Mobile UI Design & Development Guide

PowerServer Mobile® 2020

FOR WINDOWS

DOCUMENT ID: ADC50003-01-2020-01

LAST REVISED: March 25, 2020

Appeon Inc., 1/F, Shell Industrial Building, 12 Lee Chung Street, Chai Wan District, Hong Kong.

Contents

1 Audience

This book is written for PowerBuilder developers who want to develop native iOS and Android applications in PowerBuilder IDE. It recommends the best practices that a PowerBuilder developer should follow when utilizing the PowerBuilder UI components to design a great user interface and user experience for the native mobile application.

It is strongly recommended that the user carefully reads through and fully understands iOS Human Interface Guidelines and Android Design, especially the **UI Design Basics** and the **Design Strategies** parts of iOS Human Interface Guidelines, which describe the guidelines and principles that help the user design a superlative user interface and user experience for the mobile application.

This book, though summarizing some of the key design principles from that guidelines, will mainly focus on the best practices of PB development which when followed consistently, will make sure the application looks and feels like it was designed expressly for a particular mobile device.

2 Best Practices

2.1 Mobile apps vs. PB desktop apps

There are many differences between a mobile application and a PB desktop application. It is highly recommended that you read through iOS Human Interface Guidelines and Android Design, especially the **UI Design Basics** part of iOS Human Interface Guidelines, to get familiar with the basics of the mobile UI design. What are listed below are the most important differences that you must keep in mind when designing the user interface and user experience for the mobile application in the PowerBuilder IDE.

2.1.1 Fingers vs. Mouse

Mobile devices mainly use the human finger for input; desktop computers mainly use a mouse for input. Designing a user interface for interaction with the finger is very different than designing for mouse.

- First of all, you would need to design larger UI controls for a finger than you would for a mouse pointer in the desktop environment. For example, iOS recommends the comfortable minimum size of a tappable target is 44 x 44 points, which is approximately 176 x 176 PBUs.
- Secondly, a mouse can have a hover state, so tool tips and other roll-over effects can be used in the desktop application. Besides that, a mouse can have a right mouse button, so right-click popup menu can be easily supported. However, these design elements are less natural on a mobile application.

2.1.2 Virtual keyboard vs. real keyboard

A desktop application can take advantage of a real keyboard to help people type fast and correctly, and enable shortcut key functionalities, while a mobile application mainly uses a virtual keyboard which takes up the precious screen space, and is typo-prone, therefore, you should avoid using the virtual keyboard whenever possible, and try to design the user interface for selection rather than for typing, for example, use a ListBox, TrackBar, or spin control to replace a text field.

2.1.3 Screen Size

Compared to desktop applications, mobile applications are running on a more compact screen, therefore, their window size must be smaller, while the UI controls must be large enough to be easily tapped. This means that only the primary functionalities can be displayed at the screen at a time, the secondary functionalities should be hidden. This requires you to carefully design the application structure, the user interface, and the window navigation, especially for smartphones such as iPhone.

2.1.4 Memory management

Memory is more tightly constrained for the mobile device than it is for desktop computers, therefore, it is very important to build the application as memory efficient as possible. Here are some advices for PB developers to make their application more memory efficient:

- Do not retrieve (or cache) any more data in DataWindows or DataStores than is absolutely necessary. The more data is retrieved into DataWindow and DataStores the more memory is consumed. More data means more rows and/or columns.
- Use SDI instead of MDI. Then the user cannot have multiple windows open. The more windows open the more memory is consumed. If necessary, use the API to check the memory in use, and prompt or close some windows to release memory.
- Use stored procedures as much as possible. Then heavy calculations and data processing are not happening on the mobile device.
- Minimize use of global functions, global objects, global structures, etc. except when truly necessary. Global things are always resident in memory so it consumes memory.
- Whenever possible, release objects directly (by calling the PowerBuilder DESTROY statement) rather than auto-release them, because iOS does not support garbage collection.

2.1.5 Intermittent vs. reliable connectivity

By contrast to desktop PC which mainly uses wired networks, mobile devices use wireless networks exclusively due to its "mobile" nature, while wireless networks tend to be unstable or even not available when the device is travelling. Therefore, when you are designing for the mobile application, you may need to take the network connectivity into consideration, you can develop the mobile application with offline capabilities so that it can run against the local database if no network connection and then synchronize data with the server-side consolidated database when network is available, or even develop an offline mobile application which can run on the local device without needing network connections at all.

2.1.6 Orientation Change

Device orientation can change in the mobile device and people often expect to use their mobile devices in any orientation, therefore, you will need to consider how to support orientation change when you design the application.

2.1.7 Font

Fonts that are available in the mobile platform and the PC platform may not be exactly the same, because fonts are largely dependent on the platform manufacturers. You can choose from the fonts that commonly exist in both the mobile platform and the PC platform, such as Arial (recommended font), Courier New, Euphemia, Georgia, Trebuchet MS, Verdana, Times New Roman, etc.

Arial is the most recommended font to use in the PB application, because it looks very similar to Helvetica (the iOS default font) and Roboto (the Android default font). If you choose a font that is available in the PC platform but not available in the mobile platform, the font will be mapped to the default font (Helvetica in iOS or Roboto in Android) automatically at runtime. For the mapping rules, please see Font.

2.1.8 Color

The RGB color used in the PB application will be displayed as the exact same RGB color in the mobile application at runtime. But the system colors may not, because the Windows

system colors (such as ButtonFace) are different from the mobile system colors. If you have used or intend to use the system colors in the application, you may notice there are minor differences.

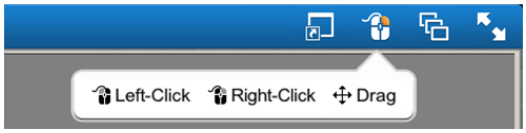
2.2 Event-handling model

The mobile OS event-handling model is significantly different from the one in PowerBuilder. Instead of delivering mouse and keyboard events, mobile OS delivers touch and gesture events. Therefore, the PB mouse and keyboard events will be mapped to the mobile touch and gesture events automatically at runtime, for example, the click event in PB is interpreted as the tap event in mobile OS, the double-click event in PB is interpreted as the tap-twice event in mobile OS. But not all of the PB mouse and keyboard events can be directly mapped to a mobile event, for example, the MouseMove event and the keyboard events (KeyDown and KeyUp) have no counterpart mobile events, you will need to understand how these events will be supported by PowerServer Mobile first, if you want to take advantage of these PB events to achieve what you want on the mobile device.

Some events such as right-click etc. are less natural in the mobile device and some events such as LButtonDown for the Drag event conflict with scrolling, therefore, these events should not be used. However, to keep compatible with the existing application logic, if these events are used, they will be preserved and triggered by using a special tool provided by PowerServer Mobile called **Assistive Touch Bar**, as shown below.

By default, the **Assistive Touch Bar** is hidden on the titlebar. To display it on the titlebar, you can call the of_setassistivetouchbtnvisible API. For details, please see the section called “of_setassistivetouchbtnvisible” in *Workarounds & APIs Guide*.

