Skin System
SharpEnviro shell replacement

written by

Martin Krämer

martin@sharpenviro.com

September 14, 2010

Contents

1 Introduction

This document will help you to understand how the SharpEnviro skin system works so that you can create your own skins. Creating skins isn't very difficult so we want to encourage you to try it out and create some new cool skins for SharpEnviro. If you have created a new skin you can share it with other users in our forum or you can contact the developers so that maybe your skin will be included in the next release of SharpEnviro or a possible update packages.

How does the skin system work?

The skin system developed for SharpEnviro is a XML based bitmap layer-based rendering system. A skin defines a rendering path from the top-most bitmap layer to the bottom layer and in the end all those layers will be merged together. The layers however aren't simple windows bitmaps, we are using 32bit bitmaps as layers which fully support transparency and alpha blending. Therefore, all images used in a skin must be saved in PNG format with transparency (32bit PNG). We also recommend you save the PNG files with the normal RGB color palette (not grayscale!).

A SharpEnviro skin is divided into several skin components which together form the complete skin. Those components are skinned independently from each other and a valid SharpEnviro skin must contain skins for all of the components.

- Buttons
- Labels/Captions
- Edit Boxes
- Menu background
- Menu items
- Notification Popups
- Progress bars
- SharpBar background
- SharpBar mini config buttons
- Taskbar items
- Taskbar preview windows
- Menu Item

How is the coloring of the skins working?

One of the more exciting features of the skinning system is the support for color schemes. SharpEnviro supports the use of color schemes which allow the colors of certain skin parts to be changed by the user. Which parts of the skin can be controlled and colored is up to the skin developer. Besides the colors itself the scheme system also allows for other properties like the alpha value of the skin to be changed. You can for example add a scheme property which would enable the dynamic changing of the background transparency for any skin components. Adding support for schemes is not a required part of a skin, but supporting scheme colors and values in your skin is a great feature and highly recommended. With schemes, you control the skin, but the end-user can define the colors and final look of it.

How to get started?

The best way to get started on creating a new skin for SharpEnviro is to modify an already existing skin. Chose the skin that matches your ideas and visions for a new skin best and start to modify this skin. By doing so you can very quickly see results even of small changes. However we still recommend to read this entire document for getting a better understanding about how the skin system and especially the advanced features work.

2 Skin System

2.1 File Structure

Each skin gets its own directory within the *SharpEnviro\Skins\* folder
SharpEnviro\Skins\MySkin\

Within the directory of the skin three XML files are to be created
SharpEnviro\Skins\MySkin\info.xml
SharpEnviro\Skins\MySkin\scheme.xml
SharpEnviro\Skins\MySkin\skin.xml

Info.xml - Skin Header and Information File

This file is a very simple XML file which contains nothing more than the name of the skin, the authors name, authors website, info description and the version of the skin. An example file would look like this:

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<SharpESkinInfo>
  <header>
    <name>Number 8</name>
    <author>Martin Krämer</author>
    <url>http://www.sharpenviro.com</url>
    <info>Big skin similiar to Windows 7</info>
    <version>0.8</version>
  </header>
</SharpESkinInfo>
```

Code Example 2.1: Header file for the skin (**info.xml**).

Scheme.xml - Scheme Color Definitions

The scheme color XML file defines which scheme color and scheme values will be available for the skin. Those are not only the values the user can change by adjusting their color schemes, but those are also the values and tags which a skin developer can use while creating a skin. When you create a skin it's recommended to take a look at this file first because you can use all the scheme tags created here later when creating the skin. A more detailed description of the scheme color system can be found in chapter 2.2.

Skin.xml - Skin Components

This file contains the actual skins for all the components which can be skinned. The basic structure of the file is very simple, under the root XML element each skin component gets it's own element under which the actual skin informations are stored.

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<SharpESkin>
  <SharpBar>...</SharpBar>
  <Menu>...</Menu>
  <MenuItem>...</MenuItem>
  <TaskItem>...</TaskItem>
  <MiniThrobber>...</MiniThrobber> <!-- SharpBar mini config buttons -->
  <Button>...</Button>
  <Font>...</Font> <!-- Captions and Labels -->
  <ProgressBar>...</Progressbar>
  <Edit>...</Edit> <!-- Edit Box -->
  <Notify>...</Notify>
  <TaskPreview>...</TaskPreview>
</SharpESkin>
```

Code Example 2.2: Basic structure of the file **skin.xml**.

