Pocket Tunes

Skin Design Reference Manual

For Pocket Tunes 2.1 and later

with PocketSkin™ Technology

Last Updated: 02/06/07

Contact skins@pocket-tunes.com for support

Introduction

Welcome to Pocket Tunes with PocketSkin™ technology. This section will introduce you to Pocket Tunes and PocketSkin™ technology and prepare you for the remaining sections.

Target Audience

This document is intended for people who wish to design custom interfaces for Pocket Tunes. Regular users of Pocket Tunes do not need to read or understand the contents of this document.

About This Document

This document is a reference manual for people who wish to design skins for Pocket Tunes 2.1 and later.

About Pocket Tunes

Pocket Tunes is a portable audio player for handheld devices. It supports MP3, Ogg Vorbis, and other formats. Pocket Tunes has a skinnable interface, which allows for third-part skin designers to create their own custom interface.

For more details on Pocket Tunes, visit www.pocket-tunes.com

What is a Skin?

A skin is a collection of graphical controls that make up what you see on the screen of your handheld device. The skin contains a list of all the controls, what they look like, what you can do with them, and what actions they take when you activate them.

What is PocketSkin™ Technology?

PocketSkin™ Technology is available only with Pocket Tunes. It provides a very powerful and flexible skin interface. Each skin element can individually control all the major features of Pocket Tunes. The flexibility that PocketSkin™ technology provides is unequaled among Palm applications and allows for customization of the user interface that rivals many PC skin interfaces.

Overview

This section will give you an overview of how skins are created and used with Pocket Tunes.

Creating a Skin

The process of creating a skin is composed of five steps:

1. Deciding on the parameters.
2. Creating the artwork.
3. Writing the skin description file.
4. Running the skin compiler.
5. Installing the skin to your handheld device.

Each of these steps is covered in their own section below.

Deciding on the Parameters

Before you begin designing a skin, you must make several decisions.

First, you must decide what screen capabilities you will require for your skin. This includes the resolution, density, and color depth.

The resolution of the screen is the number of pixels that the device can display. Currently most PalmOS 5 devices support at least 320x320 resolution. (The Handspring Treo 600 only supports 160x160.) Some devices support extended resolutions, such as 320x480, and others also support landscape mode.

The density is related to the resolution. PalmOS can operate in one of several densities. The two that Pocket Tunes supports are: low and high. Generally speaking, 160x160 is low density, and 320x320 is high density.

Color depth is the number of bits allocated to each color component. Pocket Tunes supports either 8 or 16. All currently-available PalmOS 5 devices support either of these color depths, but some devices (such as the Treo 600) will not use all of the colors available, due to hardware restrictions. If you use 8-bit color, then you must provide a color palette. This is a binary file that contains entries of 24-bit RGB values. (You can export a color palette from many graphics programs; they are sometimes called ACT files.) Please consult a graphics book for details on how to set up and use a colormap. Note that you cannot have more than 256 entries in the colormap.

If you do not provide a colormap with an 8-bit skin, then Pocket Tunes will use the system colormap. Note that your colors may not map correctly if you do this, and your graphics may not appear correctly. Also, note that Pocket Tunes installs your skin at the end of the system colormap. So, for example, if you define only 16 colors in your colormap, Pocket Tunes will keep the first 240 colors of the system colormap, plus your 16 colors. Finally, note that some devices with a dynamic input area (virtual Graffiti) may not display the dynamic input area correctly if you use a palette that redefines the colors used in the dynamic input area.

Creating the Artwork

Pocket Tunes skins are created from a collection of image files. These image files contain the graphics that will be displayed for various controls on the screen, such as the background, individual buttons, slider controls, etc.

Pocket Tunes supports the industry standard Windows Bitmap format (BMP files). All major graphics software will create files in this format. When saving your BMP file, make sure that it is saved as 24-bit without compression. The skin compiler will convert this file to 16-bit or 8-bit, as appropriate, before including the graphic in the skin file. **Important Note:** When using 8-bit color on a high-resolution device, you should make sure that all of your BMP files have an even width (in pixels). This is due to a restriction in the blitter of most PalmOS 5 devices that causes a crash when blitting an odd-width bitmap to an even location on the screen (blitting to an odd location works okay). The skin compiler will emit a warning if you try to use an odd-width bitmap with an 8-bit high-density skin. The skin will work properly (Pocket Tunes works around the crash), but the bitmaps may not be aligned correctly when drawn to the screen.

If you will be using custom fonts in your skin, you should also create them at this time. There are three types of fonts you can use:

- **Built-in fonts** are fonts that are supported on the Palm already. These are the traditional fonts you are used to seeing. Smaller versions of these fonts are also available on high-density devices.
- **Bitmap fonts** are fonts that consist of a number of bitmaps. For each character in the font, you specify a BMP file that contains an image of that character. All the bitmaps must be the same height, but they can be of variable width.
- **Resource fonts** are fonts stored in a Palm-standard nfnt or NFNT resource. These fonts can be created with several third-party tools, such as palmfontconv (http://sourceforge.net/projects/palmfontconv/)

Writing the Skin Description File

The skin description file is an XML file that describes the parameters of the skin and gives a list of all the bitmaps, fonts, and controls.

Don't be scared of XML! XML is simply a standardized way to write text-based configuration files (it can be used for lots of other things, but that's what we use it for). You can edit them with notepad or your favorite text editor. Once you see the examples in this documentation, you won't have any trouble writing your own.

This configuration file specifies the locations of all the controls, what the controls look like, and what actions the controls take when you activate them.

The rest of this document contains detailed instructions on the format of this configuration file.

Note for previous skin authors: If you've written skins for Pocket Tunes 2.0 and earlier, you will notice one important difference about the XML file. Rather than having a fixed set of controls available that you must provide, you can now define an unlimited number of controls, and you are in charge of what each control does. If you want to have two play buttons, you can. If you want to omit all the buttons altogether, you can do that too!

Validating the XML file

Once you've written the XML description file, you should validate it to make sure that there are no errors. An easy way to do this is to open the XML file using Internet Explorer (or any other browser that understands XML). You can typically do this by double-clicking on the XML file. This will display a tree view of the XML file, and it will alert you if there are any structural errors.

Running the Skin Compiler

Once you have the bitmap files and description file, you can compile your skin. To do this, you will need PTSC (Pocket Tunes Skin Compiler), which is available from http://www.pocket-tunes.com/skins/ptsc.html

From a command prompt, change to the directory where your skin files are stored, and then type the following:

```
ptsc –v2 MySkin.xml MySkin.pdb
```

You may have to specify the pathname of ptsc to run this command. Note that there are several TXT files distributed with PTSC. These TXT files must be located in the same directory as the PTSC executable. (If PTSC is added to your PATH environment, then the TXT files must be copied to your current directory.)

This will create the PDB (Palm Database) file for your skin, which you can then install to your Palm. (See the next section.) If there are any errors, they will be printed out, and skin compilation will be halted.

Installing the Skin

Installing the skin is as simple as double-clicking the PDB file and performing a HotSync. Note that only one copy of a skin with the same name can be installed at a time. So make sure to change the name of your skin to something unique.

Skin Description File Reference

The following sections describe the various attributes that can appear in the skin description file.

General Format

Because the skin reference file is an XML file, it must begin with the following tag:

```
<?xml version="1.0"?>
```

This tells the skin compiler that this is a valid XML file.

XML tags are enclosed in angle brackets. Tags can contain a number of attributes, and then can also have text or other tags embedded inside of them. For example:

```
<tag attrib1=”value1” attrib2=”value2”>text</tag>

<AnotherTag><EmbeddedTag>some text</EmbeddedTag></AnotherTag>
```

For full details on XML formatting, please consult an XML reference guide. You can also view the sample skin description file that comes with PTSC for examples.

