HOWTO GUIDE
CUSTOMISING THE KIOSK SKIN

Vista Cloud

Vista Entertainment Solutions
2022-03-15

for the ♥ of cinema

Copyright notice

Vista Entertainment Solutions Limited

PO Box 90551, Victoria Street West, Auckland 1124, New Zealand

P: +64 9 984 4570 **F**: + 64 9 379 0685 www.vista.co

Contents

Overview

Kiosk supports the customisation of colours, fonts, templates, and control styles.

The following sections provide high-level guidance for the steps involved, and describe functions that can support the creation of a custom theme. Vista recommends that you familiarise yourself with **XAML** concepts before attempting to alter the look and feel of the application.

Customisation does not require all files or values to be provided. Partial file and value provision is supported. For example, if the file `Colors.xaml` is copied to the folder `\User`, and only the background colour needs to be changed, then the other values can be safely deleted. The same is true for templates and control styles, as the system has a fallback default theme that is used when no customised value has been provided.

In all cases, Kiosk must be restarted for the changes to take effect.

Notes:

- The default styles and templates in Kiosk are designed to work primarily with its custom controls and models. To ensure that application settings are honoured (for example, those defined in the Kiosk.ini file), use the custom converters and visualisation models provided by the Kiosk API in preference to other tools.
- Vista recommends that you copy only the files that you want to customise (not every file) from `VistaNew/Base` to `VistaNew/User`.

Customising colours

1. Locate the file `Colors.xaml` in the **Kiosk** workstation folder:
   `ProgramData\Vista\VistaKiosk\Config\Skins\Type1\VistaNew\Base`.

2. Save a copy of the file in the folder `\User`.

   Note: Vista recommends that you copy only the files that you want to customise (not every file) from `VistaNew/Base` to `VistaNew/User`.

3. Open the file and locate the element you want to edit.

   For example: The default background colour for the application:

   ```
   <SolidColorBrush x:Key="MainBackgroundBrush" Color="#FF2B2725"/>
   ```

4. Replace the `Color` value with the desired colour name or hexadecimal value.

   For example:

   ```
   <!-- the color is now Dodger Blue -->
   <SolidColorBrush x:Key="MainBackgroundBrush" Color="#FF1E90FF"/>
   ```

Microsoft provides colour swatches for common colours at http://msdn.microsoft.com/en-us/library/system.windows.media.colors(v=vs.110).aspx.

Customising fonts

1. Locate the file `Fonts.xaml` in the **Kiosk** workstation folder:
   `ProgramData\Vista\VistaKiosk\Config\Skins\Type1\VistaNew\Base`.

2. Save a copy of the file in the folder `\User`.

Note: Vista recommends that you copy only the files that you want to customise (not every file) from `VistaNew/Base` to `VistaNew/User`.

3. Open the file and locate the element you want to edit.

   For example: The default font family for the application:

```
<!-- the font family used for all text elements -->
<Style x:Key="RegularFontStyle" TargetType="{x:Type IFrameworkInputElement}">
    <Setter Property="TextElement.FontFamily" Value="Segoe UI"/>
</Style>
```

4. Replace the `Font` value with the name of any font installed on the system.

   For example:

```
<Setter Property="TextElement.FontFamily" Value="Tahoma"/>
```

Customising templates

The template files found in the **Kiosk** workstation folder `ProgramData\Vista\VistaKiosk\Config\Skins\Type1\VistaNew\Base\Templates` support the customisation of visual elements used by the views and tile templates hosted in the carousel control.

A view is an area of the screen dedicated to presenting the user interface for a specific function. For example: film selection, language selection, and location selection are all views. The names of the **XAML** files in the folder `Templates` correspond to the views they support.

To customise a template, locate the required file in the folder `\Base\Templates`, and save a copy of it in the folder `\User\Templates` for editing.

