

eSWT & Mobile Extensions Programming Guide

Functional Partitioning

eSWT is partitioned into two sets of function. The “Core” set contains the minimal function required to run basic applications. The “Expanded” set contains additional function which may be more appropriate for higher end devices with larger displays and more memory resources. Both of these sets are pure sub-sets of desktop SWT. Thus applications coded to use eSWT will also function on any desktop SWT implementation.

SWT Mobile Extensions is a set of function primarily targeted for mobile devices, although some widgets it provides may also be useful on desktop platforms.

Devices may carry these functional sets in various configurations or packaging schemes. For instance, low end devices may include a Core library and a Mobile library. Higher end devices may carry a converged library that contains all three sets of function. For this reason, eRCP workbench applications should always declare eSWT and Mobile Extensions dependencies by using “Import Packages” rather than “Required Plug-ins” statements in their manifest.

Generic application developers should recognize that the widest possible audience for their apps will be gained by coding solely to Core. If the app is targeted primarily for mobile devices, then Mobile Extensions can be used. Use of Expanded function may limit applications to running on higher end devices.

Device Normalization

Unlike desktop machines which all have large screens and pointer mechanisms, mobile devices come in a wide range of shapes and sizes and have a variety of input mechanisms. As much as possible, we would like to write applications that run *well* on any kind of mobile device. Note that there is a big difference between just running on a device and running well. Usability is a vital concern for mobile devices where environments vary and expectations for ease of use are very high. eSWT and Mobile Extensions attempts to normalize devices so that the application programmer does not have to do a lot of work to handle the differences among devices. It does this in two ways: implicitly, by providing a device’s native look and feel that a user is familiar with, and explicitly by providing mechanisms that abstract input and output through the actual device hardware.

Implicit normalization automatically provides some level of device adaptation by giving applications indirect access to a device’s native widgets. Since eSWT widgets are implemented using a platform’s native widgets, eSWT widgets appear and behave similarly to widgets in native applications. The end-user can recognize and interact with these widgets as they’re use to. The programmer gains these benefits simply by using eSWT widgets.

Explicit normalization is provided via specific mechanisms that a programmer is encouraged to use. These generally fall into two categories: organizing output on a display and handling different input mechanisms.

Display Normalization:

- 1) **Use of flow based layouts is strongly encouraged** – Layouts like RowLayout and GridLayout position widgets independently of screen size.
- 2) **Use of absolute coordinates is highly discouraged** – display sizes and aspect ratios can vary considerably. To guarantee that widgets are fully displayed, the programmer would have to calculate proper coordinates for each different display size.

Even though layouts help considerably in adapting to different screen sizes, a program that is desired to run well on all devices should also perform the following:

- 1) Check if the computed layout is larger than the available screen size, and if so add scroll bars to allow scrolling the content.
- 2) Check for high aspect ratios which may restrict layout or allow for additional content. For example on a mobile device with a long, narrow landscape display, content may be better displayed in two columns or in side by side panes.

Input Normalization:

- 1) **Use of Mobile Extensions Command feature is highly encouraged** – Commands are an abstraction that the Mobile Extensions library maps to a specific mechanism depending upon the device capabilities. This will usually be pointer driven menus or soft keys.
- 2) **Use of buttons is discouraged** – many devices do not have pointer devices. Therefore, buttons on these devices must be navigated to using jog controls or arrow keys and then selected. Jogging through numerous widgets is more time consuming and may also be cumbersome depending on the device controls. If buttons are highly desired, then the application can check via the MobileDevice class to see if the screen is a touch screen. This allows use of buttons when they can be easily selected and use of Commands or some other mechanism when they can not be.
- 3) **Use of menus is discouraged** – of minor consideration is that the width of the menu bar may be quite limited on some devices. It is difficult for your application to determine how many top level menu items can be supported. Menus may also be somewhat difficult to navigate on non pointer devices.

However, a more important reason not to use menus is that you may be bypassing the device's most natural or efficient input mechanism.

Commentary on selected widgets:

Browser – not all platforms support the Browser widget as underlying native support may not be available and emulating a browser is no easy chore. Therefore, applications should always catch SWTError exceptions when attempting to instantiate a Browser widget. Apps which are not completely dependent on the Browser widget may then provide an alternate UI mechanism. Apps which are dependent can display an error dialog explaining that the application is not fully supported on this platform.

Note: Currently, the Browser widget is not supported in eSWT for Series 80. Also, due to limited underlying support in Windows Mobile 2003, some Browser methods are not fully implemented at this time. Full support will be provided in a later release of eSWT.

Button – as noted above, buttons should be used sparingly unless the device supports a pointer mechanism.

CaptionedControl – Determining which control has focus can be difficult on mobile devices where lighting conditions are often less than optimal. (For instance, hairline width cursors in a text field can be very difficult to see even under good conditions.) This widget “caption” a control with a label that shows focus highlighting whenever the control has focus. With an entire label highlighted, it is easy to see where focus is.

Command – As noted above, this object provides a device independent abstraction for user input. It is highly recommended to use Commands for all generic apps.

ConstrainedText – Is a convenience widget which sets the initial input mode of a text field and also limits the characters than can be entered within the field.

DateEditor – Allows the user to select a date, time, time offset or duration in a platform specific manner. Constructor styles allow the program to choose FULL or COMPACT display modes, or rely on the platform default.