PocketTunesSkin Tag

Everything following the ?xml tag must be enclosed in the PocketTunesSkin tag.

```
<PocketTunesSkin>
      all of your Pocket Tunes configuration goes here
</PocketTunesSkin>
```

Comments

You can use the standard XML-style comments.

```
<!-- Here is my comment. -->
```

Case Sensitivity

In general, everything in the configuration file is case sensitive. In other words, you must use upper or lower case, as specified. If you use the wrong case, the configuration file will be invalid. There are certain exceptions to this, but you will be safest to always use the correct case.

Header Parameters

The following parameters can be included directly inside the PocketTunesSkin tag.

Version Tag

This specifies the version number of this skin.

Attributes:	None
Value:	2, 3, or 4
Example:	`<Version>4</Version>`

For greatest compatibility, you should use version 2 if your skin does not include navigation or album art controls.

For greatest compatibility, you should use version 3 if your skin includes navigation controls but does not include album art controls.

You must use version 4 if your skin includes album art controls.

Name Tag

This specifies the name of the skin. It must contain only 7-bit characters (no accented characters) and be less than 32 characters long.

Attributes:	None
Value:	Your skin name
Example:	`<Name>My New Skin</Name>`

ColorDepth Tag

This specifies the color depth to use for the skin. It must be 16 or 8.

Attributes:	None
Value:	8 or 16
Example:	`<ColorDepth>16</ColorDepth>`

Colormap Tag

This specifies the colormap file (ACT file) to use. This parameter is only needed for 8-bit skins.

Attributes:	None
Value:	Colormap filename
Example:	`<Colormap>colormap.act</Colormap>`

Density Tag

This specifies the density of the skin, either “High” or “Low”.

Attributes:	None
Value:	“High” or “Low”
Example:	`<Density>High</Density>`

Resolution Tag

This specifies the X and Y resolution for the skin. Currently, all PalmOS 5 devices will support 160x160 resolution. Many will also support 320x320. Some (such as the Garmin iQue 3600, Tungsten T3, Tapwave Zodiac, or the Sony CLIE UX-50) support higher resolutions. (On devices that support a dynamic input area – the hideable Graffiti area – Pocket Tunes will hide the graffiti area if necessary to display the skin. Also, on supported devices, Pocket Tunes will rotate the display if necessary to display wide-format skins.)

Attributes:	None
Value:	X and Y tags containing the X and Y resolution.
Example:	`<Resolution><X>320</X><Y>320</Y></Resolution>`

CustomFont1 Tag

This specifies the default font for the Custom1 font. When the skin is first loaded, the custom font will be set to this value. (They user can later change this value, and the user's preference will be remembered until a new skin is loaded.)

Attributes:	None
Value:	The name or ID of a font. (See the Fonts section.)
Example:	`<CustomFont1>FontNormal</CustomFont1>`

CustomFont2 Tag

This specifies the default font for the Custom2 font. When the skin is first loaded, the custom font will be set to this value. (They user can later change this value, and the user's preference will be remembered until a new skin is loaded.)

Attributes:	None
Value:	The name or ID of a font. (See the Fonts section.)
Example:	`<CustomFont2>5</CustomFont2>`

Bitmaps

Your list of bitmaps must be enclosed in the Bitmaps tag.

Attributes:	None
Value:	Bitmap tags
Example:	`<Bitmaps>`

```
<Bitmaps>
      your Bitmap tags go here
</Bitmaps>
```

Bitmap Tag

Each bitmap is represented by its own Bitmap tag. The tag contains the filename where the image data is stored, and it can also optionally contain transparency information.

Attributes: **ID:** a number to identify this bitmap. Numbering starts at 0, and there must not be any “missing” IDs.

Value: **File** tag: the filename where the BMP file is stored.
Transparent tag (optional): contains R, G, B, and Index tags that specify the transparent color.

Example 1:

```
<Bitmap ID=”0”><File>background.bmp</File></Bitmap>
```

Example 2:

```
<Bitmap ID=”1”><File>digit0.bmp</File>
      <Transparent><R>255</R><G>255</G><B>0</B>
      </Transparent>
</Bitmap>
```

Fonts

Your list of fonts must be enclosed in the Fonts tag.

Attributes:	None
Value:	Font tags
Example:	`<Fonts>`

```
<Fonts>
     your Font tags go here
</Fonts>
```

Notes:

Each font that you define can be marked as Custom1 or Custom2. When a font is marked as either of these, it will be added to the Custom1 or Custom2 font list. These lists are displayed to the user in a new preferences screen in Pocket Tunes. The user can pick which custom font is used for Custom1 and Custom2. By default, each list contains all of the built-in Palm fonts. Any additional fonts that you define as Custom1 or Custom2 will also appear in this list when your skin is selected.

When referring to a font from a text area, you can specify the ID of one of these fonts. Or you can specify one of the built-in font types:

- FontNormal
- FontBold
- FontLarge
- FontLargeBold
- FontSmallNormal
- FontSmallBold
- FontSmallLarge
- FontSmallLargeBold
- FontCustom1
- FontCustom2

Font Tag

Each font is represented by its own Font tag. The tag contains the list of bitmaps (bitmap fonts) or the nfnt/NFNT resource filename.

Attributes: **ID:** a number to identify this font. Numbering starts at 0, and there must not be any "missing" IDs.
Type: Either "Bitmap" or "Resource", depending on which type of font you are creating.

Value: **Character** tag: (Bitmap fonts only) The bitmap for a specific character. (See the "Character Tag" section below.)
File tag: (Resource fonts only) The filename of the font data. This is binary contents of a Palm font resource (NFNT or nfnt). It cannot be an actual resource file. For example, this is the .bin file output by the palmfontconv software. (See the "Notes" section below for more details.)
Custom1 tag: Adds the font to the Custom1 list.
Custom2 tag: Adds the font to the Custom2 list.

Example 1:

```
<Font Type="Bitmap" ID="0">
      <Character Char="0x30">0</Character>
      <Character Char="1">1</Character>
      <Character Char="2">2</Character>
      <Character Char="3">3</Character>
      <Character Char="4">4</Character>
      <Character Char="5">5</Character>
      <Character Char="6">6</Character>
      <Character Char="7">7</Character>
      <Character Char="8">8</Character>
      <Character Char="9">9</Character>
</Font>
```

Example 2:

```
<Font Type=”Resource” ID=”1”>
      <File>myfont.bin</File>
      <Custom1/>
      <Custom2/>
</Font>
```

Notes:

The resource font file is the binary data portion of an nfnt or NFNT font resource. These font resources can be produced with software such as xFONT and palmfontconv. For example, to use a TrueType font with Pocket Tunes, you can create the .bin file with following command (using palmfontconv, which is available at http://sourceforge.net/projects/palmfontconv/ :

```
topalmtext font.ttf 20 | fontconv - H font.bin
```

"font.ttf" is the filename of the TrueType font. (You can find TrueType fonts in your C:\Windows\Fonts or C:\Winnt\Fonts folder on Windows computers, or you can download them from various web sites. Please read the copyright notices for the font before distributing it; some of them require payment in order to distribute them.)

"20" is the point size of the font. To determine the pixel height of the font (to determine how much space you need in your skin), run

```
topalmtext font.ttf 20 > font.txt
```

Then open font.txt in a text editor and look at the ‘fRectHeight’ line to see how many pixels tall the font is.

Character Tag

Bitmap fonts require the use of the Character tag, which is defined here. This is one character tag for each character that you wish to define a bitmap for. Any characters that do not have a bitmap assigned will use the bitmap for the lowest ASCII value assigned here.

Attributes:	**Char:** the actual character to define, or a hex code for the ASCII value of the character (e.g. “0x30” for the digit ‘0’).
Value:	The bitmap ID for the specified character
Example 1:	`<Character Char=”a”>11</Character>` (Bitmap 11 contains the visual appearance of the character ‘a’.)
Example 2:	`<Character Char=”0x30”>12</Character>` (Bitmap 12 contains the visual appearance of the character ‘0’, which has the ASCII value of 0x30.)

Navigation

As of Pocket Tunes 3.1, skins can include 5-way navigation behavior. This section describes how to specify navigation behavior.

Your list of navigation items must be enclosed in the Navigation tag.