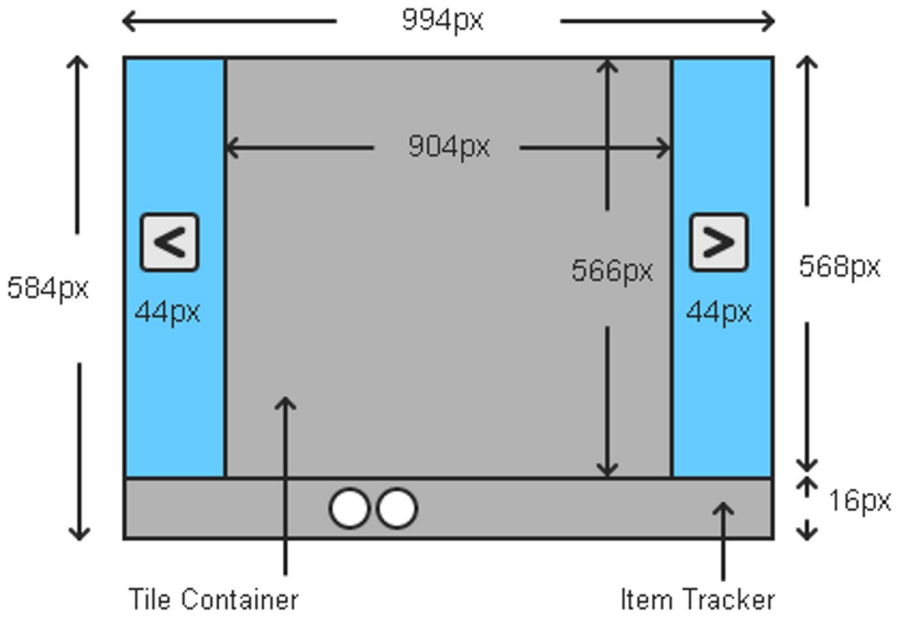
Note: *Vista recommends that you copy only the files that you want to customise (not every file) from `VistaNew/Base` to `VistaNew/User`.*

The number of tiles presented in the carousel is managed via the tile template size. The tile size is used to calculate the number of pages required to present the items. Therefore, it is important to understand the carousel control layout, and how to calculate sizes that will provide the desired layout.

Note: *If the default carousel control layout is customised, the tile templates hosted in each view may also need to be altered.*

For layouts that will display more tiles than can be presented on a single page, the tile size calculations are based on a carousel control state where both the 'previous' and 'next' buttons are visible.

The default carousel control layout is defined as follows:

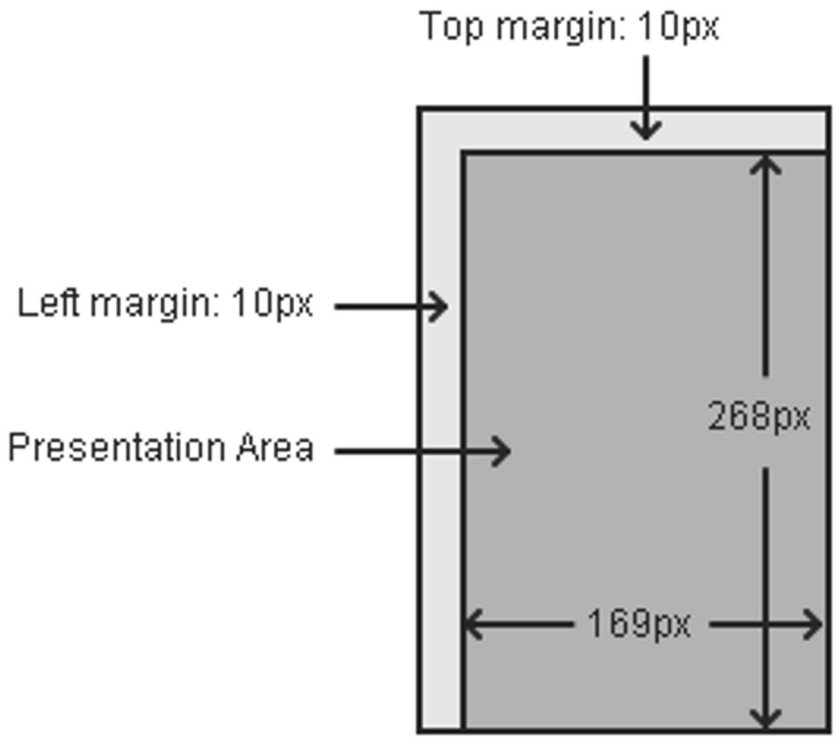


For the purpose of calculating tile dimensions, the area of interest is the tile container.

