

Porting BugMe! to Series 60 and UIQ

Andy Weinstein, Degel Software Ltd
Version 1.0, August 2003

1. Introduction

BugMe! is a successful Palm OS application. What follows is an account of how two companies, Electric Pocket Ltd. and Degel Software Ltd., worked together to produce new versions of BugMe! for two variants of Symbian OS. The significant differences between the capabilities of Palm OS and the target platforms meant that porting wasn't just a technical task. For the port to Nokia's Series 60, the application had to undergo a far-reaching redesign of its user interface. For the port to UIQ Technology's UIQ, where the user interface is much closer to Palm OS, the external changes were less radical. Each platform raised various technical issues revolving around specific API capabilities. Differing implementations of phone and camera support meant that integrating these key new features required a different high-level approach for each target platform.

2. What is the BugMe application?

BugMe! for the Palm and PocketPC is a product that allows a user to create the electronic equivalent of a sticky note. The user simply uses the screen as their writing surface, and the resulting digital ink is stored as it appears on the screen. What you write (or scribble) is what you see (with no handwriting recognition). In addition, the user can attach a timed alarm to a note. The result is that you can take out your PDA, scribble (in your own handwriting) "Kids at 3", and attach an alarm to go off at 2:30. When 2:30 comes, your PDA will beep and you'll see the note you wrote.

Electric Pocket Ltd., the maker of BugMe!, has produced many successful Palm OS and PocketPC applications. With the advent of the Series 60 platform and the introduction of the Nokia 7650, the company made a strategic decision to develop BugMe! for Symbian OS, starting with Series 60. Though UIQ was on the horizon, and a more natural target for a Palm application, Series 60 was perceived as having a larger user base and hence a larger potential market. As it happens, while Series 60 has more users, it seems that P800 users have more appetite for add-on software; sales of the P800 version of BugMe! have outstripped those of the Series 60 version. At any rate, once the decision to develop for Symbian OS was taken, Electric Pocket decided to look for a company with proven Symbian OS experience as a way of getting into this new and exciting market.

Electric Pocket found Degel Software Ltd. via <http://www.symbianpages.com/>, a site run by Symbian. It's a place for all companies doing any kind of Symbian OS related work to advertise themselves and find others – a Symbian Yellow Pages, or Symbian Pages for short. At that point, Degel had already earned a reputation for successfully porting applications to Symbian OS. Electric Pocket correctly thought Degel a safe bet to develop the Series 60 version of BugMe!.

The relationship between the two companies was good from the start as the two teams dove in and tried to figure out what it would mean to put an ink-centric product onto a platform which didn't support ink! The key to solving the problem was to focus on what each platform *does* offer, instead of what it doesn't. At the same time, a deeper understanding of the essence of BugMe! developed: the developers stopped thinking of it as an ink-centric reminder product, and reframed it as a product that takes advantage of a device's data input capabilities to allow a user to create reminders efficiently.

3. BugMe! on Series 60

Right from the start, it was apparent to all that BugMe! for Series 60 was going to be written from scratch. The difference in both the GUI concept and the underlying API available for the platform, together with the fact that there is not a lot of middleware functionality in BugMe!, made it clear that starting with a clean slate was the right thing to do. So the first task was to think about what the essence of BugMe! is, and understand how to bring it into a new setting.

BugMe! is fun and satisfying to use because the user gets to feel like they're getting the most from their device, exploiting its uniqueness. On the Palm or PocketPC that meant ink. On Series 60 it would mean photographs,

sounds, and contacts, even letting the user make phone calls directly from the alarm screen. We also decided to allow textual input, especially since the predictive text capability in Series 60 meant that this too would be an efficient data entry path for some users. But text was seen as secondary, as it was on other versions of BugMe! Concept in hand, we then stepped up to meet the challenge of designing the user interface. This had two aspects: both screen layout and workflow. From the beginning, Electric Pocket felt strongly that the professional artwork that was available for the existing versions of BugMe! should also be incorporated into the Series 60 version, since it gives BugMe! a distinct added-value over other reminder products.

So the issue became how to allow the user to efficiently create a reminder with stationery, multimedia content, and fancy alarms on the Nokia 7650. This required porting from Palm's touch-centric interface to a device with no touch screen – a phone with a five-way toggle and two soft buttons, one of which is dedicated to “Back” operations. Instead of taking a more traditional menu based approach, which would end up requiring countless clicks to open the menu, navigate it, and choose the next operation for each step, we choose a “wizard” approach.