The skin components itself can have different sub parts which represent different states or parts of that component. A Button for example can have a part which defines how it looks when it's clicked and how it will look when it's not clicked. Which skin component parts exist depends on the skin component and will be discussed later for each component in detail. The following example shows how a button skin is structured.

```xml
[...]
  <Button>
    <Text>...</Text> <!-- Text Settings -->
    <Icon>...</Icon> <!-- Icon Settings -->

    <Normal>...</Normal> <!-- Skin for a button in its normal state -->
    <Hover>...</Hover> <!-- Skin for a button when the mouse is over it -->
    <Down>...</Down> <!-- Skin for when a button is clicked -->
  </Button>
[...]
```

Code Example 2.3: Structure of the skin for the button skin component. The skin consists of three different sates: **Normal**, **Hover**, **Down**. Which state is used for drawing the button is based on the status of the mouse.

2.2 Scheme Colors

The available scheme colors of a skin are defined within the **scheme.xml** file in the skins directory. The items of this XML file are a list of the scheme values (see code example 2.4) which a skin author wants to use. Therefore it's recommended to edit and adjust this file before creating the skin.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<SharpESkinScheme>
  <item>...</item>
  <item>...</item>
  <item>...</item>
</SharpESkinScheme>
```

Code Example 2.4: List of all scheme items in the **scheme.xml** file.

Each scheme item consists of 5 tags which specify the properties of the item, those tags are shown in table 2.1.

Tag	Description
<Name>	the name of the scheme item
<Tag>	tag by which the item can be used when creating the skin
<Type>	type of the scheme item: Color, Boolean, Integer, Dynamic
<Info>	short Info Tag what the item changes (will be displayed in the scheme editor)
<Default>	default value which is used when there is no custom user scheme selected

Table 2.1: List of all tags which specify the properties of a scheme item.

The four possible scheme item types are **Color**, **Integer** and **Dynamic** whereas color and integer are the most important options. The dynamic option is only used in a special case.

Using a scheme item in the skin itself is simply done by inserting the tag of color as it is defined in the **scheme.xml** file. For example by writing $Background.

2.2.1 Type: Color

With the color type the scheme item will specify a simple color which can be used to change the color of certain skin parts. The default color for this item can either be an integer value representing the RGB color value, a Delphi string name like **clRed**, **clWhite**, a string containing single **RGB**, **HSL**, **CYMK** or **HSV** values in the form like **RGB(234,65,94)** or a hexadecimal value like #**43D2AF**. More details about the possible color codes can be found in chapter ....

```
<item>
  <Name>Background</Name>
  <Tag>$Background</Tag>
  <Info>Background Color</Info>
  <Default>RGB(64,32,0)</Default>
</item>
```

Code Example 2.5: Declaration of the scheme type **color**.

2.2.2 Type: Integer

The integer type is used to control the alpha/transparency value of skin parts and it can take values between 0 and 255.

```
<item>
  <Name>Alpha</Name>
  <Tag>$Alpha</Tag>
  <Info>Background Alpha</Info>
  <Default>196</Default>
  <type>Integer</type>
</item>
```

Code Example 2.6: Declaration of the scheme type **integer**.

2.2.3 Type: Dynamic

Dynamic scheme items cant be controlled or changed by the user, those are special items which represent colors which are dynamically changed at runtime by the source code. Currently the only usage for dynamic scheme items is to have a scheme color which represents the color of the associated icon of a button or taskbar item. A scheme item with type **dynamic** and tag $IconHighlight will dynamically change it's color to represent the average color of an associated icon. This can for example be used to change the background color of a button to match the color of the assigned icon. Note that the Tag of this dynamic item is fixed and has to be $IconHighlight.

```
<item>
  <Name>Icon Highlight</Name>
  <Tag>$IconHighlight</Tag>
  <Info>Icon Highlight Color</Info>
  <Default>clWhite</Default>
  <type>Dynamic</type>
</item>
```

Code Example 2.7: Declaration of the scheme type **dynamic**.

List of code examples

List of Tables