For tile layouts that require multiple pages, the tile layout should be based on the number of tiles of equal size that can fit into a single page hosted in an area that measures as 566px x 904px.

If the carousel will only present a single page of tiles, then the tile container area measures 566px x 992px (the container control border is 1px thick).

Tile templates should be defined as having a left and top margin value, with a right and bottom margin value of zero. The following wireframe layout represents the default film tile template in the pick film view carousel.



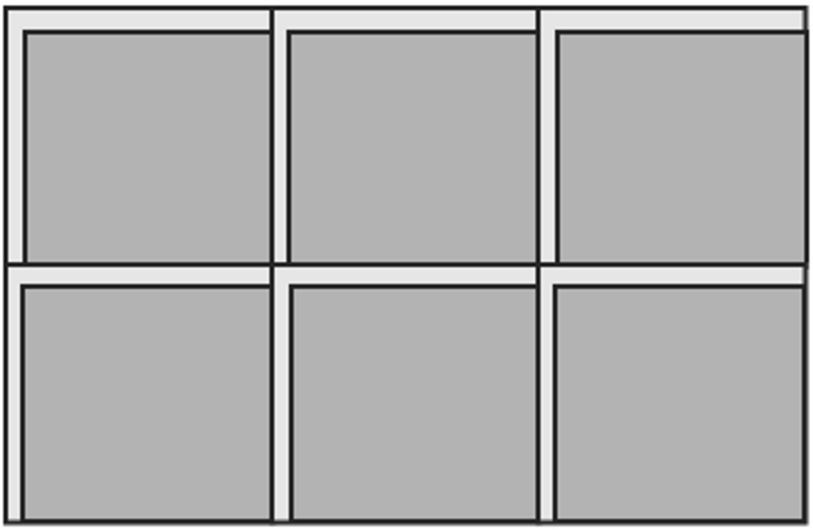
To calculate the tile dimensions:

1. Decide on the top margin value that should separate the tiles on the vertical axis, and multiply this value by the number of rows + 1 row. Subtract the result from 566, and divide the remainder by the number of rows required. This will give you the tile height.

2. Decide on the left margin value that should separate the tiles on the horizontal axis, and multiply this value by the number of tiles that should appear in a row + 1 tile. Subtract the result from 904, and divide the remainder by the number of required tiles in the row. This will give you the tile width.

Example calculation for six tiles presented as two rows of three

The required number of tiles is six, presented as two rows with three tiles in each row, where the top and left margin values are both 10px. This can be visualised as follows:



Calculation

Total vertical margin height: 10px * 3 = 30px (the top margin (10px) times the number of required rows (2) plus one).

Available tile container height: 566px – 30px = 536px (the tile container height minus the total vertical margin height).

Individual tile height: 536px / 2 = 268px (the available tile container height divided by the number of rows required).

Total horizontal margin width: 10px * 4 = 40px (the left margin (10px) times the number of required tiles in a row (3) plus one).

Available tile container width: 904px – 40px = 864px (the tile container width minus the total horizontal margin width).

Individual tile width: 864px / 3 = 288px (the available tile container width divided by the number of tiles required in a single row).