Since the creation of a reminder is quite stereotypical, we understood that instead of making the user decide what the next step should be, we could run the user through the process automatically. The process would begin with the user initiating the creation of a reminder, and then the system would prompt him: first for stationery type, then for the primary multimedia type (photo, sound, contact), then for the actual multimedia content, and finally for text and the alarm time. After the reminder is created, the user could choose to add additional content via the menu.

Once we had the design for creating a reminder, we needed to address how the reminder would appear when its alarm went off. This was a combined technical and design issue. The technical issue had to do with whether or not we could use the built-in Alarm Server to keep track of the alarms, and the design issue was how the screen would look in order to clearly differentiate the alarm screen from the editing screen, without straying too far from the overall Series 60 look.

When we looked into the issue of the Alarm Server, we found that we would not be able to control the GUI when an alarm went off – the same type of pop-up generated from the built-in Calendar application also would come up for BugMe!. Even the sound which is played when an alarm goes off can't be controlled. The underlying Symbian API does support specifying a custom alarm sound, but Series 60 overrides this to obey the user's profile. Another nice feature of BugMe! is variable snooze time, and the built-in Alarm Server GUI had no such feature. So it seemed that the Alarm Server was the wrong route.

Implementing our own alarms wasn't difficult to do – we took a simple approach based on using timers. But there was one catch that we were not able to overcome – the fact that the Series 60 profile is inaccessible to applications, making it impossible to respect “silent” mode. Also, Series 60 also does not provide an API for the vibrate functionality. In the end, we added an option in BugMe! to allow the user to set the sound to be off. This is obviously not ideal, and is a direct result of the fact that Nokia had to make some tough choices when designing Series 60. Nokia did an excellent job of providing an interface that is consistent and thoughtful at every turn, and therefore a pleasure to use. Perhaps their concern for protecting that consistency led them to keep the platform slightly less open than they might have, with the resulting tradeoffs.

We later found out from Nokia that there is an indirect way to detect whether the user has set a profile which calls for the device to be silent. This is not at all straightforward, and it isn't documented per se, but it should work. The code is now available on Forum Nokia, and here's the relevant piece:

```
TBaSystemSoundType soundType(KUidSystemSoundEvent, KNullUid);
TBaSystemSoundInfo soundInfo;
RFs& rFs = iCoeEnv->FsSession();

BaSystemSound::GetSound( rFs, soundType, soundInfo);
// if soundInfo.FileName( ) == Z:\\system\\controls\\No_Sound.wav
// then Profiles Notification Tone setting is OFF
//
// else if soundInfo.FileName( ) == Z:\\system\\tones\\NokiaTune.rng
// then Profile's Notification Tone setting is ON
```

This will allow us to do a future version of BugMe! which can respect that setting. In addition, by making use of the Agenda API, we'll be able to create a regular alarm to occur in such a case. If the user has set their profile such that regular alarms will cause the device to vibrate, then the vibrating will occur. Again, this will not be ideal, because the GUI for the Agenda alarm will be in the foreground, with BugMe! behind it. The text will have to say something like "BugMe! Silent Alarm". It will require two steps to dismiss an Alarm (first the Agenda alarm, then BugMe!), and there will be two snooze mechanisms available. But it will give the user the ability to work silently.

The design of the alarm screen itself was an interesting challenge. We wanted the alarm screen to look similar to the screen used to edit and view a reminder. We wanted that edit and view reminder screen to look like a regular Series 60 screen. But we didn't want the alarm screen to look like a regular Series 60 screen! We wanted the alarm screen to look special, and not have the look of a regular application – after all, it's an alarm, an irregular event. How could we solve this paradox? After looking carefully at what gave Series 60 apps their "standard" look, we concluded that the status pane, which is basically the upper 20% of the screen, is what made a screen look standard. Its most striking feature is the relatively thick horizontal line along the bottom of the status pane, which separates the application title, on top, from the application content, below. So we removed the status pane, but continued to draw the screen title as it would appear in a status pane. We left the application content just as it appears in the edit and view screens. By eliminating the thick horizontal line, we achieved most of our goal of "same but different". In addition, we added a 3D border all the way around the screen, further setting off the appearance of the window and giving it a kind of pop-up look. All of this without affecting the layout of the content, which remained the same as it was in the edit screen.