Figure 2.1: Assistive Touch Bar



The **Assistive Touch Bar** contains three modes (only one can be turned on, by default **Left-Click** mode is turned on):

- When **Left-Click** mode is turned on, if the finger touches down and moves within 500 milliseconds, scrolling takes place; if the finger touches down and then moves after 500 milliseconds, then either dragging takes place if drag(begin!) is called, or scrolling takes place; if the finger touches down and then lifts up without moving, the normal event sequence will be triggered. If you want dragging to take place without needing to hold for 500 milliseconds, you can turn on the **Drag** mode.

- When **Right-Click** mode is turned on, events will be triggered in the following order: when the finger touches down, RButtonDown is triggered, when the finger touches down and then moves, MouseMove is triggered, when the finger lifts up, RButtonUp is triggered.

- When **Drag** mode is turned on, dragging takes place immediately when the finger touches down a control and moves. Unlike the **Left-Click** mode, you do not need to hold for 500 milliseconds for dragging to take effect.

2.2.1 Message mechanism

PowerServer Mobile and PowerBuilder uses different message mechanisms which have different message queue. The message mechanism that PowerServer Mobile uses is mostly the mobile OS' mechanism which has the following differences from PowerBuilder's:

- The operation and event handling is concurrent.

  Suppose the Click event of a CommandButton executes a complex loop. The user clicks this button and then during the execution of the first click event, the user clicks the other controls in the window. In PowerBuilder, the other clicks will not be triggered even though the first click event is finished, while in PowerServer Mobile, the other clicks will be handled after the first click event is ended.

- The event handling is blocking.

  For example, the Click event of a CommandButton pops up a message box or executes a complex loop, the other events such as UI refresh (if StaticText is updated by the loop, it will not be refreshed immediately), Timer events etc. will not be handled.

- The event handling is asynchronous.

  Suppose the user right clicks a control or a window to display the PopMenu, and then operates on the menu item. In PowerBuilder, the sequence is pop up the menu, execute the menu item events, and then execute the remaining scripts in the RightClick event. While in PowerServer Mobile, the sequence is pop up the menu, execute the remaining scripts in the RightClick event, and then execute the menu item events.

2.3 Enhanced Mobile Controls and Functions

PowerServer Mobile generates the native mobile app with an intuitive, user-friendly, and modern user interface. The user interface is automatically adapted to the look and feel of the target device. This section lists the controls, objects, and functions that are enhanced or added by PowerServer Mobile to achieve the typical mobile-style user interface. The enhanced controls and objects include Window, Menu, Toolbar, TreeView, etc.

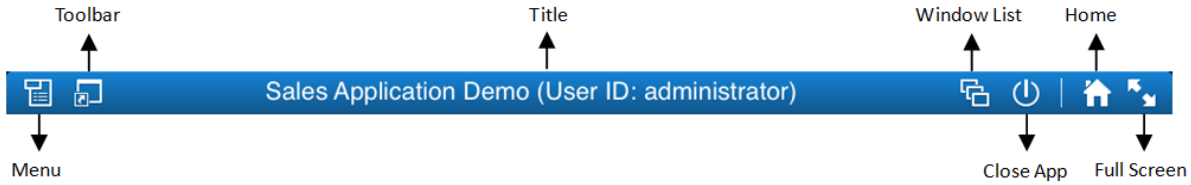
In addition, though not described in detail below, due to the difference between desktop computer and mobile device, most PB controls are adjusted slightly to be used easily on the mobile device, for example, the row height of TreeView, DropDownListBox, ListView, etc., the tiny triangle icon for DatePicker, DropDownListBox, EditMask, etc., and the checkbox of TreeView, ListView etc. are automatically adjusted with tappable size.

2.3.1 Window title bar, menu, & toolbar

The PB desktop application usually has a title bar which displays the window title, the Max, Min and Close icons, and other adornments; while the mobile application calls this a navigation bar, as it usually contains navigation buttons such as the Back icon, the Next icon. PowerServer Mobile combines the title bar and the navigation bar into one (and calls it title bar in all of user guides), by placing the window title in the middle and the icons on two ends.

Some of the icons such as the **Window List** icon, **Close App** icon, **Full Screen** icon can be shown or hidden by calling the PowerServer Mobile API. For details, refer to Section 1.4, “Mobile Device API” in *Workarounds & APIs Guide*.

Figure 2.2: Window title bar



1. The **Menu** icon will display the menu items in a treeview hierarchical structure. Notice that the treeview layout is automatically handled by PowerServer Mobile.

2. The **Toolbar** icon will display the toolbars by row(s). Considering the limited space, only icons will be shown, text will not.

3. The window title will be displayed at the center of the title bar and in the font size of 22 points by default. If the title text cannot display completely, the font size will be automatically reduced, until it goes down to the minimum font size 14 points; if the text still cannot fit into the title bar, then the text will be truncated.

4. The **Window List** icon will list the opened window(s) with the Max, Min and Close icons. You can navigate to any active window in any order, or close, minimize or maximize any window.

5. The **Close App** icon will exit the app from the background.

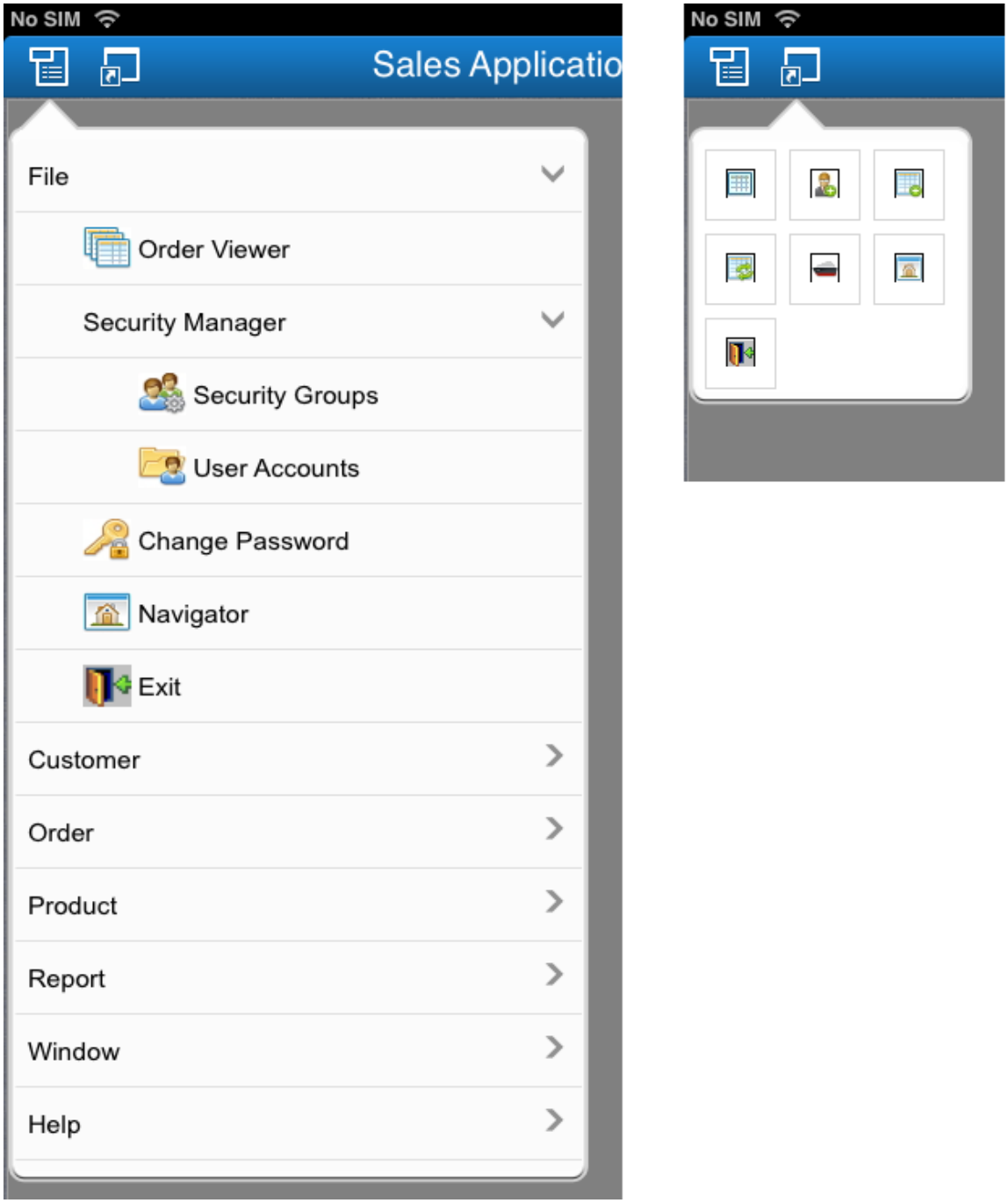
6. The **Home** icon will return you to the Appeon Workspace home page, and the app will run in the background.