Attributes: **Visible:** A condition that determines when the navigation focus ring is visible. This value is required to be "!ConditionPlaying"

Contents: A set of Nav tags

Example:

```
<Navigation Visible="!ConditionPlaying">
     your Nav tags go here
</Navigation>
```

Notes:

Nav

Each navigable item on the skin must be described by a Nav tag enclosed within the Navigation parent tag.

Attributes: **ID:** The numeric ID of the control. Within the parent Navigation tag, these numbers should start at 0 and increment by one for every subsequent Nav tag.
Visible: A condition that determines whether this focus ring item is currently visible. When this condition is false, this item will be ignored when the user navigates to it.

Contents: Bounds, Shape, Action, ButtonAction, Playlist, Above, and/or Below tags

Example:

```
<Nav ID="0" Visible="CA">
     <Bounds>
          <X>20</X><Y>139</Y>
          <Width>252</Width><Height>80</Height>
     </Bounds>
     <Shape>Rectangle</Shape>
     <Playlist>19</Playlist>
     <Below>3</Below>
     <Action>SetCondition CZ true</Action>
</Nav>
```

Notes:

The "Bounds" tag specifies the bounding rectangle for this item's focus ring.

The "Shape" tag specifies the shape of the focus ring for this item, and may be one of the following values: Rectangle, RoundedRectangle, or Circle.

The "Action" tag specifies an action that occurs when the user presses the center button on the 5-way while the focus is on this item. See the "Action Reference" in this document for more details. There can be multiple actions associated with each Nav item.

The "ButtonAction" tag optionally specifies the ID of a button to associate with the Nav item. When the user presses the 5-way while focused on that Nav item, it will act as if the user had tapped his stylus on the specified button. There can only be one ButtonAction tag.

The “Playlist” tag specifies the ID of a playlist control that this Nav item refers to. When Pocket Tunes encounters a navigable playlist control, it allows the user to navigate through the list of songs displayed in the playlist.

The “Above” tag specifies which navigation element (by ID) to switch to if the user presses the up 5-way button.

The “Below” tag specifies which navigation element (by ID) to switch to if the user presses the down 5-way button.

Navigation algorithm: Whenever the visible condition on the main Navigation control is true, a focus ring will be displayed. By default, ID 0 is focused. We recommend setting the focus to the appropriate item by using a Computation control when the skin is first loaded.

When the user presses the left or right 5-way buttons, Pocket Tunes will scan through the list of nav controls (backwards or forwards, respectively) until it finds a nav control whose Visible condition is true. It will then switch the focus to that item. If it does not find anything, it will not change the focus. The order of the controls is based on the ID that you assign.

When the user presses the up or down 5-way buttons, Pocket Tunes will consult the Above and Below tags for the current nav control. It will then continue following the Above and Below tags until it finds a nav control that is visible. When it finds one, it will change the focus. If it doesn’t find anything, the focus will not change.

Controls

Your list of controls must be enclosed in the Controls tag.

Attributes:	None
Value:	Control tags
Example:	`<Controls>` `your Control tags go here` `</Controls>`

```
<Controls>
     your Control tags go here
</Controls>
```

Notes:

There are six types of controls that can be included in your skin. Each control is detailed in its own section below.

The ordering of the controls is important. When Pocket Tunes needs to refresh the screen, it asks each control to refresh itself, in turn, in the order defined by its ID. Pocket Tunes also uses this order to determine which control handles pen taps and other events. For example, if you have a control placed on top of another control, the control with the lowest ID will get drawn first and will also have first dibs on any pen taps, etc.

Runtime Behavior

Here is a quick description of how Pocket Tunes interacts with the controls that you define in your skin while Pocket Tunes is running.

When Pocket Tunes loads a skin, it reads each control from the skin PDB file and initializes its internal data structures to get ready to operate with the controls. Once this is done, Pocket Tunes tells the skin to refresh itself.

When Pocket Tunes refreshes the skin, it goes through each control, in order, and tells that control to refresh. If the control has been modified (for example, its visible condition has changed, or the value it's displaying has changed), then the control will be redrawn. If nothing has changed, then the control is skipped.

Pocket Tunes continues to refresh every control 4 times a second. If the refresh operation causes an action to be executed, then Pocket Tunes will refresh the controls a second time to make sure that everyone is up to date. Pocket Tunes will not refresh more than twice in this situation.

When the user taps on the screen, moves the stylus, or releases the stylus, Pocket Tunes again asks each control if it is able to handle that user input. Pocket Tunes asks each control to handle the input, by order of their IDs. If a control can handle the user input, then none of the other controls will be asked about it. If an action is executed while handling the user input, then Pocket Tunes will refresh to skin once to make sure all of the controls have a chance to respond to that action.

Control Attributes

Each control has a number of items associated with it. These contain the information about where and when the control appears on the screen, what it does when you activate it, and what type of content it displays.

These items often contain information about conditions, integers, strings, and events. This information is represented in a uniform way, which is described below.

Visible Attribute

Each control contains a Visible attribute, which determines when the control is visible. If the Visible condition is true, then the control is visible. When a control is visible, it can be displayed, and it can also respond to user events.

Example:

```
<Control Type=”Button” ID=”2”
Visible=”ConditionSongSelected”>
     <!-- insert control data here -->
</Control>
```

In the above example, the button control is only visible when there is a song selected. When a song is not selected, the button is not drawn and will not respond to any user inputs. (See the “Control Conditions” section for details on how to define a condition.)

Control Integers

An integer is a numeric value that represents a number 0 or greater. (It cannot represent negative numbers. The maximum integer is approximately 4 billion – it is a 32-bit integer.) Integers are used to represent values that can have many values, such as the current song position.

Integer Constants

Pocket Tunes allows you to specify any integer value for an integer. For example, these are valid integer constants.

```
“0”
“5432”
“42”
```

Integer Variables

Pocket Tunes defines a large number of built-in integer variables, all of which are detailed in the Integer Variable Reference section below. When expressing an integer, you can use any of these built-in integer variables.

There are 26 user-defineable integer variables, named “IA” through “IZ”. Each of these variables can be defined to any value by you and can be used for any purpose. When the skin is loaded, they are all initialized to 0.

For example, the following are valid integers that you can specify.

```
“IG”
“IntegerVolume”
```

Integer Expressions

You can also combine integer variables together into expressions. You can create results that are the result of the addition (+), subtraction (-), multiplication (*), division (/), or modulus (%) of two integers. For example,

```
“IntegerVolume+IntegerVolumeIncrement”
“IntegerSongPosition+5000”
“IntegerBalance-IntegerBalanceIncrement”
“IntegerVolume/2”
“IntegerVolume*2”
“IntegerRandom%10”
```

Note that you cannot combine more than two integers at a time with these operators. See the sidebar below (in the Control Conditions section) for details on how to construct more complicated expressions.

Control Conditions

A condition is an expression that determines whether something is True of False. Conditions are used for a number of things, such as determining if a control is visible, determining when an event should execute, etc.

Condition Variables

Pocket Tunes defines a large number of built-in condition variables, all of which are detailed in the Condition Variable Reference section below. When expressing a condition, you can use any of these built-in condition variables.

There are several special condition variables available. The first two are “False” and “True”. These always evaluate to False and True, respectively.

There are also 26 user-definable condition variables, named “CA” through “CZ”. Each of these variables can be defined to any value by you, and can be used for any purpose. When the skin is loaded, they are all initialized to False.

For example, the following are valid conditions that you can specify.

```
“True”
“ConditionSongSelected”
“ConditionPlaying”
```

Condition Expressions

You can also combine condition variables together into expressions. You can create a result that is the logical AND of two conditions, or you can create a result that is the logical OR of two conditions. For AND, use the “+” symbol. For OR, use the “|” symbol.