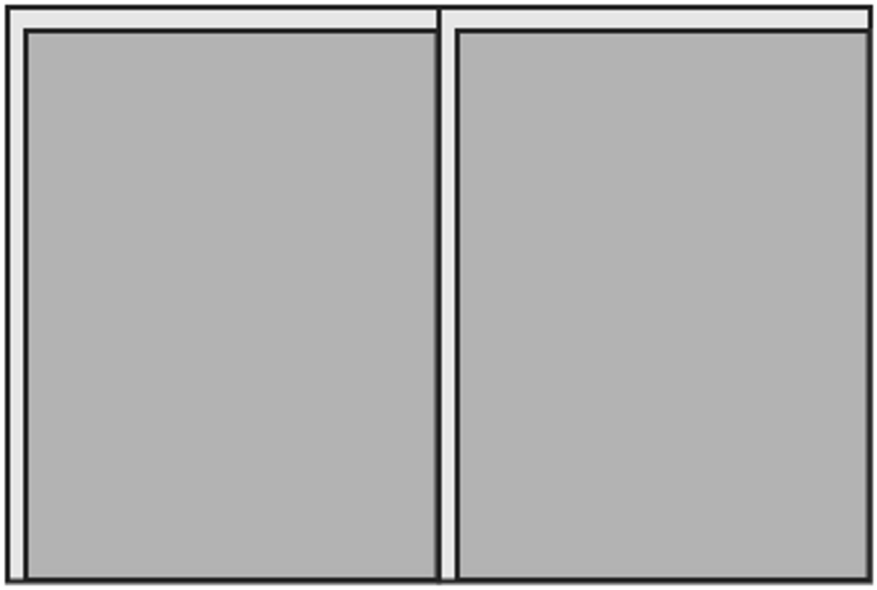
Final tile dimensions: left margin: 10px, top margin: 10px, height: 268px, width: 288px.

In XAML this example would be expressed as follows:

```
<Style x:Key="PickFilmTileStyle" TargetType="{x:Type Grid}">
       <Setter Property="Height" Value="268"/>
       <Setter Property="Margin" Value="10,10,0,0"/>
       <Setter Property="Width" Value="288"/>
</Style>
```

Example calculation for two tiles presented as one row of two

The number of required tiles is two, presented as one row with two tiles, where the top and left margin values are both 10px. This can be visualised as follows:



Calculation

Total vertical margin height: 10px * 2 = 20px (the top margin (10px) times the number of required rows (1) plus one).

Available tile container height: 566px – 20px = 546px (the tile container height minus the total vertical margin height).

Individual tile height: 546px / 1 = 546px (the available tile container height divided by the number of rows required).

Total horizontal margin width: 10px * 3 = 30px (the left margin (10px) times the number of required tiles in a row (2) plus one).

Available tile container width: 904px – 30px = 874px (the tile container width minus the total horizontal margin width).

Individual tile width: 874px / 2 = 437px (the available tile container width divided by the number of tiles required in a single row).

Final tile dimensions: left margin: 10px, top margin: 10px, height: 546px, width: 437px.

In XAML, this example would be expressed as follows:

```
<Style x:Key="PickFilmTileStyle" TargetType="{x:Type Grid}">
       <Setter Property="Height" Value="546"/>
       <Setter Property="Margin" Value="10,10,0,0"/>
       <Setter Property="Width" Value="437"/>
</Style>
```

Customising control styles

Important: *Manipulation of control templates is considered an advanced area of customisation and should not normally be required. Refer to online resources that describe custom controls and control templates before altering the default definitions.*

Example customisation of the Kiosk banner

In this example, we will manipulate the **Kiosk** banner control template so that the logo is moved to the right and placed next to the language selector, while the current date and time are moved to the far left.

By default, the Kiosk banner looks like this:

For brevity, only the elements of the template that are to be manipulated are reproduced here.

```
<vkv:AdminModeButton Grid.Column="0" Grid.ColumnSpan="4"
       IconImageSource="{TemplateBinding BannerImageSource}"
       Style="{StaticResource AdminModeButtonStyle}"
       Stylus.IsPressAndHoldEnabled="False"/>
<StackPanel Grid.Column="2">
       <TextBlock
              FontSize="25"
              Foreground="{TemplateBinding Foreground}"
              Text="{TemplateBinding CurrentTime}"
              HorizontalAlignment="Right"/>
       <TextBlock
              Foreground="{TemplateBinding Foreground}"
              FontSize="15"
              HorizontalAlignment="Right"
              Margin="0,-5,0,0"
              Text="{TemplateBinding CurrentDate}"/>
</StackPanel>
```

1. Locate the file `KioskBannerStyle.xaml` in the Kiosk workstation folder `Vista\VistaKiosk\Config\Skins\Type1\VistaNew\Base\ControlStyles`.

