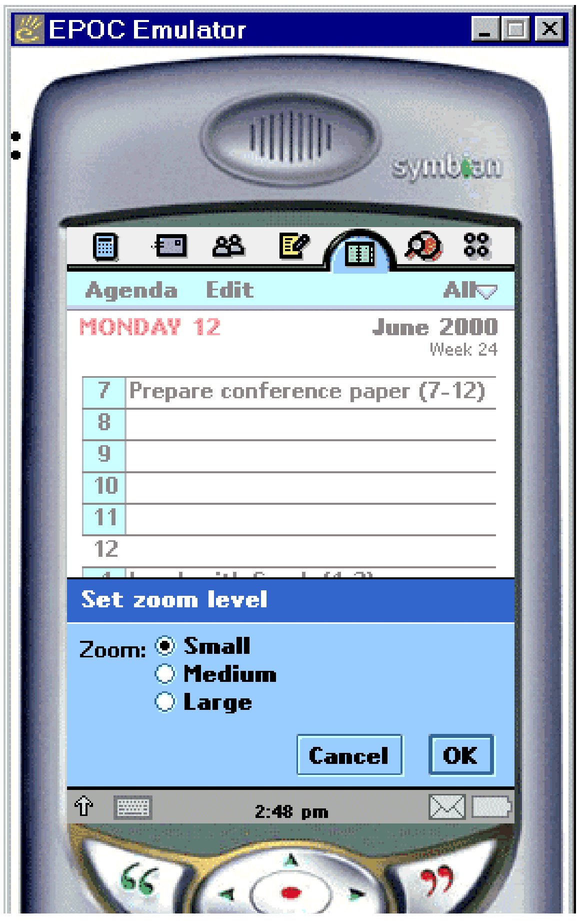
In an average application, probably about half the GUI coding is in the views. The other half is in dialog programming.

Quartz supports essentially the same dialog programming framework as EPOC R5, including

- confirmation dialogs
- single-page dialogs
- multi-page dialogs

Dialogs are significantly affected by Quartz's different form factor, lack of a keyboard, paper metaphor and the Confirm button.

All dialogs are exactly 240 pixels wide, no matter how wide their controls. Dialogs are bottom justified above the status bar. Above the top of the dialog, the window behind is faded.



EPOC R5 provided a horizontal option button list (ie radio buttons) to take advantage of the 640 pixel screen width. With only 240 pixels, it's better to use height: Quartz has a vertical option button list instead, as you can see in the zoom setting above.

Quartz also allows you to make captions shorter by breaking the text onto two (or even more) lines. You can also omit the caption, in which case the control will be left-justified to the dialog box rather than aligned with all captioned controls.

Dialog buttons

All dialogs have a single row of buttons, which is always at the bottom, and is always right justified.

If the Confirm hardware button is pressed, the default button, highlighted, will be pressed — here, that's `Done`, indicating that the settings you see will be given to the program.

In keeping with Quartz's paper metaphor, what you see is what you have.

So if the user selects another task — for instance, by switching using the app picker — the dialog is closed as if the Confirm button had been pressed.

It's therefore very important for dialog programmers to assign the correct default button, and to program a dialog just like a detail view — to save all changes when closed through the default button.

When you press any dialog button (except `Cancel`), the dialog is validated — ie, the dialog gets chance to check that all its controls are valid, and in a mutually consistent state. Then, the dialog attempts to perform the action associated with the button. Sometimes, either validation, or the action, fails. As a result, the dialog is not dismissed by the button that was pressed. When switching away from an application, Quartz simulates a press on the default button, to dismiss the dialog. If this fails to dismiss the dialog, Quartz simulates a press on `Cancel` instead. Dialog programmers *must* ensure that the dialog can be dismissed using either the default button or `Cancel`.

Some dialogs have no default button. That makes them system modal: you cannot switch away from the current application until you have dismissed the dialog. Examples of system modal dialogs include

- confirmation dialogs, which ask a question and to which you must reply `Yes` or `No`
- progress dialogs, which show while a long-running operation is happening and which allow you to cancel the operation (with a `Cancel` button) or stop it (with a `Stop` button). Cancelling undoes the entire operation. Stop doesn't undo anything that's already been done, but stops before completing everything that was requested. For instance, you can cancel a connection to the internet, but you can stop sending your outbox.

System modal dialogs generally require considered action on the part of the user.

Some dialogs use `Cancel` as the default button. This is appropriate for some complex dialogs in which `Done` cannot guarantee to be processed, because settings might be inconsistent. It's often used for multi-page dialogs, partly because these are complex, and partly because "what you see is what you have" isn't very applicable to multi-page dialogs — initially, you only see a part of the picture.

Some dialogs use an alternative label instead of `Done`. Settings dialogs can use `OK`. Dialogs which initiate actions can use a verb describing the action, such as `Find` or `Add`.

information dialogs give important application-related information and have a single, default, button labelled `Continue`.

Focus

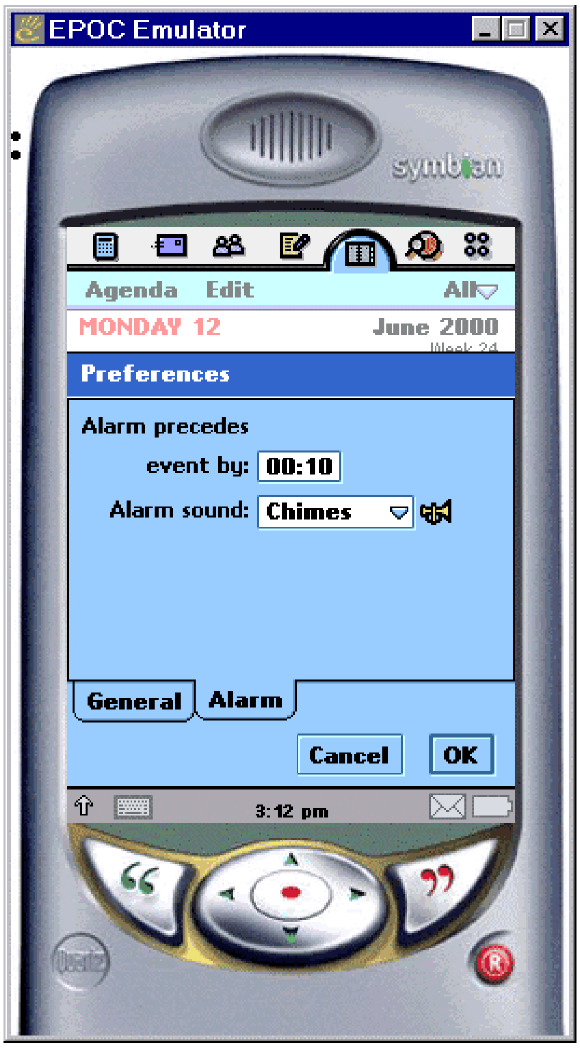
Dialogs in Quartz do maintain focus: focus is needed, for instance, to display a cursor and to invoke handwriting recognition to allow you to edit in a text field.