For example, the following are valid condition expressions:

```
“ConditionPlaying | ConditionPaused”
“ConditionPlaying + ConditionFiveWayCenterPressed”
```

Advanced Sidebar

Note that you **cannot** create more complicated expressions by using parentheses or multiple operators in a condition. If you need to do this, you can use the user-definable condition variables. For example, if you wanted to compute "ConditionPlaying | ConditionPaused | ConditionStopped", you could create the following Computation control. (See below to find out the details of the Computation control.)

```
<Control Type="Computation" ID="0" Visible="true">
      <Event Type="ChangedCondition"
            Condition="ConditionPlaying | ConditionPaused">
            SetCondition CA ConditionPlaying | ConditionPaused
      </Event>
      <Event Type="ChangedCondition" Condition="CA">
            SetCondition CB CA | ConditionStopped
      </Event>
</Control>
```

With this control active, CB will always contain the result of the expression "ConditionPlaying | ConditionPaused | ConditionStopped". (Of course, this particular example is useless, but it gives you an idea of how to do this sort of thing.)

You can also create expressions with integers using the less than "&lt;" (this is not a typo; see below for details), greater than ">", equal to "=", or not equal to "!=" operators. For example, the following are valid conditions:

```
"IntegerSongPosition != 0"
"IntegerBatteryLevel > 25"
"IntegerScreenWidth &lt; IntegerScreenHeight"
```

Note that the less than operator is represented by "&lt;". This is the XML way of representing the "<" character.

Condition Inversion

Finally, you can invert the result of an expression by playing a "!" character in front of it. Note that this inverts the entire expression and not just the first operand. For example,

```
"!ConditionPlaying"
```

is true whenever a song is NOT playing.

```
"! ConditionPlaying | ConditionPaused"
```

is true whenever we're not playing or paused. This is a bit confusing without the parentheses; it is evaluated as if it were written like this: "! (ConditionPlaying | ConditionPaused)"

Condition Inversion, Part Two

As of Pocket Tunes 2.2, you can also invert the left-hand-side or right-hand-side of a conditional expression by preceding it with the ~ character.

For example,

```
"CA | ~ CB"
```

is true when CA is true **or** when CB is false.

You can use the ~ and ! operators in the same expression. Remember that the ! operator applies to the **entire** expression and not any of the parameters!

Compatibility Note: If you create a skin using the ~ character, it will load in Pocket Tunes 2.1, but it will not function correctly. Please make sure to let your audience know that this skin only works properly with Pocket Tunes 2.2 or later!

Control Strings

A string is an alphanumeric (letters and numbers) value that can be displayed to the user. These include things such as the title of the song, the current time, etc.

String Variables

Pocket Tunes defines a number of built-in string variables, which are detailed in the String Variable Reference below.

Here are some examples of valid string variables.

```
“StringTimeAMPM”
“StringSongTitle”
```

Control Events

An event describes something that can occur while Pocket Tunes is running. It consists of an indication for when the event occurs as well as an action to take when it does occur.

Attributes:	**Type:** The type of event. This can contain different values, depending on which control the event is associated with. **Condition:** A condition expression indicating when this event will be triggered.
Value:	The action to be executed. See the Control Actions section.
Example 1:	`<Event Type=”Clicked”>Play</Event>`
Example 2:	see below

```
<Event Type=”Clicked” Condition=”ConditionScanning”>
     DoneScanning
</Event>
```

Control Actions

You can define an action to be executed whenever an event is triggered. (See the previous section for details on events.)

Pocket Tunes defines a large number of built-in actions, which are detailed in the Action Reference below. Some actions take parameters, such as conditions, variable names, etc. Those parameters should be separated by spaces from the action name.

Computation Control

The computation control has no visible component. It is meant as a means of performing automatic computation of complex expressions. There may be 0 or more of these controls.

Attributes: **Type:** “Computation”
ID: The numeric ID of the control.
Visible: A condition that determines whether or not the control is visible. Irrelevant for this type of control.

Contents: A set of events that are executed during the lifetime of the skin. See below for the different types of events recognized.

Example:

```
<Control Type=”Computation” ID=”0” Visible=”true”>
   <Event Type=”ChangedCondition”
      Condition=”ConditionPlaying | ConditionPaused”>
      SetCondition CA ConditionPlaying | ConditionPaused
   </Event>
   <Event Type=”ChangedCondition” Condition=”CA”>
      SetCondition CB CA | ConditionStopped
   </Event>
</Control>
```

<u>Events</u>

This control recognizes the following event types:

Condition: Whenever the event’s condition is true, the event will be triggered. This is evaluated every time the controls are refreshed, which occurs once or twice every four seconds, and also whenever a user input event causes an action to be executed.

ChangedCondition: Whenever the event’s condition changes value, this event is triggered. Note that this is checked only when the controls are refreshed, so it may not trigger the event immediately. This will be triggered once immediately when the skin is loaded.

ChangedInt: This event takes an additional parameter named “Int”, whose value is an integer expression. Whenever the value of this expression changes, this event is triggered. Note that this is checked on ly when the controls are refreshed, so it may not trigger the event immediately. This will be triggered once immediately when the skin is loaded.

Background Control

The background control displays the background bitmap for the skin. This control is required; if it does not exist, the background may not display correctly. There may not be more than one of these controls. This control must appear before any other controls (except for the computation controls) or else it will not draw correctly.

Attributes: **Type:** “Background”
ID: The numeric ID of the control.
Visible: A condition that determines whether or not the control is visible. If the background is not visible, it will be drawn as white.

Contents: **Bitmap tag:** This tag takes an optional “Condition” parameter, which determines if the specified bitmap is drawn. The tag’s value is the bitmap ID to be drawn. This bitmap must be the full size of the screen or else the remaining area will not draw properly. There may be 0 or more of these tags. The first tag whose condition is met is the one that will be drawn. If none of them meet their conditions, then the background may not draw properly.

Example 1:

```
<Control Type=”Background” ID=”0” Visible=”true”>
      <Bitmap>2</Bitmap>
</Control>
```

Example 2:

```
<Control Type=”Background” ID=”0” Visible=”true”>
      <Bitmap Condition=”CF”>3</Bitmap>
      <Bitmap Condition=”!CF”>4</Bitmap>
</Control>
```

<u>Events</u>

Background Controls do not contain any events.

Button Control

The button control is designed primarily to be used as an area of the screen that is clickable. However, it can also be used as a status indicator or for other purposes.

Attributes: **Type:** “Button”
ID: The numeric ID of the control.
Visible: A condition that determines whether or not the control is visible. If a button is not visible, it will not be drawn, and it will not respond to any user input.

Contents: **Appearance tag:** This tag determines the appearance of the button at various points in time. Its value is a bitmap ID to be displayed when the conditions are met. It accepts two parameters: When and Condition. The When parameter is either “Clicked” or “Unclicked”. This controls the appearance of the button when it is in the clicked or unclicked state. (See below for a discussion of how buttons get clicked.) The Condition parameter is an optional condition expression (the default is “true”) that indicates when this particular bitmap should be drawn. The first appearance whose Condition evaluates to true is drawn.
Value: This tag determines the Conditional value associated with the control. When this value is true, the button will appear in the “Clicked” state; when false, it will appear in the “Unclicked” state. Note that, by default, clicking on the button does not affect the value associated with it. If you wish to create a toggle button, for example, you shold create an action associated with the Clicked event. (See below.)
Bounds: This tag contains 4 tags that specify the location of the control: X, Y, Width, and Height.
Event: One of four supported event types: Clicked, Pressed, Released, and Held.