2. Save a copy of the file in the folder `\User\ControlStyles`.

3. Edit the file so that it looks like this:

```
<StackPanel Grid.Column="1" HorizontalAlignment="Left">
    <TextBlock
           FontSize="25"
           Foreground="{TemplateBinding Foreground}"
           Text="{TemplateBinding CurrentTime}"
           HorizontalAlignment="Right"/>
    <TextBlock
           Foreground="{TemplateBinding Foreground}"
           FontSize="15"
           HorizontalAlignment="Right"
           Margin="0,-5,0,0"
           Text="{TemplateBinding CurrentDate}"/>
</StackPanel>
<vkv:AdminModeButton Grid.Column="2"
    IconImageSource="{TemplateBinding BannerImageSource}"
    Style="{StaticResource AdminModeButtonStyle}"
    Stylus.IsPressAndHoldEnabled="False"/>
```

4. Restart the application, and the Kiosk banner will look like this:

Customising value converters

In addition to customising the look of visual elements through standard **XAML** notation, there is also a collection of custom value converters that can be used. These converters can manipulate values or visual elements to avoid complex expressions, and simplify presentation scenarios.

1. To use one or more of the custom value converters, add the following namespace to the top of the XAML file being edited.

   ```
   xmlns:vkv=http://schemas.vista.co.nz/ui/visualisation
   ```

2. Declare the required converter type to use.

   ```
   <vkv:StringToUpperConverter x:Key="StringToUpperConverter"/>
   ```

3. Use the converter to perform a type conversion between the source (Title) and target (Text) values.

   ```
   Text="{Binding Path=Title, Converter={StaticResource StringToUpperConverter}}"
   ```

Converters - reference information

Converter name	Description
`BoolToBrushConverter`	Converts a Boolean value to a brush resource value. Returns the application resource brush identified by the `PositiveBrushKey` value if a Boolean value is not supplied, or if the supplied value is 'true'. Otherwise it returns the application resource brush identified by the `NegativeBrushKey`.
`BoolToCollapsedVisibilityConverter`	Converts a Boolean value to a visibility value. Returns **Visible** if a Boolean value is not supplied or if the supplied value is 'true'. Otherwise it returns **Collapsed**.
`BoolToHiddenVisibilityConverter`	Converts a Boolean value to a visibility value. Returns **Visible** if a Boolean value is not supplied or if the supplied value 'true'. Otherwise it returns **Hidden**.
`CollectionCountToVisibilityConverter`	Converts a collection item count value to a visibility value. Returns **Visible** if a collection is not supplied or if the item count of the supplied value is greater than zero. Otherwise it returns **Collapsed**.
`DateTimeToLongDateConverter`	Converts a date–time value to the long date format string. For example: Thursday, 7th Nov 2013.
`DateTimeToMediumDateConverter`	Converts a date–time value to the medium date format string. For example: Thu 7 Nov.
`DateTimeToTimeConverter`	Converts the time part of a date to the 12-hour string format.
`DateTimeToTimePeriodConverter`	Converts the time period part of the date to the time period designator string format. For example: AM.
`DecimalToCurrencyConverter`	Converts a decimal value to the configured application currency format. Returns an empty string if the value supplied is not of decimal type.
`ImageSourceToFillConverter`	Converts an image source value to a media stretch value. Assesses the image dimensions in reference to the image container to determine an appropriate stretch mode that will scale the image for optimal presentation. This converter has been developed for use with tile templates, although it can be applied to other use cases if required.
`ImageSourceToVisibilityConverter`	Converts the supplied image source value to a visibility value. Returns **Visible** if the `ImageSource` has a value. Otherwise it returns **Collapsed**.
`ItemsControlSeparatorVisibilityConverter`	Determines the visibility of a separator element used in an item's control based on the position of the item in a source collection. It is assumed that the separator is visible unless it is participating in the layout for the first item in the parent collection. Use the `InvisibilityMode` property to determine whether the separator is collapsed or hidden when not visible.
`RemainingShowTimesToRunsConverter`	Converts the number of show times remaining for a session to a **Run** collection. If the value is less than one, an empty string is returned.