7. The **Full Screen** icon will hide the title bar, to allow more space for the window.

Window menu and toolbar are not usual in mobile apps as is in PB apps, probably because they take up the precious screen space and are tiny and placed closed in the original layout therefore cannot be easily manipulated by a finger. However, they are very handy when navigating between windows, especially in the MDI window. Therefore, if any menu or toolbar is used in the PB application, they will be preserved on the PowerServer mobile application, and will be laid out differently in order to make them easy to use by fingers.

Window menu and toolbar on an iPad:

Figure 2.3: Menu & toolbar on iPad

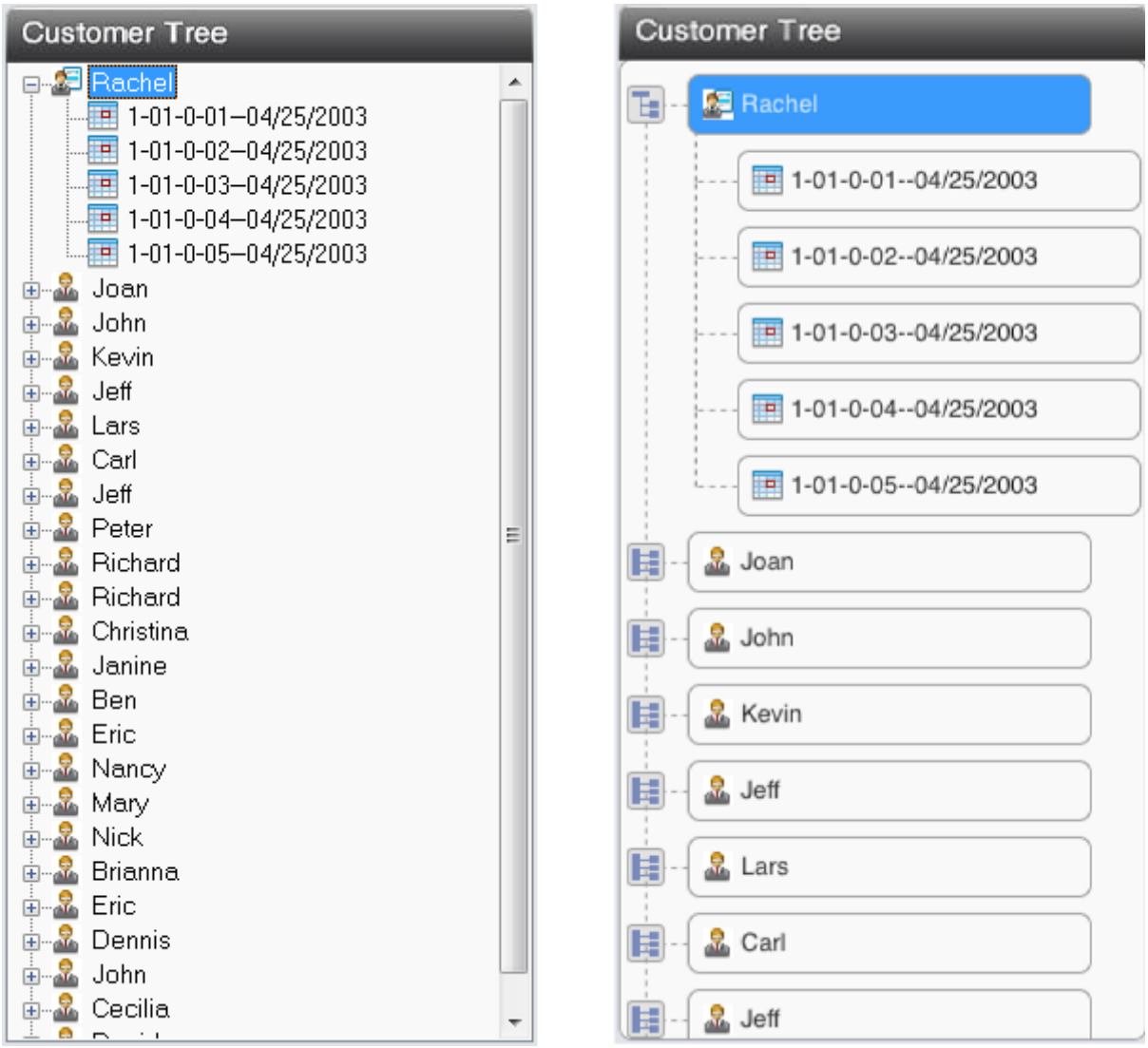


2.3.2 TreeView

The TreeView control is automatically adjusted to be easily tapped by fingers in the mobile device. The plus and minus signs to expand and collapse the tree are automatically replaced with larger icons that are easy to touch by fingers. The height of the treeview item is also automatically increased to be 44 points.

PB-style TreeView (left) VS. mobile-style TreeView (right):

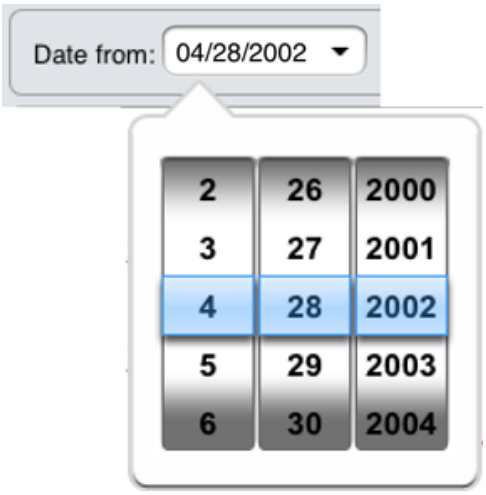
Figure 2.4: PB-style TreeView VS. mobile-style TreeView



2.3.3 Mobile-style Picker

To get a date/time picker exactly like the native mobile Date and Time Picker, you can set the EditMask control to a date/datetime/time mask and then enable its Spin property.