Contact integration was a key feature that sets apart the Series 60 version of BugMe! from the Palm OS version. Because the Symbian OS devices are always also phones, it is an obvious and almost required enhancement for an application being ported to take advantage in some way of the device's communication capabilities. In BugMe! for Series 60, the user can choose a contact from the contact list to add to a reminder. When the alarm goes off, BugMe! offers the option to directly dial the contact. If the contact has more than one phone number, the application presents a list of the available phone numbers for the user to choose from, and then initiates the call accordingly. The code for this had to be implemented "manually" – the contact info was read from the contact database using the Symbian OS low-level API. Nokia does provide a slightly higher level interface to the contact database, but it hides some of the functionality in the low-level API and proved unsuitable for our needs. On the other hand, Nokia supplies a GUI control to implement a searchable list. This was helpful in quickly implementing the GUI for the searchable list of contacts, from which the user chooses.

4. BugMe! on the Sony Ericsson P800

The move to the Symbian OS UIQ platform, as implemented by Sony Ericsson for the P800, was in many ways a more natural one for BugMe! Because UIQ is based on a touch screen interface, the original BugMe! for Palm OS concept of quick reminders driven by ink could carry over without change. It was clear to us that this would dictate the basic shape of the product. The differences would arise from three factors: the device's native added-value capabilities (phone and camera), the slightly different UI parameters and conventions, and the goal of producing a somewhat trimmer, highly focused product.

Technically, the basis for the UIQ version was the Series 60 code. While the UIQ GUI is closer to the Palm, its underlying API is totally different and the non-GUI parts of the UIQ API are almost identical to Series 60, particularly in those portions used by BugMe! So we were able to port over the non-GUI part of the code from the Series 60 version with very little effort.

One area where we thought that a new UIQ API might serve us well did not pan out. We found that we had to preserve our internal handling of alarms using timers, in spite of the existence of a new UIQ Alarm API. Unlike the Series 60 Alarm API, the UIQ Alarm API does have a flag for defeating the GUI of an alarm. Unfortunately the P800's implementation of the API does not respect the flag – the built-in alarm pop-up is always shown.

The first major GUI issue which arose was the implementation of a thumbnail grid. In the Palm OS version of BugMe!, the user's notes can be viewed as thumbnails, nine at a time in a grid layout. At first we assumed that the P800 would have a control to support this mode of presentation directly, since this type of layout is one of the

options available in the built-in Application Picker. There, the different application icons together with the titles are laid out in a 2 x 4 grid. It turns out that instead of a grid control, UIQ has a more general “scrollable container” control. This allows for arbitrary placement of GUI elements in a window of unlimited virtual size, and provides scrolling capability for viewing that virtual window piece by piece. We based our thumbnail grid code on this control, and were able to use the resulting code to implement two different screens: one for showing thumbnails of user created notes, and another for showing thumbnails of predefined stationary samples. The second screen is used for choosing the stationary for a new note. The graphic display of stationary thumbnails is a feature new to the P800 version of BugMe!. The Palm OS version listed the stationary templates by name only.

The next GUI issue that highlighted differences between UIQ and Series 60 was contact integration. As you will recall, the Series 60 implementation of contact integration required using the low-level Symbian API. All of this changed with the move to UIQ. There, we were able to use two mechanisms to seamlessly provide both the Contact choosing functionality and the phone dialing functionality. Choosing a contact became a very simple affair, using the class `CContactUIFindDialog`. We were able to remove all of the low-level contact database code. The second piece, dialing the phone, was actually moved out of BugMe!; instead, BugMe! transitions to the address book application, to a screen that shows the chosen contact’s details. There, the user simply clicks on the desired phone number (or email address, or other messaging info), and the address book app transitions to the phone app or other appropriate messaging app, with the user’s details filled in. While this kind of cross-application integration, called view switching, is also available to some degree under Series 60, UIQ further developed this technology, which it calls dynamic navigation link (DNL) into a centerpiece of its philosophy of GUI. It exposes a great deal of the functionality of its built-in applications, and encourages developers to take advantage of them.