Focus is not indicated by a highlight on the captioned control: this is unnecessary for all except for text fields, and for text fields the flashing cursor indicates the location of focus.

Focus is transferred between fields using the pen, or the up and down arrow keys.

Multi-page dialogs

Multi-page dialogs are supported. For instance, the Agenda preferences dialog:



Alarm precedes event by: 00:10

Alarm sound: Chimes

Tabs are on the *bottom* of the dialog rather than the top, to keep them close to the dialog buttons. The dialog is justified to the height of the tallest page (not the one showing here).

Use multi-page dialogs very sparingly: they can be intimidating to the user, which is against the spirit of Quartz.

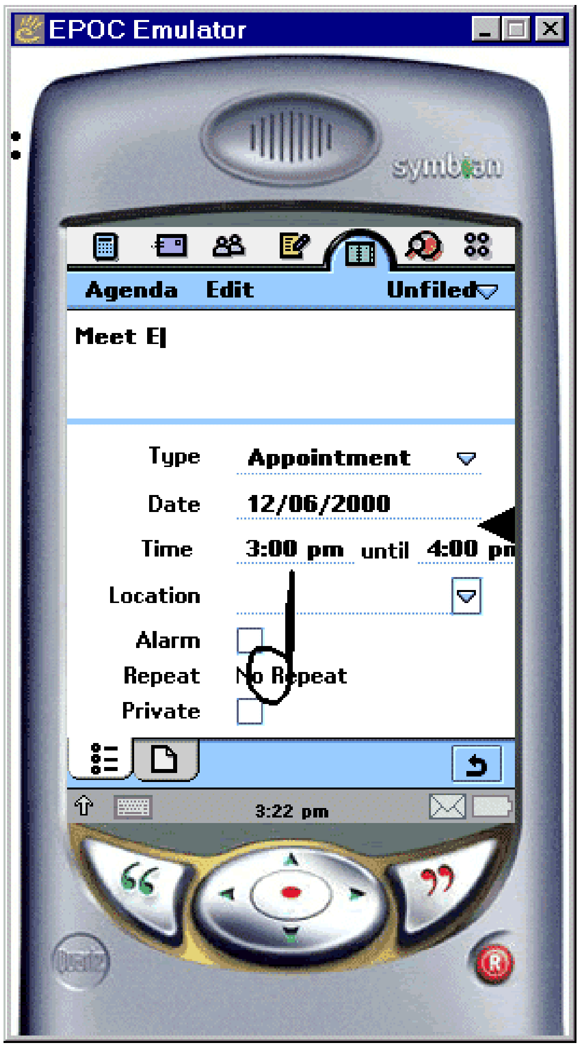
Text input and handwriting recognition

There are two ways to enter text in Quartz:

- via handwriting recognition
- through a virtual keyboard

Actually, on the emulator, you can enter text directly, using the PC’s keyboard — this is a convenience for developers, and you shouldn’t let this convenience allow you to forget your real users if you’re designing an application which is heavy on input.

Here’s the handwriting recognizer, while entering a letter *d*:



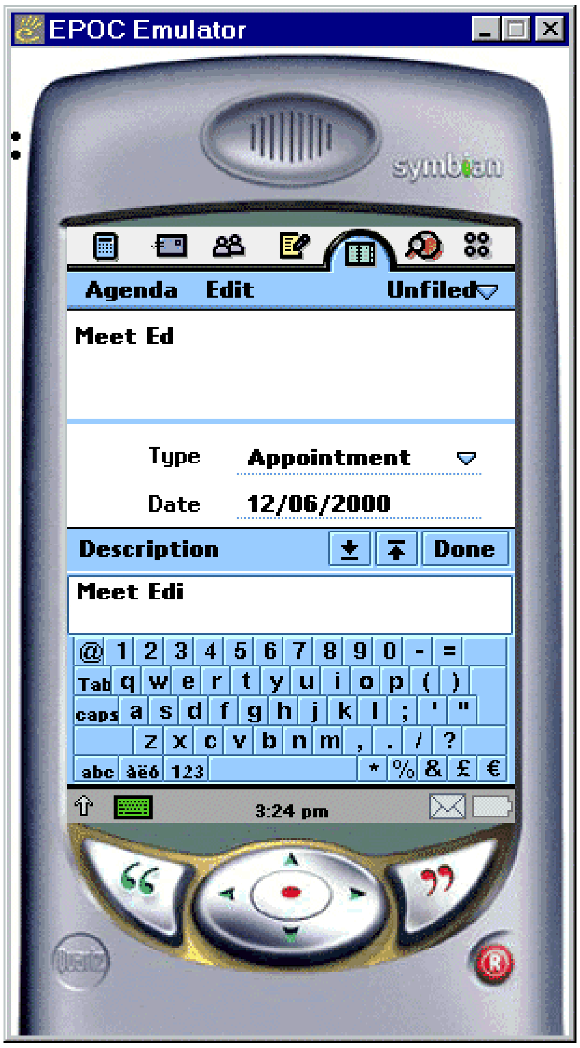
When focus is on a text field, the end-user simply writes with the pen — below the black triangle on the right-hand side for lower-case, level with it for upper-case, and above it for numeric input and punctuation.