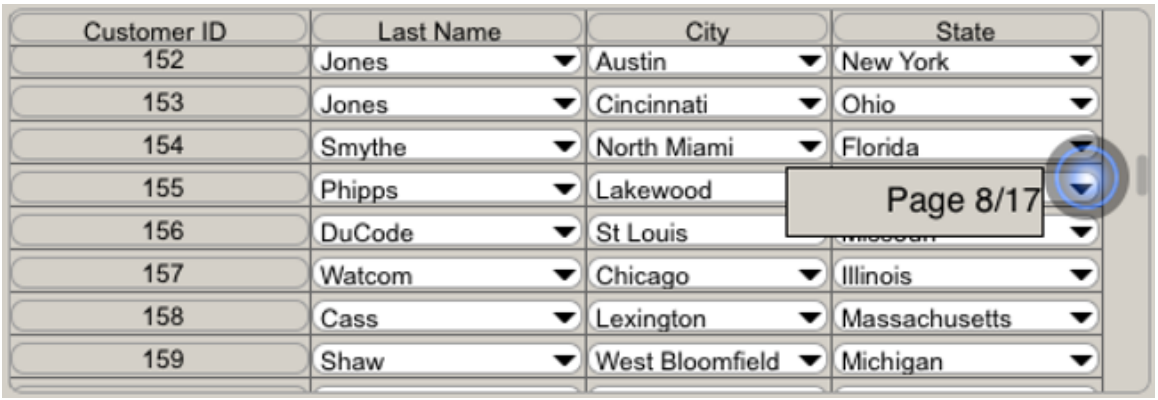
Figure 2.5: Date picker



2.3.4 Speedy scrollbar

Speedy scrollbar is a tool added by PowerServer Mobile to help scroll quickly through a long list of data records. In the DataWindow control, the user can use the speedy scroll bar to scroll through a large amount of data page by page, instead of row by row. The speedy scroll bar automatically appears when the user swipes up or down in a DataWindow with more than 10 pages of data. The speedy scroll bar will disappear if it is not used for 2 seconds.

Figure 2.6: Speedy scrollbar



2.3.5 HScrollBar and VScrollBar properties

The horizontal scrollbar and vertical scrollbar for controls will be automatically replaced with the mobile-style scrollbar, instead of the Windows-style scrollbar.

2.3.6 SetPointer function

There is no mouse pointer in the mobile device like the desktop computers, therefore most of the pointer shapes that can be displayed in the desktop computer will not be displayed in the mobile device, except for "*HourGlass*". When it is set to "*HourGlass*", the mobile-style activity indicator will display in the mobile device to indicate the busy state. The activity indicator will also automatically appear every time when the client calls the server, you can set to empty string SetPointer("") to not automatically display the activity indicator.

Figure 2.7: Activity indicator

2.3.7 Mobile APIs

In addition to controls, PowerServer Mobile also added a bunch of functions to call the mobile native APIs, such as functions to call camera, device, GPS, bar code, etc. For details, refer to Section 1.4, “Mobile Device API” in *Workarounds & APIs Guide*.

2.3.8 Android Back button

To support the Back button that the Android device provides, a user event named "appeon_android_back" is added to the Application object. This event will be triggered when the Back button on the Android device is pressed. For details, refer to Section 3.3.1.1, “Application” in *Supported PB Features for PowerServer Mobile*.

2.4 Mobile UI Design

When you take the first step in developing a mobile application in PowerBuilder or start to adjust an existing PowerBuilder application for mobile deployment, the first question you will probably ask is: how should I design the UI layout so it can automatically fit with various mobile devices. This section will give high-level guidelines to design the UI for the mobile screen, and introduce the UI resizing tools to automatically adjust the layout to fit with the mobile screen.

2.4.1 Mobile UI design guidelines

Below are guidelines for designing a mobile-friendly UI for a compact screen:

- Maximize the window size

  Mobile applications are usually displayed in full screen, so as to efficiently use the valuable space of the mobile screen. The simplest way to make a PoweBuilder window display in full screen in various mobile devices is to set the WindowState of the window from the normal (default) to maximized. However, the controls in the window will remain in the same location and size. You will need to adjust the location and size of a control using the mobile UI resizing tools provided by PowerServer.

- Change the window background to white

  Normally the background of a mobile application is white or some other colors, but rarely gray. Therefore, it is recommended you change the background color of the PowerBuilder window from gray (default) to white or other mobile-friendly color.

- Fill with appropriate number of controls

  Consider carefully for what to include on the window, so as not to crowd the window or leave valuable space unused.

- Use font size between 12 and 17 points

The default font size in PowerBuilder is 9 or 10 points which is way too small on a mobile screen. You'd better use a font size between 12 and 17 points.

- Set the touchable target to 176 * 176 PBU (which is 44 * 44 points)

  Fingers rather than mouse are used to operate the application on a mobile device. Therefore, it is important to design the target large enough to be easily touched by fingers. Touchable targets include buttons, text boxes, radio buttons, check boxes, list boxes, row header and footer, data rows, etc. Set the height of these targets to 44 points (or 32 points by minimum).

- Take advantage of the mobile UI resizing tools provided by Appeon.

  Appeon PowerServer provides two mobile UI resizing tools to automatically resize and move the controls to fit with the mobile screen: the Appeon Resize APIs (**recommended**) and the AEM | Mobile UI Resizing tool. The Appeon Resize APIs enable the controls (including those in the DataWindow object) within the user object, window, and tab to have the auto-resizing feature with only a few lines of script.

  The AEM Mobile UI Resizing tool automatically scales the window as well as its controls and font according to the predefined rules for all different kinds of screen sizes. The AEM Mobile UI Resizing tool does not support resizing controls in DataWindow objects.

2.4.2 Appeon Resize APIs

PowerBuilder uses absolute layouts, which means, the controls within the window will not change its position and size when the screen size changes. This may create UI problems when a window is displayed on different screen sizes, especially on mobile screens which are much smaller than PC screens. One window that looks perfect on the tablet may be too large to display completely on the smartphone. Even for smartphones which have slightly different screen size, it is difficult to design a window that fit perfectly with all these different sizes.

Using the Resize PowerScript function or the PFC resize service may be a way out, however, it would be some work to write code to control the size and location of each individual control within the window. Considering this, Appeon extends the functionality of Resize PowerScript function and the PFC resize service to greatly simplify coding and automate most of the tasks.

The extended functions is packaged into an object called eon_appeon_resize in the Appeon Workarounds PBL under the PowerServer installation directory (for example, C:\inetpub \wwwroot\appeon\developTempFile\appeon_workarounds\appeon_workarounds.pbl). The eon_appeon_resize object provides you the greatest control and flexibility on the resizing of the window, controls, user objects, and font; it also allows you to dynamically change the resize behavior for a particular window/user object/control at runtime.