HyperLink – Displays a label in a style which indicates it can be selected. Selecting the label (containing valid link text) will cause an appropriate platform function to be launched. For instance, selecting an e-mail address link will launch the platform's e-mail program, selecting a URL will launch the platform's browser and selecting a phone number will cause the platform's dialer to dial that number. Note that not all platforms may have corresponding platform function. For instance, PDAs may not have a built in phone.

ImageLoader – Note that the ImageLoader class for the Windows Mobile 2003 platform does not currently support loading multiple images from an animated GIF file. Only the

first image will be loaded. Full support will be included in a future release. Until then, applications must load animation images manually. Image loading and saving on all platforms may be limited. Applications should be prepared to receive `ERROR_UNSUPPORTED_FORMAT` exceptions when using this API.

ListBox – This is a Model-View-Controller (MVC) style widget which displays a list of items in the most efficient/useful way for a given platform. The application may specify many hints as to how the list should be presented, but the details of the actual presentation are device dependent.

ListView – This is an advanced List widget which allows inclusion of an image with each list item. The ListView layout can also be changed at runtime in order to display more or fewer items depending on the user's preferences.

MobileDevice – This object allows the application to query information about the specific device the app is running on and be informed about device state changes as they occur. For instance, an event is sent when the device is closed, or a remote keyboard becomes available.

MobileShell – This widget allows applications to create a full screen window on a device, such as might be desired for a slide show or game. The window created can also dynamically change its trim to provide a window with more or less application area. MobileShell can also be used to poll for key state as commonly done within game execution loops.

MultiPageDialog – This dialog allows the application to show multiple pages which are displayed one at a time. It provides function similar to TabFolder in desktop SWT, but in a more platform dependent fashion. The more restricted nature of this dialog over TabFolder allows it to map more readily to native platform functionality.

QueryDialog – This dialog allows the application to query the user for a single piece of data and is provided as a convenience class.

SortedList – This list widget automatically sorts its items in ascending or descending order. In addition, the end user can filter the list. That is, by entering characters, items not containing that subset of characters will not be displayed in the list. Such function is useful on devices where scrolling through a list may not be as quick or easy to do as on a desktop.

TaskTip – This class provides an asynchronous indicator that a long running task is progressing. The indication is similar in purpose to a mouse-over generated *ToolTip* which provides a bit of extra information that is generally unobtrusive in nature. An application may open and close a TaskTip as desired.

TextExtension – Is basically the Text widget with some mobile specific features. It allows a program to set the semantic meaning of the text field so that platform specific

features may be utilized to aid text entry. For instance, if the field is intended to hold an e-mail address, the widget may access the system e-mail application to retrieve possible e-mail address completions that the user may select from. The initial input mode may also be set to a particular character set. The user can change the input mode as needed. Finally, the widget allows the setting of an initial casing mode, so that text can be entered in all upper case or one of several other modes. Again, the user may change the casing mode.

TimedMessageBox – This dialog shows one of several styles and audible patterns. It automatically closes after a short period of time, thus reducing the need for user interaction to dismiss the dialog. For instance, such a message may be useful to inform the user that someone wants to start a chat session.

eSWT Programming Tips

Disposal of Colors - You should dispose of the colors you create when there is no longer a need for that color. You should **not** create the color, set it in the GC, dispose the color, then attempt to use the GC. In this instance, you are disposing of the color before you are done using it.

Layouts – Standard SWT practice is to either call `layout()` or `pack()` on a container with a layout before opening the container. Otherwise, the layout calculation is not performed and elements may not be visible.

Commands – The Command constructor *type* parameter may only be used for mapping Commands to softkeys, thus *priority* values should be used as well as *type* for appropriate ordering when Commands are displayed within a menu.

Command activation – Remember that Commands may not be the only (or best) interaction method on all devices. Direct selection of items should also be supported. I.e. a list widget may support several Commands such as Open, Edit, and Delete. Direct selection (i.e. clicking) of a list item should invoke the default Command action (i.e. Open). In some cases, a double click action may also be supported (i.e. double clicking an item may invoke the Edit action).

PaintListeners – Listeners should be added before a window is opened. Otherwise, the window will not paint correctly when opened.

Key Events - Don't assume that there will be an elapsed time between key pressed and key released events. On a device with virtual keyboard these events may be sent back to back.

Virtual Keyboard – On PPC, the virtual keyboard covers the bottom of display. Do not place text fields requiring keyboard input in the bottom portion of a

window, or if you do, ensure that the window is scrollable so that the user may scroll the text field into view. Other platforms provide better mechanisms such as providing an output field integrated with the virtual keyboard so the user can see what is typed regardless of where the widget is positioned.

Screen Size – Don't assume that the screen will always be small. Some devices are capable of connecting to large displays, where a few widgets, expanded to take up the entire screen will look very odd. If going over a certain panel size would look odd, then limit your panel size, or adjust its content.

Limitations in eSWT M5 build:

On WM2003, the cascade indication arrow on a cascade menu item containing an image may be overwritten. A work-around for this problem is to set the menu image after the menu has been created.

On Win32 and WM2003, support for loading multiple images from a GIF file is not currently supported. Only the first image in a multi image GIF is loaded. Work-around is to create animations via a series of image files.

On Win32 and WM2003, saving an Image to an OutputStream (via ImageLoader) is not implemented. Work-around is to save image to a file instead.

On Win32 and WM2003, Browser.setText is not implemented. Work-around is to use a file URL to reference a static or dynamically built file.

On WM2003, some Browser control methods are not implemented. No work-around available.

On Series 80, Browser is not implemented. No work-around available.