While the triangle is showing, any tap (instead of a stroke) is fed through to the application underneath, so that on the one hand you can use the whole screen area for handwriting input, and on the other hand the recognition framework doesn't prevent normal navigation using pen taps. The framework disappears in any case if focus isn't currently in a text field.

The flipside of this optimization is small: a period must be entered by a small circle, not a simple dot.

As a programmer, it's simple to activate the recognition framework: simply give focus to a control, such as an edit window, which requires it. By taking focus away again, the recognition framework is deactivated.

Here's the virtual keyboard:



You can invoke the virtual keyboard by pressing the keyboard icon on the status bar. The keyboard can be dragged up and down so you can see data underneath it. When a dialog is active, the initial position is chosen to avoid the currently-focussed field.

System support

When compared with EPOC R5, the look-and-feel of the Quartz GUI shows some continuity, and some departures. Obviously, for you as developers, that means some continuity and some departures in the underlying APIs, and likewise in your programs.

This paper concentrates on interaction design. It's beyond its scope to go into the programming details too deeply. However, here's a brief overview.

EPOC R5's EIKON GUI is replaced by a two-layer GUI comprising

- *Uikon*, a general-purpose GUI layer shared by both Crystal and Quartz
- *Qikon*, Quartz-specific additions to Uikon, and also look-and-feel implementations of some Uikon facilities

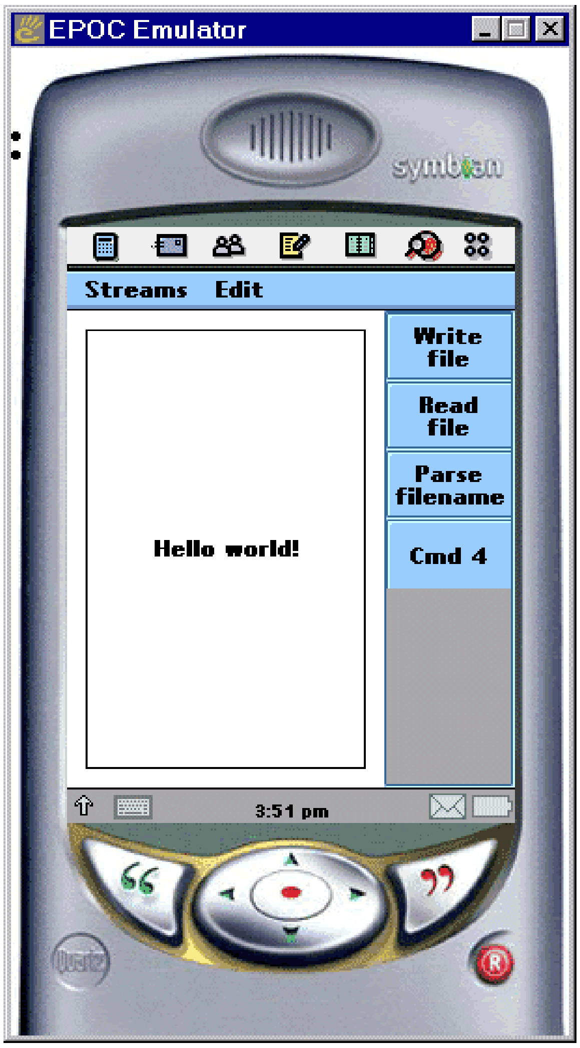
Uikon includes a new *view architecture*, which defines the idea of a view, and allows an application to register and implement multiple views. The view architecture is used to switch between applications and views, and to implement DNLs.

A key Qikon component is the category manager, which does the hard work of maintaining the folder menus required by Quartz applications.

The Uikon/Qikon split has upsides and downsides. On the one hand, it allows a carefully controlled subset of code to be shared between Quartz and Crystal (Crystal uses Uikon+CKON instead of Uikon+Qikon). Also, the common heritage in EIKON (such as class names beginning with `CEik*` and `TEik*`) eases migration from EIKON code.

On the other hand, you are faced with two naming conventions (`CEik*` for Uikon and `CQik*` for Qikon), and two sets of header files, when using the GUI for a Quartz application.

Also, some Uikon APIs are not relevant for Quartz so, although they are there, you should ignore them except, perhaps, for testing purposes. One excellent example: Uikon supports the old-style EIKON toolbar. So you can code a GUI like this:



which works, and is handy for test code. But for many reasons toolbars are wholly inappropriate for a real Quartz application. Most obviously, they waste horizontal space — a plentiful resource in EPOC R5, but scarce in Quartz. More subtly, toolbars were there to advertise obscure features and to make them easily accessible: Quartz deprecates obscure features, and has other ways to make things easy to use.

Don't even think about using an EIKON toolbar in a real Quartz application! Design away the need for a toolbar if you can, or use the button bar if you must.

An important system support component for Quartz (and for forthcoming smartphone designs) is a *memory manager* which ensures that, when memory is getting tight, applications which don't need to be running are shut down.

Finally it's worth mentioning Quartz's Java implementation. Quartz implements PersonalJava 3.0 and JavaPhone 1.0. JavaPhone provides access to the heart of Quartz's communicator functionality — the contacts, schedule, power monitoring and telephony APIs. Quartz's PersonalJava implementation has been tailored to suit the device: for instance, file-related dialogs have been removed, the menu bar is displayed permanently, Quartz controls are used as Java peers, and handwriting recognition is enabled when appropriate. On the other hand, no Java APIs exist for some very useful Quartz goodies — DNLs, and the view architecture. You won't be able to use these from pure Java code.

Designing for Quartz

Contents

Don’t just write for Quartz. Design for Quartz.

This paper aims to give you the tools you’ll need to design for Quartz, so that you have time to think the implications through. Then you’ll be ready when Symbian begins to release Quartz SDKs and tutorial materials which go into greater depth on writing Quartz applications in C++.