For more information about the eon_appeon_resize object, refer to Section 1.3.12, “Appeon Resize Object” in *Workarounds & APIs Guide*.

2.4.3 AEM Mobile UI Resizing tool

AEM Mobile UI Resizing tool allows the user to specify the screen size of different mobile devices and define the resizing rules accordingly. Once set, it enables the window and all

controls (except for controls in DataWindow objects) as well as fonts within the window to proportionally scale up or down to fit with the mobile screen. For detailed instructions, refer to Section 4.5, “Mobile UI Resizing” in *PowerServer Configuration Guide for .NET* or PowerServer Configuration Guide for J2EE.

Below is comparison of Appeon Resize APIs (recommended) and AEM | Mobile UI Resizing tool:

- UI effect

  AEM Mobile UI Resizing tool resizes all controls despite the actual needs. Some controls may look ugly after resized. Appeon Resize APIs divides the controls into resizable controls and unresizable controls, according to some advanced UI layout tools such as Java Swing, so it produces better UI effect.

- Control and font resize

  AEM Mobile UI Resizing tool will resize all controls (except for controls in DataWindow objects) as well as fonts. Appeon Resize APIs will resize the resizable controls only. The unresizable controls will not change the size. Fonts will not be resized either.

- Code change

  AEM Mobile UI Resizing tool requires no code change. Appeon Resize APIs requires users to write a few lines of script in the target window. If there are many windows and the window is not inherited, then you would need to write scripts in every window.

- Configuration

  AEM Mobile UI Resizing tool requires the user to specify the screen size of each mobile device and define the resizing rules accordingly, while Appeon Resize APIs does not.

- Performance

  For windows with many controls, AEM Mobile UI Resizing tool delivers better performance than Appeon Resize APIs.

2.5 Window for iOS Screens

2.5.1 Unit conversion

PowerBuilder and iOS use different measurement systems to produce a consistent size of output that is device independent. In PowerBuilder, size is measured in PowerBuilder Units (PBUs), while in iOS, the measurement unit is Points (**Notice** that Point and Dot are used interchangeably in both Appeon documentation and product UI).

It is important for users to understand the concept of Point and how many Points in the width and height of each device type. Compared to Point, people are more familiar with Pixel, the measurement of the display resolution. In the iOS coordinate system, there is a mapping relationship between Pixel and Point, which is one Point equals to one Pixel on a standard-resolution screen, while one Point equals to two Pixels on a high-resolution screen (such as Retina Display). For details, please read the relevant page at Drawing and Printing Guide for

iOS. Now understanding the relationship between Point and Pixel, you can easily figure out how many Points in the width and height of each device screen, as shown in the table below.

Table 2.1: Device screen size

Device	Display Resolution (in Pixels)	Screen Size (in Points)
iPad 2	1024 x 768 pixels	1024 x 768 points
iPad 3/4	2048 x 1536 pixels	1024 x 768 points
iPad mini	1024 x 768 pixels	1024 x 768 points
iPhone 4/4S or iPod Touch 4	640 x 960 pixels	320 x 480 points
iPhone 5/5C/5S or iPod Touch 5	640 x 1136 pixels	320 x 568 points
iPhone 6	750 x 1334 pixels	375 x 667 points
iPhone 6 Plus	1080 x 1920 pixels	414 x 736 points

Now that you understand how many Points in each device type, you can use the following formula to convert from Point to PBU:

Point-to-PBU conversion:

In the English environment:

In the X axis: PBUnit = (*PointX* x 6144 + 96 x 7) / (96 x 14)

In the Y axis: PBUnit = (*PointY* x 768 + 96) / (96 x 2)

In the Japanese environment:

In the X axis: PBUnit = (*PointX* x 6144 + 96 x 8) / (96 x 16)

In the Y axis: PBUnit = (*PointY* x 2048 + 96 x 3) / (96 x 6)

Notes:

PointX indicates the number of Point in the X axis.

PointY indicates the number of Point in the Y axis.

See also

Pixel-to-PBU conversion:

Since you can also easily get the display resolution (in Pixels) and PPI from the device specification, you may use the formula to convert from Pixel to PBU:

In the X axis: PBUnit = (*PixelX* x 6144 + *PPI* x 7) / (*PPI* x 14)

In the Y axis: PBUnit = (*PixelY* x 768 + *PPI*) / (*PPI* x 2)

Notes:

PixelX indicates the number of pixels in the X axis.

PixelY indicates the number of pixels in the Y axis.

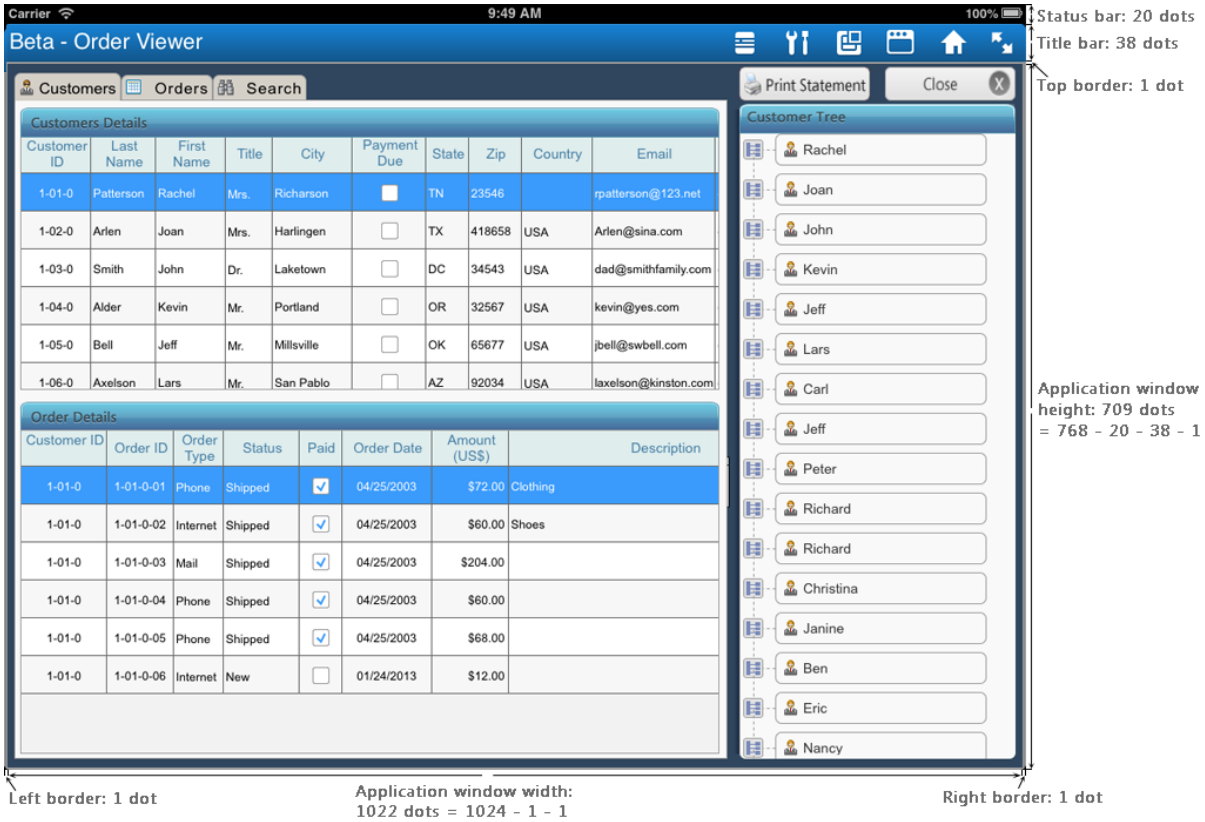
PPI stands for Pixel Per Inch, which is 132 for iPad 2, 264 for iPad 3/4 and iPad mini, 326 for iPhone 4/4S/5/5C/5S/6 and iPod Touch 4/5, and 401 for iPhone 6 Plus.