Example 1:

```
<Control Type=”Button” ID=”10” Visible=”true”>
        <Bounds>
                <X>106</X><Y>236</Y><Width>60</Width><Height>58</Height>
        </Bounds>
        <Value>ConditionFiveWayLeftPressed</Value>
        <Appearance When="Clicked" Condition="true">32</Appearance>
        <Event Type="Held" msStart="500" msRepeat="250">
                ScanBackward
        </Event>
        <Event Type="Clicked" Condition="!ConditionScanning">
                PreviousTrack
        </Event>
        <Event Type="Clicked" Condition="ConditionScanning">
                DoneScanning
        </Event>
</Control>
```

Example 2:

```
<Control Type="Button" ID="2" Visible="true">
 <Bounds>
    <X>17</X><Y>9</Y><Width>24</Width><Height>24</Height>
 </Bounds>
 <Appearance When="Unclicked" Condition="ConditionPlaying">22</Appearance>
 <Appearance When="Unclicked" Condition="ConditionPaused">23</Appearance>
 <Appearance When="Unclicked" Condition="ConditionStopped">24</Appearance>
 <Appearance When="Clicked" Condition="ConditionPlaying">22</Appearance>
 <Appearance When="Clicked" Condition="ConditionPaused">23</Appearance>
 <Appearance When="Clicked" Condition="ConditionStopped">24</Appearance>
</Control>
```

Buttons are considered “clicked” when the user holds the pen down on top of it, or when the value associated with the button is true. (Note that if the user drags his stylus away from the button, it can momentarily become “unclicked” until the user drags his stylus back over the button or releases the stylus altogether.)

Events

Button controls may respond to any of the following events:

Clicked: This event occurs when the button is tapped and then released.

Pressed: This event occurs when the button is first depressed, caused by the user tapping on it. Note that this event is not executed when the user drags off of the button and then back onto it.

Released: This event occurs when the user lifts his stylus from the screen after pressing the button. Note that this event is always executed, regardless of whether the user moved the stylus away from the button.

Held: This event occurs when the button is held down for a certain amount of time. The event takes two required parameters: msStart and msRepeat. msStart specifies the number of milliseconds that the button is held down after which the Held event is executed. (There are 1000 milliseconds in a second.) msRepeat specifies the number of milliseconds thereafter when the Held event is executed (use 0 if you don’t want it to repeat more than once). (See the example above.) Note that you should only rely on this being evaluated 4 times per second, so you should only rely on 250ms granularity for this action.

Slider Control

The slider control is designed to be used to represent a continuous value within a certain range. The value is represented by the position of the slider, and if the user slides the slider, the value will be updated.

Attributes: **Type:** “Slider”
ID: The numeric ID of the control.
Visible: A condition that determines whether or not the control is visible. If a slider is not visible, it will not be drawn, and it will not respond to any user input.

Contents: **Bitmap tag:** This tag contains the ID of the bitmap for the “bar” portion of the slider control.
Bounds tag: This tag contains 4 tags that specify the position and size of the area in which the slider can slide: X, Y, Width, and Height.
Direction tag: Either “Horizontal” or “Vertical”, depending on which direction the slider will slide.
Increment tag: The number of increments, or steps that the slider has. This must be an odd number greater than 2. It represents the number of “stopping points” where the slider will stick as it is sliding. Specify 0 for a continuous slider.
DirectAccess tag: (new in 4.0.0) Specify this flag to allow tapping directly on the desired destination location, without first requiring that the user grab the slider bar.
Value tag: Contains four other tags that indicate the value, the minimum value, the maximum value, and the step value.

- **Var tag:** The name of the integer variable that contains the value of this slider. This cannot be an integer expression.
- **Min tag:** An integer expression representing the minimum value that this control can represent.
- **Max tag:** An integer expression representing the maximum value that this control can represent.
- **Step tag:** An integer expression representing the step value for this control. This is the value that the control is increased or decreased when it is changed by some mechanism other than sliding.

Event tag: One of three supported event types: StartSliding, Sliding, and StopSliding.

Example:

```
<Control Type="Slider" ID="4" Visible="ConditionSongSelected">
 <Bounds><X>15</X><Y>153</Y><Width>289</Width><Height>11</Height></Bounds>
 <Bitmap>27</Bitmap>
 <Direction>Horizontal</Direction>
 <Increment>0</Increment> <!-- continuous -->
 <Value>
  <Var>IntegerSongPosition</Var>
  <Min>0</Min>
  <Max>IntegerSongLength</Max>
  <Step>1000</Step>
 </Value>
</Control>
```

<u>Events</u>

StartSliding: This event occurs when the user first taps on the slider bar and before it has been moved.

Sliding: This event occurs whenever the user slides the slider to a new position.

StopSliding: This event occurs whenever the user releases the stylus to leave the control at its new position.

Text Area Control

The text area control displays any string or integer value to the user. There are a large number of options for displaying the text using different methods.

Attributes: **Type:** “TextArea”
ID: The numeric ID of the control.
Visible: A condition that determines whether or not the control is visible. If a text area is not visible, it will not be drawn, and it will not respond to any user input.

Contents: **Scrolling tag:** One of three values: Horizontal, Vertical, or None. When text is set to scroll horizontally, it will be moved to the left by several pixels 4 times per second. Horizontal scrolling can only apply to a single line of text; if you pick a value with multiple lines, it may not draw properly. In contrast, vertical scrolling works best with values that have multiple lines. It will scroll the text area up until one full line has been scrolled. It will then pause briefly before continuing to scroll. Note that if the full text appears in the window, no scrolling with take place. This applies to both horizontal and vertical scrolling. Also note that you cannot set scrolling to Horizontal or Vertical if you use an alignment other than “Left” (see the Alignment tag below).
Font tag: This tag contains the name or the ID of the font to use for displaying the text. Any font can be used. (See the Fonts section for a list of the available font names.)
Alignment tag: This tag has one of three values: Left, Right, or Center. When set to Left, the text will be left-justified to the left side of the text area. When set to Right, it will be aligned with the right side. When set to Center, it will be centered in the text area. Note that you cannot enable scrolling if you use Right or Center alignment.
BgColor tag: The background color of the font. This parameter is ignored when using a bitmap font or if the Transprent flag has been set (see below for details on the Transparent flag.).
FgColor tag: The foreground color of the font. This parameter is ignored when using a bitmap font. (When displaying the playlist in a text area, the foreground and background colors will be reversed for the currently-selected song.) **Note:** If you create a non-scrolling text area that uses a non-transparent bitmap font, then you can eliminate the flicker of updating by setting the BgColor and FgColor to the same value.
MinDigits tag: If you are displaying a number in the control, you can specify the minimum number of digits to be displayed. For example, if you specify 2, all values will be displayed with at least two digits: 00, 05, 10, etc.
Value tag: Contains the integer or string variable (no expressions are allowed) to be displayed in the text area.
Event tag: A clicked event can be associated with a text area control.
Bounds tag: This tag contains 4 tags that specify the position and size of the text area: X, Y, Width, and Height.
Transparent tag: If this tag exists, then the text in the text area is drawn with a transparent background. In other words, the background “shows through” behind the font. This can be optionally specified for a built-in or

resource font, but you must specify this whenever using a bitmap font that has transparent characters. **Note:** Using a transparent font for a large scrolling text area can use up a large amount of dynamic memory on the device. It can also slow down scrolling. If your background is a solid color, don't use this tag.

Example 1:

```
<Control Type="TextArea" ID="16" Visible="ConditionSongSelected">
        <Bounds>
                <X>237</X><Y>168</Y><Width>28</Width><Height>19</Height>
        </Bounds>
        <BgColor><R>0</R><G>0</G><B>0</B></BgColor>
        <!-- If the fg color equals the bg color, then the background is
        treated as non-uniform -->
        <FgColor><R>0</R><G>0</G><B>0</B></FgColor>
        <Scrolling>None</Scrolling>
        <Font>0</Font>
        <Alignment>Right</Alignment>
        <Value>IntegerSongPositionMinutes</Value>
        <Event Type="Clicked">SetCondition ConditionCountDown
                !ConditionCountDown</Event>
</Control>
```

Example 2:

```
<Control Type="TextArea" ID="23" Visible="true">
        <Bounds>
                <X>20</X><Y>63</Y><Width>279</Width><Height>90</Height>
        </Bounds>
        <BgColor><R>200</R><G>200</G><B>255</B></BgColor>
        <FgColor><R>255</R><G>255</G><B>255</B></FgColor>
        <Transparent/>
        <Scrolling>Vertical</Scrolling>
        <Font>FontCustom2</Font>
        <Alignment>Left</Alignment>
        <Value>StringSongComments</Value>
</Control>
```

Events

Clicked: This event is triggered when the user taps on the text area.