Here are basic design scenarios:

migration	You have a running EPOC R5 application. You’ve designed it like we design our applications — with separate GUI and engine components — and you want to port the GUI to Quartz. In most cases, this is possible. You shouldn’t need to rewrite the engine. You will need to rewrite the GUI — large parts in some places, small parts in others. You may want to change application functionality. In any case, design first: take hard and deliberate decisions about the user experience. Take inspiration from our experience porting the standard applications, and from the pointers below.
no hope	You have a running EPOC R5 application, which can’t be ported to Quartz. Examples would include a typing tutor (no keyboard on Quartz!), or a file manager (no user-visible file system on Quartz!).
inspiration	The convenience, power and integrated wireless communication offered by Quartz will enable many entirely new applications which weren't suitable in any previous type of device — EPOC-based or otherwise.
porting	You have a running application on another platform such as the PC, Palm or Pocket PC, which you believe would be good on Quartz. Clearly, porting to a different platform is going to be a major exercise. So it’s doubly worthwhile to think about the *design* aspects before committing yourself to coding.
minimal	You are porting an application from another platform — EPOC or otherwise. You are not primarily concerned about ease of use, because your end-users are not in the mass market but are specialists who just want the application quickly. Any GUI, using any set of EPOC APIs, will do. You probably won’t use the view architecture or DNLs. You might even use the EIKON toolbar if you have to. You might not save data when your application goes into background — this is risky, but plausible if you can be certain that the system on which your application is deployed will never come close to running out of memory.
Java	Most of the above choices are open to you when developing in Java. For further details, see below.

If your scenario matches the migration, inspiration or porting scenarios above, then what follows is for you.

Make the tough decisions

The very first thing you must do is make the tough decisions. If you’re used to designing for PCs, or even for EPOC R5, you will find Quartz difficult at first:

- on Quartz, the list views and paper metaphor are quite different from these other systems. You’ll get used to them, but you’ll need to work at it and try a few things with your first few application designs.
- on Quartz, you have to ruthlessly strip out unnecessary functionality. Distinguish functionality that is absolutely necessary, from nice-to-haves, and things that get in the way. Don’t skimp on anything that is absolutely necessary. If you have time, add a few nice-to-haves. But don’t do anything that gets in the way, even if at first it looks nice-to-have.

As you’re designing an application, you’ll probably change your ideas about its structure, and what’s really necessary, several times as you go through. But bear these factors consciously in mind, and don’t stop until you’re happy you’ve got it right.

Bear in mind the client area rules: the screen is always 240 pixels wide, and is usually around 230-260 pixels deep. That’s about 14 lines of text in a reasonable size. Exploit the list view metaphor when you can: it’s powerful and will scale independently of the precise screen height. When you need a graphic view (a voice recorder, a month-at-a-time display, the current state of a game), design it with some spare room in the vertical direction.

Don’t use an EIKON-style toolbar! You might find a Quartz-style button bar useful, but you should confine its uses to essential navigation and view switching.

Controls written for EPOC R5 can usually remain intact. For instance, a rich text editor control may not need much change, even though its environment in the notes tab of Quartz’s Jotter application might be very different from its environment in EPOC R5 Word.

Any part of a control which implements select-and-open semantics may, however, need alteration for Quartz, to implement single-tap semantics instead.

EPOC R5 supported multi-view applications, by ad hoc replacement of members of the app UI class — toolbar, toolband, menu bar and shortcut keys. Applications written for Quartz should use the view architecture for this purpose instead.

Quartz views should save their data when the view is deactivated, and dialogs should be programmed to implement their default button in case the view containing them is deactivated.

If your application was heavily file-based, you’ll need to consider how to constrain its functionality so that files are no longer needed. You can take an approach like Jotter (titles in the list view function like a file name), or folders in list views (which function like directories).

Sometimes, you may discover that what you thought was one application is in fact two applications — like the distinction between Agenda and ToDo in Quartz. Don’t forget that agenda and contacts data are managed by EPOC servers in Quartz, which you can access from other application programs also, if you need to.

More on Java

The goal of Quartz’s Java implementation is quite different from that of EPOC R5’s.

- in EPOC R5, the primary requirement was to enables JDK 1.1 applications, written perhaps for a desktop platform, to run directly on an EPOC device, provided the device had enough memory and the screen was basically big enough.
- in Quartz, the objective is to make the Java implementation good enough so that applications written in Java, specifically for Quartz, would look as similar as possible to native applications.

Java has now gone beyond JDK 1.1, which was current during EPOC R5’s Java implementation project, to Java 2. Java 2 Standard Edition and Java 2 Enterprise Edition programs are almost certainly no-hope cases for Quartz: there is no point in aiming to support them.

Instead, Quartz provides a rich implementation of Java 2, Micro Edition, with the PersonalJava profile and JavaPhone APIs. Additionally, Quartz builds in as much Quartz look-and-feel as possible to the AWT peer classes. The net result is that a Java application written specifically for Quartz will look and work well: furthermore, in keeping with Java’s openness, such applications will probably be easy to port to and from similar hardware platforms.

If you want to code in Java, note that you may have to reinvent some Quartz look-and-feel. Specifically, Java on Quartz doesn’t support the view architecture, so a multi-view application in Java would need to emulate whatever parts of the view architecture it needed.