2.5.2 Window Size

Let's now have a close look at the layout of an iOS-based device screen (taking iPad 2 as example). It will help us calculate the available screen space for the application window.

In Landscape orientation:

Figure 2.8: Landscape view of an iOS-based screen

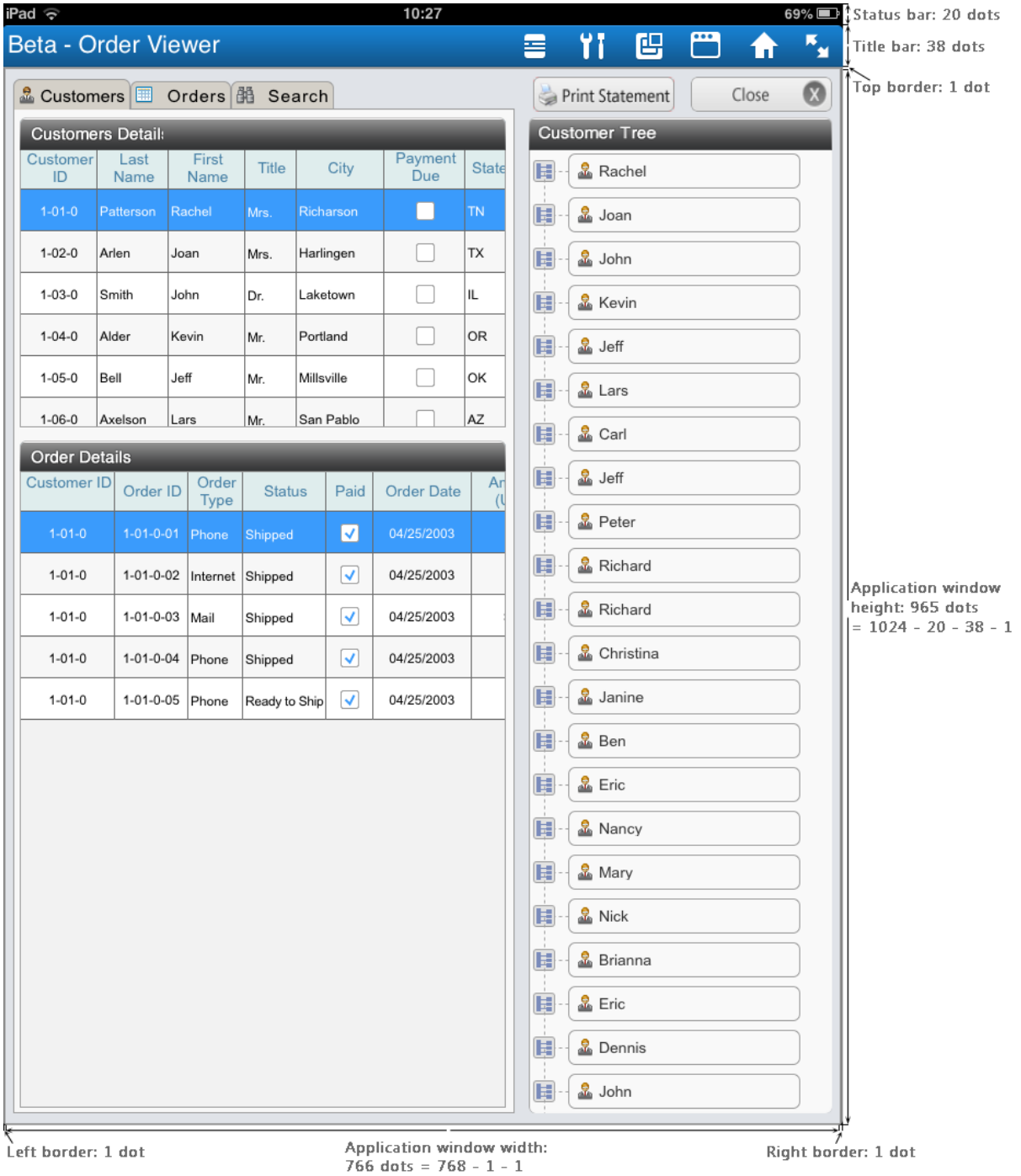


From the above graph, we understand that window size is not equal to the screen size. The application cannot use the entire screen space, because iOS adds a status bar (20 points in height) and Appeon Workspace adds a title bar (38 points in height), a top border (1 point in height), a left border (1 point in width), and a right border (1 point in width) around the application. Therefore, in Landscape orientation, the space available for the application on iPad 2 is 1022 * 709 (points).

Now let's use the formula to convert it to PBUs, which is 4672 * 2836 (PBUs). This means, if you set the window size to 4672 * 2836 PBUs and then set the window state to **Maximized**, the window will exactly fill up the available screen space in landscape orientation on iPad 2.

In Portrait orientation:

Figure 2.9: Portrait view of an iOS-based screen



From the above graph, the available screen space for an application window in Portrait orientation is 766 * 965 (points). When using the formula to convert it to PBUs, we get 3502 * 3860 PBUs.

Now that we understand the available screen space for application window, we can calculate the window size for the other iOS-based devices. The following table lists the window size (in full screen view) on iPad, iPhone, & iPod touch, measured in both PBUs and Points.

Table 2.2: Window size on iPad, iPhone & iPod touch

Device	Width to deduct	Height to deduct	Landscape				Portrait			
			Width (pt)	Height (pt)	Width (PBU)	Height (PBU)	Width (pt)	Height (pt)	Width (PBU)	Height (PBU)
iPad 2	-(1+1)	-(20+38+1)	1022	709	4672	2836	766	965	3502	3860
iPad 3 or 4	-(1+1)	-(20+38+1)	1022	709	4672	2836	766	965	3502	3860
iPad mini	-(1+1)	-(20+38+1)	1022	709	4672	2836	766	965	3502	3860
iPhone 4 or 4S	-(1+1)	-(20+38+1)	478	261	2185	1044	318	421	1454	1684
iPhone 5	-(1+1)	-(20+38+1)	566	261	2587	1044	318	509	1454	2036
iPod touch 4	-(1+1)	-(20+38+1)	478	261	2185	1044	318	421	1454	1684
iPod touch 5	-(1+1)	-(20+38+1)	566	261	2587	1044	318	509	1454	2036

Notes:

- Width to deduct: - (Left Border + Right Border)
- Height to deduct: - (iOS Status Bar + Title Bar + Top Border [+ MicroHelp Status Bar])

  MicroHelp Status Bar exists in the MDI frame only when the window type is set to **MDIHelp**. The height of MicroHelp Status Bar is 20 points.
- By setting the window to the above width and height and setting the window state to **Maximized** in the PB IDE, the window will exactly fill up the available screen space on the target device at runtime. By setting the window state to **Maximized**, the window title bar will merge into the Appeon Workspace title bar at runtime, otherwise it will display under the Appeon Workspace title bar, in such case you will need to deduct both the Appeon Workspace title bar (38 points) and the window title bar (38 points) in the height direction when calculating the window size.