`SessionDateToDayConverter`	Converts a session date to its day representation.
`SessionStateToDescriptionConverter`	Converts the session state to a string.
`SessionsToFirstAvailableConverter`	Converts a collection of type **ScheduledFilmDay** to a collection of type **Run**. This converter determines the first session available across the available scheduled **Kiosk** film days.
`SessionsToRunsConverter`	Converts a collection of type **ScheduledFilmDay** to a collection of type **Run**. This converter determines the first three sessions available across the scheduled **Kiosk** film days.
`SessionTextBlockToWidthConverter`	Converts a text block to a width that will accommodate its content in a tab control.
`StringToUpperConverter`	Converts the supplied value to an upper case string, or returns an empty string if the value supplied cannot be converted to a string.
`StringToVisibilityConverter`	Converts the supplied value to a visibility value. Returns **Visible** if the string has a value, otherwise it returns **Collapsed**.
`TabToEqualWidthConverter`	Converts a tab width to a size that will be equal for all tabs contained within a tab control.

Increasing and decreasing the visual scale of Kiosk

If required, enhance the usability of the Kiosk screen by adjusting its visual scale. In portrait mode, decreasing the scale can be useful when the physical screen is too big for users to comfortably operate. In landscape mode, increasing the scale can allow Kiosk to fill more of the screen.

1. Open the `Kiosk.ini` file in **Notepad** or a similar text editor.

   `ProgramData\Vista\VistaKiosk\Config\Kiosk.ini`

2. Locate the setting `DisplayScalePercentage`, and increase or decrease the value as needed.

   This determines the scale of your Kiosk display as a percentage of Kiosk's default size (100). The minimum allowed value is 70.

Customising the idle screen when Kiosk is scaled down

If scaled below 100% in portrait mode, **Kiosk** displays an image behind the main interface, filling the background. When the screen is idle, only the background image is visible. A small icon and message prompts users to touch the screen to activate Kiosk. You can customise these elements.

1. To change the background image, locate the folder `ProgramData\Vista\VistaKiosk\Config\Skins\Type1\User`.

2. Save your image in this folder with the name `ScaledDown_Wallpaper.png`.

3. To change the hand icon on the idle screen, save your image in the same folder with the name `touch_big.png`.

4. To add your own prompt text, locate the folder `ProgramData\Vista\VistaKiosk\Config\Language`.

5. Open the file `INTENG_KC.XML` (or the file for the language specified in the `Kiosk.ini` setting `Language1`).

6. Locate the node **<FORM NAME="Common">**.

7. Under this node, add the following text, entering your own prompt text as the value:

```
<CONTROL CODE="UnlockIdleKiosk" NAME="" VALUE="Your prompt message when kiosk is idle" LENGTH="0" TRANSLATED="TRUE" />
```

Positioning the sections of the Kiosk skin

When in **portrait mode**, **Kiosk's** on-screen content is divided into three horizontal panels. By default, the panel containing sections used for transactions is positioned in the centre. You can reposition these panels to suit your setup. For example, if your Kiosks are positioned higher on a wall, having the transactional section in the lower part of the screen increases accessibility.

1. Open the `CfgSkin.xml` file in your customised skin folder.

   For example:
   `C:\ProgramData\Vista\VistaKiosk\config\Skins\Type1\...\CfgSkin.xml`

2. Locate the `General` property list:

   Skin > Screens > General

3. Edit the numerical values for the top, centre, and bottom panels.

   The panel given the value of **0** will appear at the top, while the panel given the value of **2** will appear at the bottom.

   For example:

```
<prop name="TOPPANEL.ROWINDEX" value="0" />
<prop name="CENTREPANEL.ROWINDEX" value="2" />
<prop name="BOTTOMPANEL.ROWINDEX" value="1" />
```

   This configuration positions the panel used for transactions in the *lower* third of the screen.