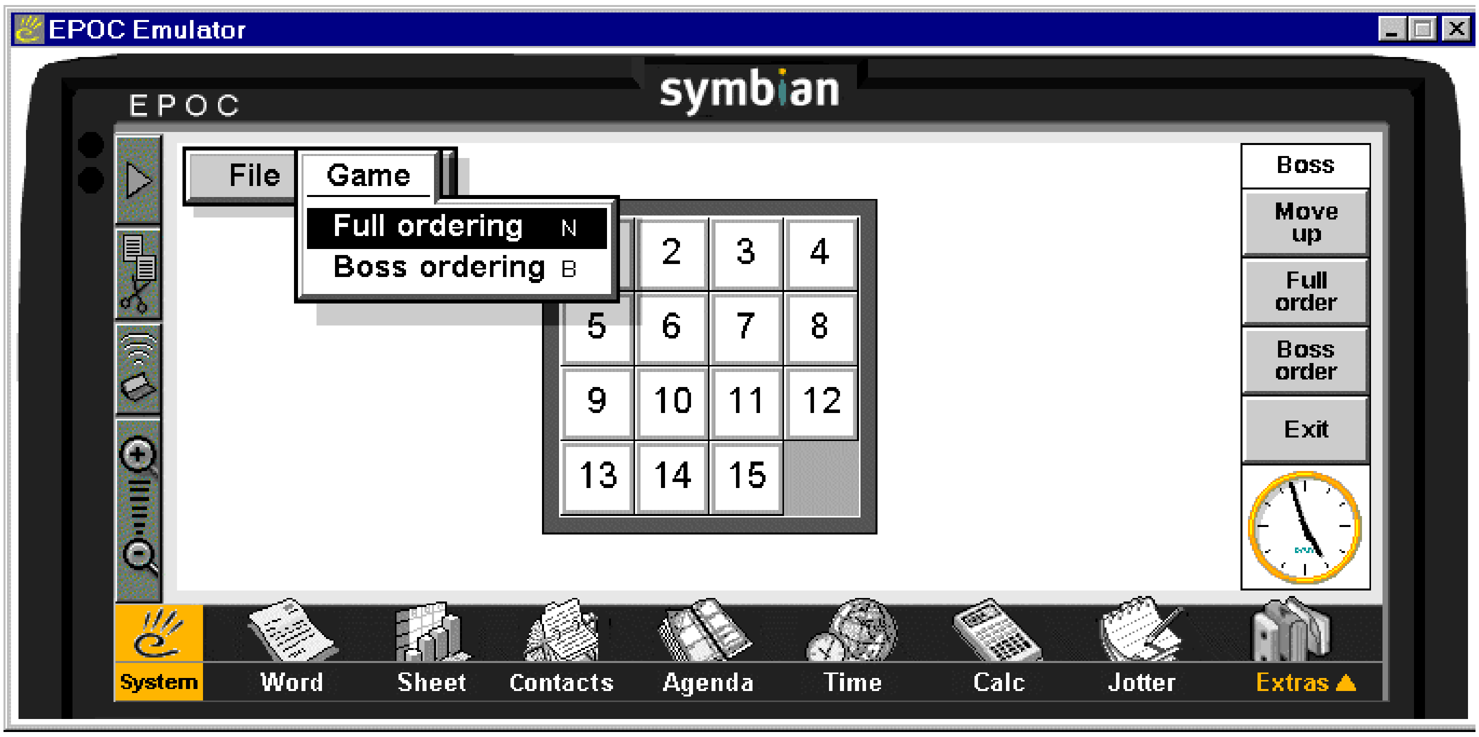
Some simple examples

Section Contents

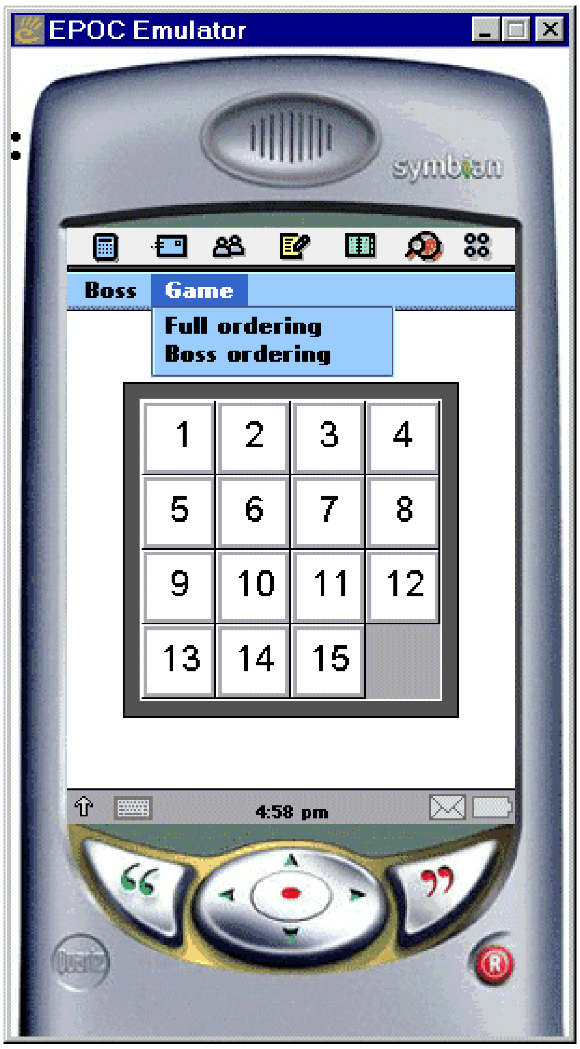
I’ll finish on a personal note with some examples taken from tutorial exercises I’ve written for EPOC R5.

Boss Puzzle

The first is the Boss Puzzle, which I wrote in March 1997 for the very first public EPOC SDKs. Here's the original:



The Boss Puzzle wastes screen space on EPOC R5. Here it is on Quartz:



You'll notice

- the square game board fits much more nicely into the Quartz form factor than it does into the standard EPOC R5 form factor