This brings us to a rather important point about using DNLs, which must be considered seriously by any developer porting to or developing for the UIQ environment. As implemented, DNLs are generally a one-way street. That is, once you transition to the other application, there is no automatic return to the originating application when the user finishes with the target application. This is part of the UIQ philosophy, which rightly sees the phone user as usually performing a linear sequence of short, well-defined, tasks. However, an application writer who is concerned with their application’s flow has to take this into consideration when using DNLs.

In BugMe!, for example, the transition to the Address Book application is made from the alarm screen, and constitutes the user’s “response” to the alarm. This is an excellent transition point, because in a very meaningful sense the user is “done” with BugMe!, for now. However, the same could not be said were BugMe! to use the built-in Sound application to record and play voice annotations to reminders. In such a case, the user would transition to the Sound app in order to record or listen to the voice annotation, but then would have to re-invoke BugMe! from the Application Picker (or from one of the accelerator buttons) in order to continue using BugMe! – even if they just want to get another look at the reminder on the screen. This is one of the reasons we did not choose this path for implementing voice annotation under UIQ.

Under Series 60, view switching works somewhat differently. There, similar types of application transitions are available, but unlike UIQ, when a user is finished with the target application, control returns to the originating application. This is actually not the result of an underlying difference between Series 60 and UIQ. In fact, both use the same Symbian OS API to implement this functionality. Instead, it is the result of different conventions in the two systems which application writers are encouraged to follow.

In Series 60 the default behavior of applications is for them to terminate when you are done with them, usually via a click on the right softkey, which by convention is always assigned to some kind of a “Back” operation. When an application terminates, the Symbian OS View Server takes care of bringing to the foreground the last view that was on the screen before the currently terminating application was invoked. UIQ, on the other hand, has the convention that applications are not terminated by the user. The user simply goes on to the next application, by choosing from the application picker. Actual application termination typically occurs only when resources become scarce. At that point the operating system can ask applications to terminate, and according to convention, they should do so. Incidentally, in the case of BugMe!, the need to maintain the timer objects which are the basis for its alarm capability made us implement BugMe! to buck this convention.

When BugMe! for the P800 was developed, there was no API available for using the camera directly. Instead BugMe! took advantage of UIQ’s `CQikSelectMediaFileDialog` to allow the user to choose a photo (or other

image) from the full range of pictures stored on the phone. This was quite a departure from the Series 60 version's integral camera support, which enabled the use of the camera as one step in the wizard sequence of creating the reminder. Since BugMe! was released, Sony Ericsson has made available an API for using the camera from inside an application.

The vibrate API has also been made available, but was missing at the time that BugMe! was developed. The issue of profiles and ensuring BugMe! being silent, which we discussed in the context of Series 60, was much less problematic in UIQ. But this is not because of the availability of any more APIs. The difference is that the P800 has, across the bottom of the screen, a set of system functions that are always available to the user. One of them allows the user to silence the device, so that *all* sounds, regardless of the application producing them, will not be played. This solves a large part of the problem. Now that the vibrate API is available, the road is open to solve the rest of it.

5. Conclusion

In this article, we have touched on a large number of the issues which arose in porting BugMe! from Palm OS to both Series 60 and UIQ. There are more, and surely the developer of any particular application will find themselves encountering other challenges and opportunities. We think that it's worth the extra effort to rethink your application in light of the target platform before diving in. The secret to success is to understand the design philosophy and the API strengths of each platform. Understanding the design philosophy doesn't mean restricting yourself to convention. It means being able to make informed choices about when non-conventional design is worth it, and what you'll be trading in by bucking convention. Understanding the API strengths of the platform will help you mold and enhance your application in ways that will, for very little developer effort, make your application feel like a "native" app on the target device. At the same time, you'll be adding value. Symbian OS awaits your app. Good Luck!

Andy Weinstein, andyw@degel.com, is a Senior Applications Analyst at Degel Software Ltd., the "Software SWAT Team" – a consultancy of highly experienced software professionals covering a broad range of technologies, with a recent emphasis on handheld platforms. For more information, see <http://www.degel.com/>. BugMe! is available from <http://www.electricpocket.com/>

Want to be kept informed of new articles being made available on the Symbian Developer Network?

[Subscribe to the Symbian Community Newsletter.](#)

The Symbian Community Newsletter brings you, every month, the latest news and resources for Symbian OS.