Progress Bar Control

The progress bar control is designed to represent continuous values that cannot be changed by the user. The value is represented by drawing a portion of the control's bitmap. For example, if you have a progress bar to represent the current battery life, and the battery life is 25%, then the left 25% of the battery bitmap will be displayed (horizontal) or the bottom 25% of the battery bitmap will be displayed (vertical).

Attributes: **Type:** "ProgressBar"
ID: The numeric ID of the control.
Visible: A condition that determines whether or not the control is visible. If a progress bar is not visible, it will not be drawn, and it will not respond to any user input.

Contents: **Bitmap:** The ID of the bitmap to be used for displaying this control.
Event: A clicked event.
Value: An integer expression representing the value of this control.
MaxValue: An integer expression representing the maximum value that the control's value can have. (The minimum value is assumed to be 0.) If the actual value ever goes above this MaxValue, the control will be displayed completely.
Bounds: This tag contains 4 tags that represent the position and size of the bitmap: X, Y, Width, and Height.
Direction: This tag contains either "Horizontal" or "Vertical", depending on the desired orientation of the progress bar.

Example:

```
<Control Type="ProgressBar" ID="24" Visible="true">
      <Bitmap>36</Bitmap>
      <Value>IntegerBatteryLevel</Value>
      <MaxValue>IntegerMaxBatteryLevel</MaxValue>
      <Direction>Horizontal</Direction>
      <Bounds>
            <X>80</X><Y>300</Y>
            <Width>30</Width><Height>10</Height>
      </Bounds>
</Control>
```

<u>**Events**</u>

Clicked: This event is triggered when the user taps on the progress bar.

Album Art Control

The album art control is used to display a graphical image associated with the currently playing song. The image will be resized (with its aspect ratio preserved) to fit into the control. (It is strongly recommended that skin designers use a square region for album art controls.)

Attributes: **Type:** “AlbumArt”
ID: The numeric ID of the control.
Visible: A condition that determines whether or not the control is visible. If an album art control is not visible, it will not be drawn, and it will not respond to any user input.

Contents: **Default:** (optional) The ID of a bitmap to be displayed in this control if the control is visible and the current song has no associated graphical image.
Event: (optional) A clicked event.
Bounds: This tag contains 4 tags that represent the position and size of the bitmap: X, Y, Width, and Height.

Example:

```
<Control Type="AlbumArt" ID="24" Visible="true">
      <Default>36</Default>
      <Bounds>
            <X>240</X><Y>80</Y>
            <Width>60</Width><Height>60</Height>
      </Bounds>
      <Event Type="Clicked"> ShowAlbumArt </Event>
</Control>
```

Example:

```
<Control Type="AlbumArt" ID="24"
            Visible="ConditionSongHasAlbumArt">
      <Bounds>
            <X>240</X><Y>80</Y>
            <Width>60</Width><Height>60</Height>
      </Bounds>
</Control>
```

Events

Clicked: This event is triggered when the user taps on the album art control.

Condition Variable Reference

This section contains a list of all the condition variables and what they represent.

False [read-only]

This built-in value always evalutes to false.

True [read-only]

This built-in value always evaluates to true.

CA – CZ [read-write]

These 26 values are for use by the skin for storing any transient conditions. When the skin is loaded (or when Pocket Tunes starts), these values are all initialized to false.

CPA – CPZ [read-write]

These 26 values are for use by the skin for storing any persistent conditions. Unlike CA – CZ, these values are remembered even if the user quits Pocket Tunes. However, if the user picks a new skin, these values are destroyed. When the skin is first loaded, these are initialized to false.

ConditionSongSelected [read-only]

This value is true whenever the user has selected a song, whether or not it’s playing.

ConditionPlaying [read-only]

This value is true whenever a song is being played.

ConditionPaused [read-only]

This value is true whenever a song is paused.

ConditionStopped [read-only]

This value is true whenever a song is stopped.

ConditionStereo [read-only]

This value is true if the current song is in stereo.

ConditionMono [read-only]

This value is true if the current song is in mono.

ConditionRepeat [read-write]

This value is true if the repeat setting is turned on. When true, Pocket Tunes will restart the playlist after it has played all the songs.

ConditionShuffle [read-write]

This value is true if the shuffle setting is turned on. When true, Pocket Tunes will play the songs in the playlist in a random order.

ConditionScanning [read-only]

This value is true if the user is currently scanning forward or backward in the audio. This value is set in response to the ScanForeward or ScanBackward action.

ConditionPrefTurnOffScreen [read-write]

This value is true if the “Turn off screen while playing” preference is selected.

ConditionPrefTurnOffScreenUnlessInCradle [read-write]

This value is true if the “unless the Palm is in its cradle” option is selected.

ConditionPrefShowSongListInCommentArea [read-write]

This value is true if the “Show song list in comment area” option is selected.

ConditionPrefEnableBackground [read-write]

This value is true if the “Enable background play” option is selected.

ConditionPrefBackgroundPenSwipe [read-write]

This value is true if the “Bring up console with pen swipe” option is selected.

ConditionPrefBackgroundButton [read-write]

This value is true if the “Bring up console with button” option is selected.

ConditionPrefEnableCrossfade [read-write]

This value is true if the “Enable crossfade” option is selected. This option is ignored on Pocket Tunes Basic.

ConditionPrefAskForBookmarkName [read-write]

This value is true if the “Always ask for bookmark name” option is selected.

ConditionPrefAutoBookmark [read-write]

This value is true if the “Enable Auto-Bookmarking” option is selected. This option is ignored on Pocket Tunes Basic.

ConditionPrefFlashLEDScreenOff [read-write]

This value is true if the “Flash LED while screen is off” option is selected.

ConditionPrefBlankScreenInOtherApps [read-write]

This value is true if the “Blank screen while in other apps” option is selected.

ConditionPrefIgnoreScreenTapsWhenBlanked [read-write]

This value is true if the “Ignore screen taps when blanked” option is selected.

ConditionPrefStartPlayingOnStartup [read-write]

This value is true if the “Start playing on startup” option is selected.

ConditionPrefStopPlayingOnLowBattery [read-write]

This value is true if the “Stop Playing on low battery” option is selected.

ConditionPrefStopPlayingAfterXMinutes [read-write]

This value is true if the “Stop playing after X minutes” option is selected.

ConditionPrefWarnAboutAudioQuality [read-write]

This value is true if the “Warn me about audio quality” option is selected.

ConditionPrefEqualizer [read-write]

This value is true if the “Equalizer” option is selected.

ConditionCountDown [read-write]

This value is true if the current song position should be displayed as a count-down to the end rather than a count-up from the beginning.

ConditionFiveWayLeftPressed [read-only]

This condition is true whenever the left button on the 5-way navigator is pressed down. Obviously this only works on devices that have a 5-way navigator. When the 5-way axes are swapped, this will be true with the swapped button is pressed.

ConditionFiveWayRightPressed [read-only]

This condition is true whenever the right button on the 5-way navigator is pressed down. Obviously this only works on devices that have a 5-way navigator. When the 5-way axes are swapped, this will be true with the swapped button is pressed.

ConditionFiveWayCenterPressed [read-only]

This condition is true whenever the center button on the 5-way navigator is pressed down. Obviously this only works on devices that have a 5-way navigator.

ConditionFiveWayUpPressed [read-only]

This condition is true whenever the up button on the 5-way navigator is pressed down. Obviously this only works on devices that have a 5-way navigator. When the 5-way axes are swapped, this will be true with the swapped button is pressed.

ConditionFiveWayDownPressed [read-only]

This condition is true whenever the down button on the 5-way navigator is pressed down. Obviously this only works on devices that have a 5-way navigator. When the 5-way axes are swapped, this will be true with the swapped button is pressed.

ConditionRegistered [read-only]

This condition is true if the user has a valid registration code.

ConditionExpired [read-only]

This condition is true if the user’s trial period has expired and they have not yet entered a registration code.