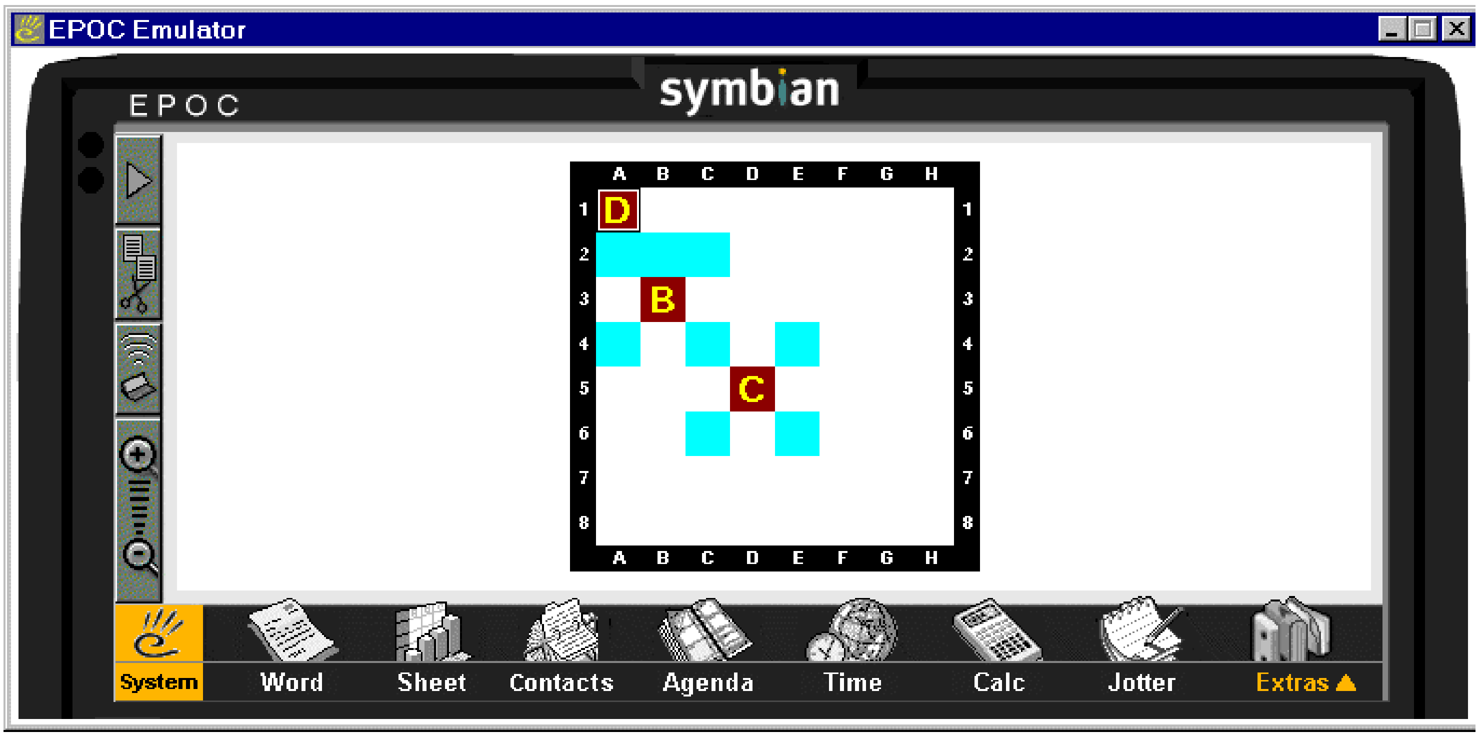
- the menu options are identical — though I've renamed the `File` menu to `Boss`
- there are no shortcut keys on the menu items
- I don't need a toolbar

There was very little to do to port the Boss Puzzle. The view already required only a single tap to move a tile, so no changes were needed there.

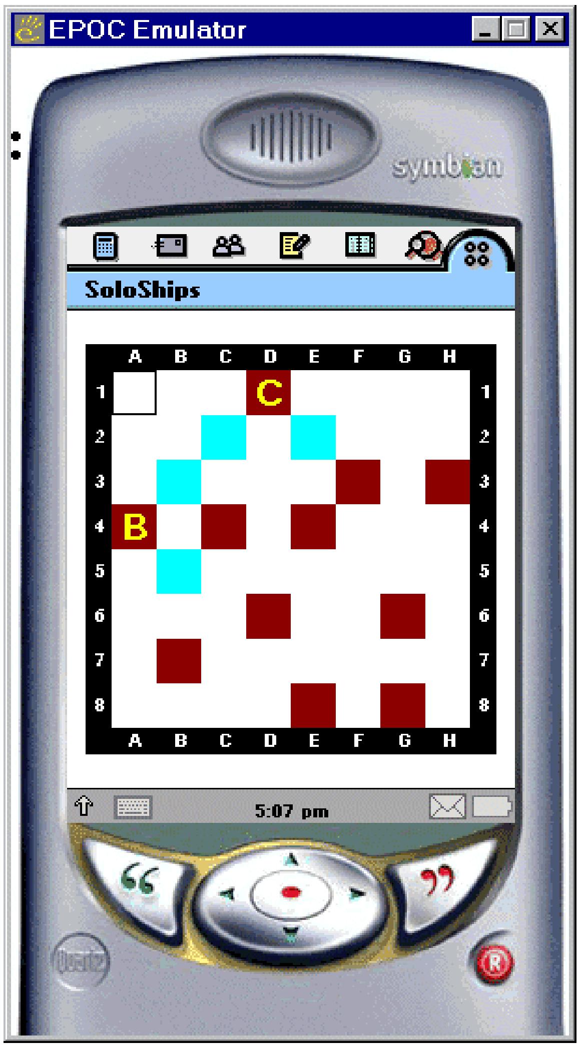
Solo Ships

If you've read *Professional Symbian Programming*, you might be wondering how I'm doing with Battleships on Quartz.

You'll remember that Battleships was a sufficiently complicated program that, even on EPOC R5, I developed it in several stages. The first stage we meet in the book is Solo Ships, a single-player version of the game which looks like this:



Like the Boss Puzzle, Solo Ships wastes space on the EPOC R5 landscape screen. On Quartz, it looks a little more comfortable:



Solo Ships fits the screen better because it’s programmed to fill the screen, minus a border of 10 pixels. Boss, on the other hand, was programmed with a fixed size.

Solo Ships required some interesting adjustments:

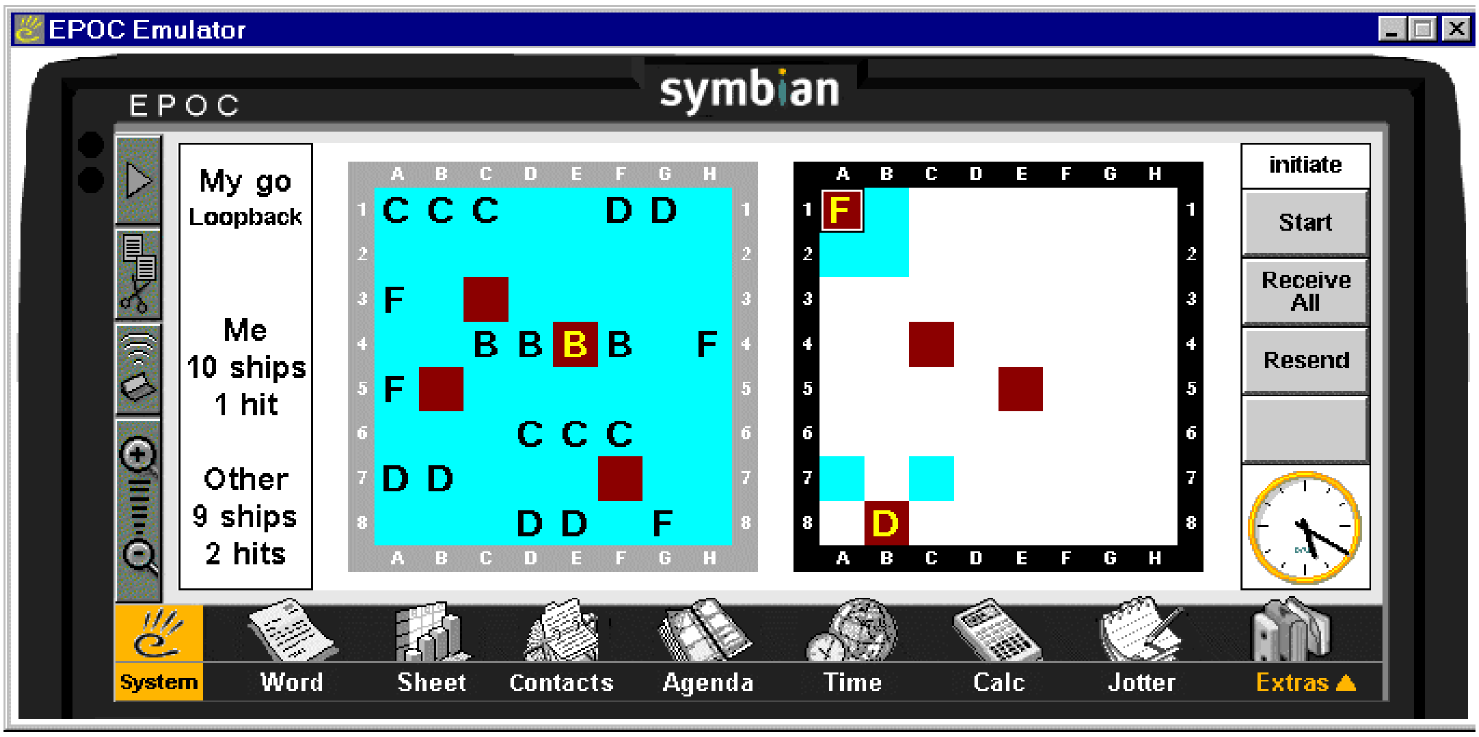
- previously, I had used select-and-open. Converting to single-tap involved some interesting programming decisions. I now think that my code is better than the original EPOC R5 code, but a more complex view would certainly have required more detailed programming changes. These changes could potentially interact with optimized redrawing code.
- on EPOC R5, I had used arrow keys for navigation, and Space as the “hit” button. In Quartz, the Confirm key is mapped to Enter on the emulator. I had to change my key handling function appropriately.
- I had built powerful zoom and object embedding functions into Solo Ships. My primary motivation for doing so was to show off these features in EPOC R5. One of the rules in Quartz is that you remove unnecessary features: I have not exposed zooming in the UI, and there is no way to embed on Quartz. I left the zooming and embedding code in the application, even though it is dead.

In both Boss and Solo Ships, the port was relatively easy — about four or five hours work in total. The result was an application that worked well for Quartz, but which left a little dead code. That’s usual for migrating any program, so I’m happy enough about it.

Battleships

Section Contents

I haven’t yet implemented full Battleships on Quartz. Here’s a game in progress on EPOC R5:



Battleships is a much better fit on EPOC R5's landscape screen.

I can connect to opponents in one of three ways: infrared for local games, SMS text messaging for remote games, and loopback to another instance of the program on the same machine, for local testing.

I can have several games active at once, each identified by a different file. I can close each game and return to it later, perhaps in response to an incoming SMS message.

Here are the problems I'll have to tackle with the port to Quartz:

- how will I fit the two fleet views and status information onto a smaller, essentially square, screen?
- how will I replace the file-based method of opening, saving and re-opening games in EPOC R5, with a user interface that hides files?
- how will I develop the communications components and the application in stages, so that I can be sure at each step alone the way that each component is working?

Each of these issues will require some real thought.

There's one piece of very good news, though: Quartz has inbuilt telephony, and a nice API to incoming messages. With the Quartz version of the game, sending and receiving moves via SMS will be *very* easy for end-users. As a developer, I'll have to think about how to exploit this, but I'm excited that I'll be able to deliver a much easier-to-use application.

The ease of use from integrated telephony, and hiding the file system, could be the critical usability factors which allow Battleships to really take off in its Quartz implementation.

Hiding the file system

Battleships for EPOC R5 uses a separate file for each half-game. A player can exploit this in two ways:

- use a single file, to open a game with another player on a different machine using infrared or SMS communication
- use two files, to play a game using loopback (for testing) on the same machine

Because a separate file is used for each half-game, a player can have an artbirary number of games running at any time.

We can't use files in this way on Quartz. We can only launch application programs, and any application must hide files from the end-user. I considered the following alternative designs for Battleships:

- use one application only. The application can play a single game simultaneously — multiple games in parallel are not supported. The application can play using genuine comms (eg IR or SMS) only — loopback is not supported. This is very simple, but perhaps too restrictive. I would like to be able to play multiple simultaneous games. And although loopback isn't a star feature for real games, I need it for testing.
- as above, but use one application per protocol, so that one game can be played using infrared, and another using SMS (say). In fact, I could add further applications for loopback — one for the initiating side, and one for the listening side. It looks as though this approach could generate a lot of applications.
- use a single application, with a list view containing each half-game. This seems the natural approach. Whether it works well for the end-user or not will depend critically on how the list is displayed.

I would be prepared to go with the first approach if I *really* had to cut down Battleships to the basics. The second approach replaces the evils of a user-visible file system with the evils of a sprawling set of related applications, which is probably much worse. The third approach fits in well with Quartz, and allows us to produce an elegant, scalable application supporting any number of games and any number of protocols.

So I will use a list view to contain a list of half-games. Each half-game can be new, in progress, or finished — only when a game is explicitly deleted is it removed from the list. If you open an item in the list, it will show a detail view with the current state of the half-game.