Customising the text field size for card numbers

When your patrons manually enter their **booking reference number** or **Loyalty card number** into **Kiosk**, the text entry field should fit the full number on a single line. If your business uses longer card numbers, adjust the **width** and **height** of the text field on both the Booking Pickup and Booking Reference pages to achieve this.

1. Open the `CfgSkin.xml` file in your customised skin folder.

   For example:
   `C:\ProgramData\Vista\VistaKiosk\config\Skins\Type1\...\CfgSkin.xml`

2. Locate the `KeyboardEntry` property list:

Skin > Screens > General > Controls > KeyboardEntry

3. Edit the values for **width** and **height**.

   See the below XML as an indication of where to find these properties:

```
<prop name="ID" value="SKIN" />
   <proplist>
     <prop name="ID" value="Screens" />
     <proplist>
       <prop name="ID" value="GENERAL" />
    ...
    <proplist>
       <prop name="ID" value="Controls" />
       ...
       <proplist>
          <prop name="ID" value="KeyboardEntry" />
          <prop name="FONTCOLOR" value="68,62,59" />
          <prop name="FONTSIZE" value="30px" />
          <prop name="LEFT" value="615" />
          <prop name="TOP" value="340" />
          <prop name="WIDTH" value="300" />
          <prop name="HEIGHT" value="56" />
       </proplist>
```

4. Use your own background image (`Keyboard_entry.png`) to match the size of the text entry field.

Customising your Loyalty card swipe image

Follow the steps below to customise the Loyalty card swipe image on **Kiosk**.

1. Locate your customised copy of `LoyaltyScanCardViewTemplate.xaml`, or create a copy of the original version and save it in the below folder.

   `C:\VistaInstall\Downloads\Kiosk_<version>\VistaKiosk\VistaKiosk\Config\Skins\Type1\VistaNew\User\Templates`

2. Within the file, locate the template that has `LoyaltyScanCardTemplate` as the `Key`.

3. At the end of this template, under the "`Main area`" comment, locate the line for the first image.

4. Delete the current properties of `AnimatedSource`.

5. Set the mark-up extension to `vkv:CustomisableImage`.

6. Decide whether you want to customise this image by specifying a setting name or a file name, and follow the appopriate steps below.

Specify the setting name:

1. Enter `SettingName` and set the value to `Loyalty_Anim_Swipe_Filename`.

This tells the image object which `Kiosk.ini` setting determines the image to be used.

2. Enter `DefaultImageName` and set the value to the file name of an image.

   This default image will be used if the image specified in the `Kiosk.ini` setting cannot be found. It can match the one specified in the `Kiosk.ini` setting.

   For example: `AnimatedSource="{vkv:CustomisableImage SettingName=Loyalty_Anim_Swipe_Filename, DefaultImageName=Anim_Swipe_Insert}"`

3. Save the file.

Specify the file name:

1. Enter `IsUsingFileName=True`

   This tells the image object to use a file name rather than a `Kiosk.ini` setting.

2. Enter `ImageName` and set the value to the file name of the image to be used.

   Note: This image must be located in `VistaKiosk\bin\Config\Skins\Type1` in either the `VistaNew` or `User` folder, according to the `Kiosk.ini` setting `SkinSubType`.

   For example: `AnimatedSource="{vkv:CustomisableImage IsUsingFilename=True, ImageName=Anim_Swipe_Insert}"`

3. Save the file.

We've converted several **Kiosk** screens from **Windows Forms** to **WPF** (Windows Presentation Foundation), a framework that allows for greater flexibility in the user interface. If you've customised any of the following Kiosk screens using the `CfgSkin.xml` file in a release *earlier than 5.0.3*, and are upgrading to 5.0.3 (or later), you'll need to replicate your customisations by modifying certain XAML files.

Screens affected by this change (as named in `CfgSkin.xml`):

- ENTERBOOKINGREF
- ENTERCCV
- ENTERPAYMENTPASSWORD
- ENTERPIN
- ENTERZIP
- ERRORUSERINFO
- INSERTVOUCHERS
- PAY
- PICKUP
- PICKUPPROCESSING
- PRINT
- PROMO
- TAXINFORMATIONCOLLECTION
- TAXNAME
- TAXNUMBER

Index