ConditionRegisteredBasic [read-only]

This condition is true if the user has a valid Basic registration code.

ConditionRegisteredDeluxe [read-only]

This condition is true if the user has a valid Deluxe registration code.

ConditionBatteryCharging [read-only]

This condition is true if the battery is currently charging.

ConditionSeekable [read-only]

This condition is true when the current audio file is seekable (i.e. it's not a streaming file, like ShoutCast).

ConditionEqualizerPossible [read-only]

New in Pocket Tunes 2.2
This condition is true whenever it is possible to use the equalizer. In Pocket Tunes 2.2, this is always true.

ConditionTurnOffScreenPossible [read-only]

New in Pocket Tunes 2.2
This condition is true whenever it is possible to turn off the screen. Essentially, this is whenever a song is playing and the user has the preference item enabled that allows turning off the screen. It is always false on models that don't support screen turn-off.

ConditionPrefBassBoost [read-write]

New in Pocket Tunes 2.2
This value is true if the "Bass Boost" option is selected.

ConditionSongHasAlbumArt [read-only]

New in Pocket Tunes 4.0
This condition is true whenever the currently playing song has an album art image associated with it.

Integer Variable Reference

This section contains a list of all the integer values and what they represent.

IA – IZ [read-write]

These 26 values are for use by the skin for storing any transient integer values. When the skin is loaded (or when Pocket Tunes starts), these values are all initialized to 0.

IPA – IPZ [read-write]

These 26 values are for use by the skin for storing any persistent integers. Unlike IA – IZ, these values are remembered even if the user quits Pocket Tunes. However, if the user picks a new skin, these values are destroyed. When the skin is first loaded, these are initialized to 0.

IntegerPlaylistLocation [read-write]

This value contains the index of the current song within the current playlist. 0 is the first song. If you change this value, Pocket Tunes will skip to the index specified. If Pocket Tunes was previously playing, it will begin playing the new track; otherwise, it will just display the new track's song information.

IntegerNumInPlaylist [read-only]

This value contains the number of songs in the current playlist.

IntegerVolume [read-write]

This value contains the current volume setting.

IntegerVolumeIncrement [read-only]

This value contains the amount by which the volume is increased when using the 5-way navigator to increase it.

IntegerVolumeMax [read-only]

This value contains the maximum volume. Note that this value may be different on different devices. Also note that this value may change while Pocket Tunes is running. (This is due to obscure legal reasons about the maximum volume allowed from handheld devices in France. Don't ask!)

IntegerVolumeMin [read-only]

This value contains the minimum volume. When set to this value, Pocket Tunes will be muted (silent).

IntegerBalance [read-write]

This value contains the current balance setting. This controls the left-right panning of the audio. The center value for the balance is: (IntegerBalanceMin + IntegerBalanceMax) / 2

IntegerBalanceMin [read-only]

This value contains the minimum (left-most) balance setting.

IntegerBalanceMax [read-only]

This value contains the maximum (right-most) balance setting.

IntegerBalanceIncrement [read-only]

This value contains a reasonable increment value for incrementing/decrementing the balance setting.

IntegerSongPosition [read-write]

This value contains the current song position in milliseconds (1/1000 of a second). Obviously this value updates very frequently. When you set this value, Pocket Tunes will seek to the specified position. (If the file was being played, it will continue playing at the new position.)

IntegerSongLength [read-only]

This value contains the total number of milliseconds for the current song.

IntegerSongPositionHundredMinutes [read-only]

This is a convenience value that contains the 100's of minutes portion of the song position. For example, if your current position is 50:32, this value is 0. If your current position is 104:10, this value is 1.

IntegerSongPositionMinutes [read-only]

This is a convenience value that contains the current minutes value for the song position. For example, if your current song position is 50:32, this value is 50. If your current position is 104:10, this value is 4.

IntegerSongPositionSeconds [read-only]

This is a convenience value that contains the current seconds value for the song position. For example, if your current song position is 50:32, this value is 32.

IntegerKhz [read-only]

This value contains the KHz sampling rate for the current audio file.

IntegerKbps [read-only]

This value contains the KBps (kilobits-per-second) value for the current audio file. This value can change while playing variable-bitrate audio files.

IntegerPrefTurnOffScreenSeconds [read-write]

This value contains the number of seconds after which the screen will turn off. This is ignored unless the ConditionPrefTurnOffScreen condition is true.

IntegerVolumeBoost [read-write]

This value contains the volume boost setting, where:

- 0 = off
- 1 = +3dB
- 2 = +6dB

- 3 = +9dB
- 4 = +12dB
- 5 = +15dB

Note that volume boost is not always available, due to legal restrictions on the maximum volume that a device can produce in some countries (France).

IntegerPrefBackgroundPenFromIndex [read-write]

This value contains the background preferences “pen swipe from” choice, where:

- 0 = Apps
- 1 = Menu
- 2 = Favorites
- 3 = Find

IntegerPrefBackgroundPenToIndex [read-write]

This value contains the background preference “pen swipe to” choice, where:

- 0 = Apps
- 1 = Menu
- 2 = Favorites
- 3 = Find
- 4 = Screen
- 5 = Graffiti

IntegerPrefBackgroundButtonIndex [read-write]

This value contains the background preference “Bring up console with button” choice, where:

- 0 = Button 1
- 1 = Button 2
- 2 = Button 3
- 3 = Button 4
- 4 = Up
- 5 = Down
- 6 = Left
- 7 = Right
- 8 = Voice Memo

IntegerPrefCrossfadeMilliseconds [read-write]

This value contains the crossfade duration in milliseconds. This value is ignored unless the ConditionPrefEnableCrossfade condition is true. This value may range from 5 to 10000.

IntegerPrefStopPlayingAfterXMinutes [read-write]

This value contains the number of minutes after which Pocket Tunes will stop playing. This value is ignored unless the ConditionPrefStopPlayingAfterXMinutes condition is true.

IntegerBatteryLevel [read-only]

This value contains the current level of the battery. 0 is empty.

IntegerMaxBatteryLevel [read-only]

This value contains the battery level when fully charged.

IntegerTrialDaysLeft [read-only]

This value contains the number of days left in the user's trial, or 0 if it's expired. This value contains 0 if the user has a valid registration code.

IntegerRandom [read-only]

This value contains a different random number whenever it's referenced. Use the % (modulus) operator to generate a random number within a certain range.

IntegerTimerTicks [read-only]

This value contains the number of timer ticks that have occurred. There are usually between 60 and 100 ticks per second, but this varies on different devices.

IntegerTimerSeconds [read-only]

This value contains the number of seconds since a certain date in the past. Use this for performing timing-related events.

IntegerTimeHourZeroToTwentyThree [read-only]

This value contains the current hour of the day, in 24-hour format: from 0 to 23.

IntegerTimeHourTwelveToEleven [read-only]

This value contains the current hour of the day, in 12-hour format: from 12 to 11.

IntegerTimeMinute [read-only]

This value contains the current minute of the time.

IntegerTimeSecond [read-only]

This value contains the current second of the time.

IntegerTimeMonth [read-only]

This value contains the current month (between 1 and 12).

IntegerTimeDayOfMonth [read-only]

This value contains the current day of the month (between 1 and 31).

IntegerTimeYear [read-only]

This value contains the current year.

IntegerScreenWidth [read-only]

This value contains the current screen width in pixels. This number is the low-density equivalent of the screen width. In other words, a typical square screen has a width of 160, even on high-resolution devices. This value can change if the user opens or closes the dynamic input area, on devices that support a dynamic input area.

IntegerScreenHeight [read-only]

This value contains the current screen height in pixels. This number is the low-density equivalent of the screen width. In other words, a typical square screen has a height of 160, even on high-resolution devices. This value can change if the user opens or closes the dynamic input area, on devices that support a dynamic input area.

IntegerEqualizerMin [read-only]

New in Pocket Tunes 2.2
This is the minimum value allowed for each equalizer band.