2.5.2.1 Auto-fit to Screen

Except for setting the window size to fixed values which will exactly fill up the screen space, you can also take advantages of the UI resizing tools to enable the window to automatically fit with the screen. These UI resizing tools are described in Mobile UI Design.

2.5.3 Window for iPad & iPhone

Generally speaking, you can create different sets of windows specific to iPad and iPhone, or create one set of windows for both iPad and iPhone. Our recommendation is creating two sets of windows: one for iPad, and the other for iPhone, though the window number doubles, it is easier to control for developers.

2.5.3.1 Different sets of windows

As iPad and iPhone have very different screen sizes, our recommendation is designing windows for iPad and iPhone respectively. For example, you can design a window w_iPhone for iPhone, and a window w_iPad for iPad, and open the window according to the device type; you can call the interface of_getDeviceType to get the device type at runtime.

Even for the same device type, as screen size changes greatly when orientation changes, we recommend designing windows for Landscape and Portrait respectively. For example, you can design several windows such as w_iPhone_Landscape, w_iPhone_Portrait, w_iPad_Landscape, and w_iPad_Portrait, and you can call the interface of_getOrientation to get the device orientation at runtime.

2.5.3.2 Same set of windows

Instead of designing two sets of windows and opening the window according to the device type at runtime, you may want to create just one set of windows for all devices and adjust the window and control size at runtime via the Resize event. However, this would be complicated in some cases and would require you to consider every single situation that causes window to resize, including:

1. Resize caused by the user manually resizing or maximizing a window at runtime if the window is set to support resize;

2. Resize triggered by script. For example, in business logic, there may be situations when a condition is met, the window size needs to change accordingly.

3. Resize occurred when window is created. This happens only one time. Unlike PC where window size is fixed, there could be different window sizes on mobile devices, even if it is the same PowerBuilder window. Different height and width could be passed to the resize event, because the device to run the application may be an iPhone, or an iPad, in landscape or portrait mode.

4. Resize triggered by orientation changes. Unlike PC, mobile devices support orientation changes between landscape and portrait, and the size differs greatly.

All these different situations add up the complexity of UI adjustment. Take a simple example here, you may have a request that the window should not respond to the Resize event or the layout of controls should not change whenever the user resizes a window, but the window should support orientation changes between landscape and portrait. But apparently, there is no way to exactly figure out under what situation the Resize event is triggered and then process the event accordingly.

To try to reduce the complexity somewhat, we provide the interface of_getOrientation to determine whether the current device is in landscape or portrait view; and we also provide the orientation change event which will be triggered when the device orientation changes.

2.5.4 Window orientation change

There are several ways to support orientation change, such as using the PowerBuilder Resize event, or calling the PowerServer Mobile APIs. For detailed usages about the APIs, refer to Section 1.4, “Mobile Device API” in *Workarounds & APIs Guide*.

2.6 Control

Most of the existing PowerBuilder controls can be used and deployed to the mobile device, except for RichTextEdit, Animation, InkEdit, InkPicture, and OLE. You can use the PowerBuilder controls in the mobile application, just like how you use them in the PowerBuilder application, but keep in mind to set proper size for them and leave spacing between them. Apple recommends the target area to be average finger-tip size which is 44*44 points (approximately 176 * 176 PBUs). This is the ideal size we should strive for, but definitely this is no iron rule. You may find other practical rules for the real situations, such as set the control size to 44 points high or wide and reduce the other dimension to no smaller than 30 points (approximately 87 PBUs) if space is really limited, or leave space or padding between controls to make the target area closed to this size.

2.6.1 Control Size

Most PowerBuilder controls such as RadioButton, CheckBox, SingleLineEdit, StaticText, whose default sizes are appropriate for finger tapping on the mobile device, therefore, you can directly use them at the default size, but remember to leave enough Spacing between them.

For some controls such as CommandButton, DatePicker, and EditMask, whose default sizes are too small to be easily tapped by the finger, it is very important to remember to set proper size for them. The following table lists these small controls and recommends the minimum height. The recommended minimum height mainly comes from the default height of the Cocoa Touch controls, for example, a Rounded Rectangle Button is 37 points high by default.

Table 2.3: Small controls and recommended minimum heights

PB Controls	Height (in Points)	Height (in PBUs)
CommandButton	37	148
DatePicker		
EditMask		

A general rule of thumb is for controls whose size can be set in PowerBuilder IDE, you should set proper size for them (44 points = 176 PBUs is recommended), for controls whose size cannot be set in PowerBuilder IDE, PowerServer Mobile will take care of this for you by automatically setting proper size for them at runtime.

2.6.1.1 Control width for text display

The same font may be displayed at different width under different platforms, therefore, it is recommended that you leave extra space (for example, 20 to 30 PBUs in width) in the control for the text to display properly, otherwise, the text may go over the control border or display in the new line.

2.6.2 Row Height

The default DataWindow row height is fine for a mouse clicking, but is way too tiny to be tappable for a finger, therefore, you should always remember to manually set proper row height in PowerBuilder IDE if you want to allow users to manipulate data easily in your

application on the mobile device. The minimum height of a Table View cell of Cocoa Touch is 44 points, and a Table View header or footer is 22 points.

Table 2.4: The minimum height of a Table View cell of Cocoa Touch

PB Controls	Height (in Points)	Height (in PBUs)
DataWindow: Header height & Footer height	22	88
DataWindow: Row height	44	176

The same problem exists for the row height of the items in a ListBox, DropDownListBox, ListView, or TreeView, but unlike DataWindow row height, they cannot be set in PowerBuilder, therefore, PowerServer Mobile will address this problem and will set the row height to 32 points (approximately 128 PBUs) or above automatically at runtime.

2.6.3 Spacing

It is important to leave spacing between UI controls, especially if the control is smaller than the recommended size. Leaving enough spacing between controls ensures they can be easily tapped. If you create smaller control or place controls too close together, people must aim carefully before they tap and they are more likely to tap the wrong control.

A general rule of thumb is the closer you place buttons together, the larger those buttons should be.

2.7 DataWindow presentation styles

Almost all of the DataWindow presentation styles are supported: CrossTab, Composite, Freeform, Graph, Grid, Group, Label, N-Up, Tabular, TreeView, and RichText, except for OLE.

2.8 Unsupported PB Properties

Some PB properties cannot be supported or are not necessary to support. For example, mobile devices usually do not have a real keyboard, so cannot take advantage of the keyboard to speed up or facilitate user operations, therefore, the PB properties related with keyboard keys including arrow keys, shortcut keys etc. will not be supported in the mobile application. Such PB properties include TabOrder, TabStop, Default, Cancel, DeleteItem, ExtendedSelect etc.