I'm using the term half-game here, because it is more precise. In the UI, I would not dream of exposing such a technical word! In the UI, I'll just call each half-game a "game".

The important thing now will be to design the list in such a way that it is really easy to use. I can control two factors here:

- which order do I display items in the list?
- what do I display about each item?

In setting the display order, I want the most active games near the top. So, I'll make sure that each game's summary information includes the date and time of the last move, and sort the list using that field.

The information I display about each item should enable me to identify the game, and should also enable me to identify whether any action is needed on my part. I will use

- a text description to identify the opponent's name, address (if needed) and communications protocol
- bold, or a flag, to indicate that it's my turn to play

The detail view

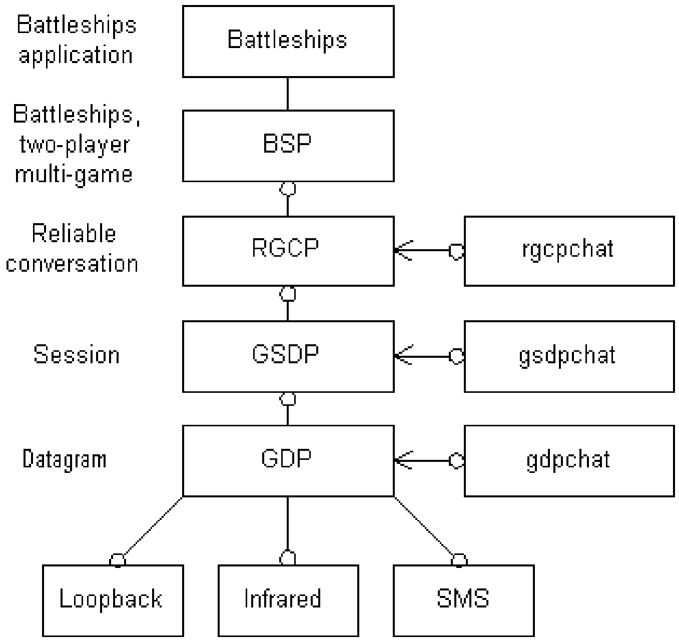
Tapping an item should open its detail view. The detail view must show the current state of the game, which includes

- my fleet
- my opponent's fleet
- summary information

I will use tabs to show my and my opponent's fleet. I will use some information on the button bar to show summary status information, and a curly arrow on the button bar to indicate that I wish to return to the list view. I might use a third tab, or a dialog, to show more detailed status information.

Getting there from here

In the EPOC R5 version of Battleships, we faced the question of how to get a reliable comms stack — TOGS — working, and then build Battleships on top of it.



The problem breaks down as shown above. We'll proceed to implement TOGS, and Battleships, for Quartz by the following steps:

`gdp.h`, `gdploop.dll`, `gdpchat.app`	Minimum GDP loopback test application. We're confident the technology will work ok. The issue here is the design of the Quartz version of `gdpchat`. In EPOC R5, we had fixed locations for the last outgoing and last incoming message. It would be better to use a list. Also, the protocol and to-address should be selectable at run-time rather than hardcoded into the program, and the addresses should be shown along with the message text.
`gdpir.dll`, `gdpsms.dll`	Once gdpchat in loopback mode has been developed, the concrete GDP implementations can also be developed, in parallel with the development of GSDP and RCGP below.
`gsdp.dll`, `gsdpchat.app`	GSDP server, and associated chat application. Here we get some more complexity in the EIKON application. GSDP supports multiple sessions. In EPOC R5 this is done with multiple files. In Quartz, this isn't allowed. We need base view = list of opened sessions. A second view, on a tab from button bar in base view, is a list view of *all* messages on all sessions. Detail views are available for sessions (showing protocols, addresses, port numbers and the like), and for messages (showing text, and perhaps time and date).
`rgcp.dll`, `rgcpchat.app`	RGCP protocol and "chat" application — which, in reality, is a conversation application. `rgcpchat` supports multiple sessions, each of which is a conversation. This is the right time to bring in the full model proposed for Battleships: base view = half-sessions, for each half session, detail view = list of incoming and outgoing messages, and for each message, detail view = full message text and info such as timing etc.
`battleships.app`	This should be a straightforward adaptation of `rgcpchat.app` and Solo Ships. The detail view of a conversation, instead of being a list of messages, is a Battleships game. The game view involves two panes, one to show my fleet and one to show the opponent's.

Summary

Quartz is Symbian's reference design for tablet communicators. Reference designs are a new way for Symbian to do business: the responsibilities of product development are clearly divided between Symbian and the manufacturer, to maximize innovation in the industry.

Quartz is addressed at less technical end-users than previous PDA platforms. This results in many significant high-level UI design decisions: the paper metaphor, task orientation, browse-mostly, single-tap to open, hiding the file system and task list. Everything else, including the details of list-based applications and dialog semantics, follows from the basic usability criteria and the Quartz form factor.

When designing your own applications, you should take inspiration from the built-in applications, and follow the style guide carefully. Like the built-in applications themselves, you may find you need to bend the rules occasionally — but you should never ignore them, and you should never break them entirely. Time spent at the design stage will be rewarded by an application which fits well on the Quartz platform, and is a pleasure to use.

With smartphone designs in the works, Symbian is aiming at smaller and more portable devices, and a greater number of less technical users. Lessons learned from Quartz will be invaluable when designing applications for these new device families and their users.

Further Reading

Professional Symbian Programming, by Martin Tasker et al, published by Wrox Press (February 2000), provides a comprehensive introduction to programming EPOC R5. Chapter 15 gives some useful additional background to device families.

See the author’s companion paper, **Memory Management in Wireless Information Devices**, also presented at the July 2000 Wrox Professional Wireless Developer Conference, for more information on memory management on devices with no user-visible task list.

Symbian owns, develops and licenses a software platform for next generation mobile phones. Owned by Ericsson, Matsushita, Motorola, Nokia and Psion, Symbian's mission is to license the Symbian platform to all mobile phone manufacturers and to create a mass market for next generation mobile phones by working closely with wireless networks, content, services, messaging and enterprise wide solution providers. See **www.symbian.com** for more information.

Trademarks and acknowledgements

The author wishes to thank the many Symbian staff who supplied valuable information and review comments.