IntegerEqualizerMax [read-only]

New in Pocket Tunes 2.2
This is the maximum value allowed for each equalizer band.

IntegerEqualizerNeutral [read-only]

New in Pocket Tunes 2.2
This is the neutral value for each equalizer band.

IntegerEqualizer1 [read-write]

IntegerEqualizer2 [read-write]

IntegerEqualizer3 [read-write]

IntegerEqualizer4 [read-write]

IntegerEqualizer5 [read-write]

New in Pocket Tunes 2.2
These are the values of the equalizer bands.

IntegerEqualizerNumBands [read-only]

New in Pocket Tunes 2.2
This is the number of bands in use with the current equalizer. (In Pocket Tunes 2.2, this is 5.)

IntegerSongYear [read-only]

New in Pocket Tunes 3.0.6
This is the year the song was produced. This value is extracted from the song's metadata. If it doesn't exist, the value will be 0.

IntegerSongTrack [read-only]

New in Pocket Tunes 3.0.6
This is the track number for the current song. This value is extracted from the song's metadata. If it doesn't exist, the value will be 0.

IntegerPlaylistDisplayPos [read-write]

New in Pocket Tunes 3.0.9
This is the position in the playlist of the first song being displayed in the playlist control on the screen. The user can scroll within this list without changing the current selection, so this value can change when this occurs. The list can be scrolled by changing this value. If there is more than one playlist control visible on the screen, this value is undefined.

IntegerPlaylistNumSongs [read-only]

New in Pocket Tunes 3.0.9
This is the number of songs in the current playlist.

IntegerPlaylistNumLines [read-only]

New in Pocket Tunes 3.0.9
This is the number of lines visible in the active playlist control. This can be used to calculate the maximum display position for the playlist, for example. This value can change if the current font size changes. If there is more than one playlist control visible on the screen, this value is undefined.

String Variable Reference

This section contains a list of strings and what they represent. All strings are read-only.

StringSongFilename

This string contains the current filename (or database name, URL, etc.) of the file being played.

StringSongTitle

This string contains the current title of the song being played. This value is retrieved from the ID3 tags in MP3 files, or the info stored in Ogg files. If there is no title, the filename is used instead.

StringSongComments

This string contains the current comments of the song being played. This is a multi-line string that includes lots of information about the song being played. This information is extracted from the ID3 tags for MP3s or the info stored in Ogg files. If there are not comments, the filename is used instead.

Note: If you have a text area that displays this value, your text area can be converted into a playlist if the user selects the “Show playlist in comment area” preference.

StringSongTitleAndComments

This string contains a combination of StringSongTitle and StringSongComments.

StringSongLength

This string contains the duration of the current song, formatted like this: MM:SS. For example, “53:12” or “3:02”.

StringSongType

This string contains an abbreviated name for the type of music being played. For example, it could contain “mp3”, “ogg”, “wav”, or some other format.

StringCurrentTime

This string contains the current time, formatted using the default settings for the Palm. (set these from the Formats preferences panel.)

StringPlaylists

This is a special string whose value is empty. If you have a text area set to this value, then it will display a “playlist control” instead of the text area control. The playlist control displays a list of all the songs in the current playlist and allows the user to scroll around in this list and pick a new song to be played. While this control is displayed, no Clicked events will be processed for the text area. Also note that the currently selected song will be displayed using the background color set for this control.

StringTimeAMPM

This string contains either “am” or “pm”, depending on the current time of day.

StringSongChannels

This string contains either “stereo” or “mono” (could be translated into a different language), depending on the current song. If no song is playing, this string is blank.

StringSongArtist

New in Pocket Tunes 3.0.6
This is the artist of the current song. This value is extracted from the song’s metadata. If it doesn’t exist, the value will be an empty string.

StringSongAlbum

New in Pocket Tunes 3.0.6
This is the album of the current song. This value is extracted from the song’s metadata. If it doesn’t exist, the value will be an empty string.

StringSongGenre

New in Pocket Tunes 3.0.6
This is the genre of the current song. This value is extracted from the song’s metadata. If it doesn’t exist, the value will be an empty string.

StringPlaylistName

New in Pocket Tunes 3.0.9
This is the name of the current playlist. If a playlist is selected, the name is taken from the playlist’s name. If something else is selected, this may contain other information, such as “All Content”, the name of the artist or album selected, “2 Songs”, or something else.

StringSongPosition

New in Pocket Tunes 4.0
This is the playback position within the current track, formatted as HHH:MM:SS.

StringSongPositionCountdown

New in Pocket Tunes 4.0
This is the remaining time of the current track, formatted as HHH:MM::SS. This value will always be 0:00 for stream (i.e., non-file) content.

Action Reference

This section contains a list of all the actions available and a description of how to use them.

SetCondition CVar CVal

This action takes two parameters: the name of a condition variable and a condition expression. The action sets the condition variable to the given condition expression.

SetInt IVar IVal

This action takes two parameters: the name of an integer variable and an integer expression. The action sets the integer variable to the given integer expression.

ClickButton

Currently not implemented.

IncrementSlider

Currently not implemented.

DecrementSlider

Currently not implemented.

EnableControl

Currently not implemented.

SetTextAreaScroll

Currently not implemented.

ClickTextArea

Currently not implemented.

Play

If the audio is paused or stopped, begin playing.

PlayPause

If the audio is paused or stopped, begin playing. If the audio is playing, then pause.

Pause

If the audio is playing, pause it. If the audio is paused, start playing.

PauseIfPlaying

If the audio is playing, pause it. Otherwise, do nothing.

Stop

If the audio is playing or paused, stop it.

PreviousTrack
Switch to the previous track in the playlist.

NextTrack
Switch to the next track in the playlist.

ChoosePlaylist
Open the “Playlists” form.

Open
Open the “Choose Songs” form.

TurnOffScreen
Turn off the screen. This only works if the audio is playing and the preferences enable screen turn-off.

ShowSongInfo
Display a form with the title and comments for the current song.

ShowGeneralPrefs
Display the general preferences form.

ShowBackgroundPrefs
Display the background preferences form.

ShowCrossfadePrefs
Display the crossfade preferences form. An error will be displayed if the user does not have Pocket Tunes Deluxe.

ShowBookmarkPrefs
Display the bookmark preferences form. An error will be displayed if the user does not have Pocket Tunes Deluxe.

ShowAdvancedPrefs
Display the advanced preferences form.

AddBookmark
Add a bookmark at the current location. This may display a form asking for the bookmark name, depending on the user’s preferences. An error will be displayed if the user does not have Pocket Tunes Deluxe.

ShowBookmarks
Display the bookmarks form. An error will be displayed if the user does not have Pocket Tunes Deluxe.

ShowEqualizer

Display the equalizer form.

ShowChooseSkin

Display the choose skin form. Note that, unlike all the other actions, this action does not get executed immediately. Instead, Pocket Tunes waits until all the other controls have finished updating and handling any user events, and then the skin is destroyed and the Choose Skin form is displayed.

ShowAbout

Display the About Pocket Tunes form.

ShowMenu

Display the Pocket Tunes menu. Note that this does not get executed immediately. The menu will be displayed after all the other controls have finished updating and handling any user events.

ScanForeward

Begin (or continue) scanning forward within the current audio file. After you are finished scanning, you must execute the DoneScanning action.

ScanBackward

Begin (or continue) scanning backward within the current audio file. After you are finished scanning, you must execute the DoneScanning action.

DoneScanning

Let Pocket Tunes know that you have finished scanning forward/backward in the audio file.

ShowSkinPrefs

Display the Skin Preferences form.

ForceRefresh

New in Pocket Tunes 2.2
As soon as possible, redraw all of the controls. You should strive to never have to use this action; it can cause a lot of flickering, which is very annoying to users. You should also ensure that you don't end up calling this in an endless loop, as it can cause Pocket Tunes to become unusable.

ShowAlbumArt

New in Pocket Tunes 4.0
Show the currently playing song's album art image in full-screen mode.

ShowInternetRadio

New in Pocket Tunes 4.0
Open the Internet Radio tuner form.