For a complete list of unsupported features, please refer to Supported PB Features for PowerServer Mobile.

2.9 Font

Below are guidelines for using font in the mobile UI:

- Font size -- Letters in the same font size will be displayed smaller on the mobile device screen than on a desktop computer, because the physical screen size of a mobile device is smaller than on the desktop computer. You must use a larger font to ensure legibility. In general, 14 - 17 point is the recommended font size, and 12 point is the smallest font size that can be easily read.

- Font style -- Font styles such as Bold, Italic, Underline (scarcely used) etc. may be supported depending on the mobile OS fonts.
- Font color -- The mobile OS default font color is Black.
- Font types -- There are a bunch of fonts commonly supported by both iOS and PowerBuilder, such as Arial (recommended font), Courier New, Euphemia, Georgia, Trebuchet MS, Verdana, Times New Roman etc. The iOS default font Helvetica is not available in PowerBuilder, we recommend you use Arial which exists in both iOS and PowerBuilder and looks very similar to Helvetica. If you use a font that is not available in iOS, it will be displayed as Helvetica in iOS.

  Android is an open platform and the available fonts in Android are mostly dependent on the device manufacturers, so the fonts may vary on different devices. The default font in Android is Roboto which looks very similar to Arial and Helvetica. If you use a font that is not available in Android, it will be displayed as Roboto in Android.

  For the mapping rules between Windows fonts and iOS fonts, please see the table below:

Table 2.5: Font Mapping Rules

Windows Font	iOS Font
Arial	ArialMT
	Arial-BoldMT
	Arial-ItalicMT
	Arial-BoldItalicMT
Baskerville Old Face	Baskerville
	Baskerville-Bold
	Baskerville-Italic
	Baskerville-BoldItalic
Courier New	CourierNewPSMT
	CourierNewPS-BoldMT
	CourierNewPS-ItalicMT
	CourierNewPS-BoldItalicMT
Courier	Courier
	Courier-Bold
	Courier-Oblique
	Courier-BoldOblique
Euphemia	EuphemiaUCAS
	EuphemiaUCAS-Bold
	EuphemiaUCAS-Italic
Georgia	Georgia
	Georgia-Bold
	Georgia-Italic

Windows Font	iOS Font
	Georgia-BoldItalic
Palatino Linotype	Palatino-Roman
	Palatino-Bold
	Palatino-Italic
Times New Roman	TimesNewRomanPSMT
	TimesNewRomanPS-BoldMT
	TimesNewRomanPS-ItalicMT
	TimesNewRomanPS-BoldItalicMT
Trebuchet MS	TrebuchetMS
	TrebuchetMS-Bold
	TrebuchetMS-Italic
	Trebuchet-BoldItalic
Verdana	Verdana
	Verdana-Bold
	Verdana-Italic
	Verdana-BoldItalic
Any Other Fonts	Helvetica

2.10 Image & Icon

2.10.1 Supported Image Format

PowerBuilder, iOS, Android, and PowerServer Mobile supports different types of image files. **PNG** and **JPG** are the recommended image types for PowerServer Mobile. Though BMP and ICO are supported by PowerServer Mobile, they are not recommended to use because they are Windows-platform specific and may have problems to display in iOS or Android.

Table 2.6: Image Types

Format	File Extensi	PB	iOS	Android	PowerServer Mobile
Portable Networks Graphics	.PNG	Supported	Supported	Supported	**Supported & Recommended**
Joint Photographic Experts Group	.JPEG or .JPG	Supported	Supported	Supported	**Supported & Recommended**
Static Graphic Interchange Format	.GIF	Supported	Supported	Supported	Supported

Format	File Extensi	PB	iOS	Android	PowerServer Mobile
Animated GIF files	.GIF	Supported	Supported	Supported	Not Supported
Windows Bitmap Format	.BMP or .BMP	Supported F	Supported	Supported	Supported
Windows Bitmap Format	.RLE	Supported	Not Supported	Not Supported	Not Supported
Windows Icon Format	.ICO	Not Supported	Supported	Not Supported	Supported
Windows metafiles	.WMF	Supported	Not Supported	Not Supported	Not Supported
Tagged Image File Format	.TIFF or .TIF	Not Supported	Supported	Not Supported	Not Supported
Windows Cursor	.CUR	Not Supported	Supported	Not Supported	Not Supported
XWindow bitmap	.XBM	Not Supported	Supported	Not Supported	Not Supported

2.10.2 Image & Icon Size

You can use images or icons in various areas such as menu, toolbar, tab bar, ListView, TreeView etc. Make sure you prepare the images and icons in the recommended size so they look good and perform well in the mobile device.

Table 2.7: Image & Icon size for iOS and Android devices

Icon	Size for iPad (in Pixels)	Size for iPhone & iPod Touch (in Pixels)	Size for Android Screen* (in Pixels)
App icon on the mobile device	72 x 72 (144 x 144 for retina display)	57 x 57 (114 x 114 for retina display)	48 x 48
App icon for display in Apple App Store or Google Play	512 x 512	512 x 512	512 x 512
App icon for display in Appeon Workspace	86 x 86 (172 x 172 for retina display)	86 x 86 (172 x 172 for retina display)	86 x 86
Menu icon	20 x 20 (40 x 40 for retina display)	20 x 20 (40 x 40 for retina display)	20 x 20
Toolbar icon	20 x 20 (40 x 40 for retina display)	20 x 20 (40 x 40 for retina display)	20 x 20
Tab bar icon	20 x 20 (40 x 40 for retina display)	20 x 20 (40 x 40 for retina display)	20 x 20
ListView icon	30 x 30 (60 x 60 for retina display)	30 x 30 (60 x 60 for retina display)	20 x 20

Icon	Size for iPad (in Pixels)	Size for iPhone & iPod Touch (in Pixels)	Size for Android Screen* (in Pixels)
TreeView icon	20 x 20 (40 x 40 for retina display)	20 x 20 (40 x 40 for retina display)	20 x 20
PictureListBox	20 x 20 (40 x 40 for retina display)	20 x 20 (40 x 40 for retina display)	20 x 20
DropDownPictureListBox	20 x 20 (40 x 40 for retina display)	20 x 20 (40 x 40 for retina display)	20 x 20

* For Android devices, to create an icon for different densities, you should follow the 2:3:4:6:8 scaling ratio between the five primary densities (MDPI, HDPI, XHDPI, XXHDPI, and XXXHDPI respectively). For example, consider that the size for an app icon is specified to be 48x48 px. This means the baseline (MDPI) asset is 48x48 px, and the high density (HDPI) asset should be 1.5x the baseline at 72x72 px, and the x-high density (XHDPI) asset should be 2x the baseline at 96x96 px, and so on.

2.10.3 Guidelines for designing Images & Icons

Beautiful, compelling icons and images are a fundamental part of the user experience. Please follow Icon and Image Design in *iOS Human Interface Guidelines* and Iconography in *Android Design* to design good-looking images and icons.

Index