Contents

[Common Information 2](#_Toc467677595)

[Backend Access Credentials (Product Editor only) 2](#_Toc467677596)

[Backend Access Credentials (Admin) 2](#_Toc467677597)

[Kartris Download Path 2](#_Toc467677598)

[Kartris Commercial License Purchase Path 2](#_Toc467677599)

[Author Details 2](#_Toc467677600)

[Introduction 2](#_Toc467677601)

[Developers Guide 3](#_Toc467677602)

[Implementation 3](#_Toc467677603)

[Custom Server Control – ImageList 4](#_Toc467677604)

[Modifying the Kartris Code 5](#_Toc467677605)

[Change Log 7](#_Toc467677606)

Kartris Image Swatch Selector

Allowing end users to change the main product images through an image option control

# Common Information

If you have already read the guidance note below and are looking for that password, or setup string that you cannot remember, you will find the common data you might need below:

### Backend Access Credentials (Product Editor only)

User: demo

Password: demo

### Backend Access Credentials (Admin)

User: Admin

Password: s(M#Y+mh

### Kartris Download Path

<http://www.kartris.com/t-Downloads.aspx>

### Kartris Commercial License Purchase Path

<http://www.kartris.com/Products--Services/Kartris-Commercial-License__p-2-14.aspx>

### Author Details

At the end of this document.

# Introduction

We were looking for a control that allows customers to a Kartris e-commerce site to be able to select a product option, and then have the main product images change to show that image. For example, if a customer is viewing a car, and we change the option for colour to ‘red’, we want the main image to change to a red car.

Kartris didn’t have this function, and so a new user control was required.

At the same time as providing this functionality we were also interested in implementing this image switching for combined options, where you could have multiple options such as colour and texture, and the image would show the combined image. For example, if looking at a skirt, you could selected ‘red’, and see a red skirt, and then select ‘floral’ and see a red floral skirt.

# Developers Guide

## Implementation

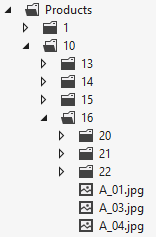
The actual image selection option group, known as the Swatch Selector could be implemented as a data list containing image controls (<img>) relatively easily, however, Kartris options are dynamically created in a control called OptionGroupContainer.ascx and this expects the option control to implement a list control interface. This means that we either had to extensively alter the OptionGroupContainer.ascx control with different methods of handling different option control types, or our swatch selector had to implement the list control interface.

As a result, the Swatch Selector contains a custom server control written for this application in addition to the alterations to the exiting files.

Because of the requirement to store images for each variation of each product that was going to use the Swatch Selector, we also had to come up with a solution for storing and naming the product images within the existing Kartris framework. In the end, we decided to make product option images a sub folder of the main product image folder.

For example, if you had product 10, you would have a folder [~/Images/Products/10/] which would contain all of the standard pictures for product 10, then within that you would have folders for each option, and within them the folder for each option combination. So product 10 could have option 16, and all images for that would be in [~/Images/Products/10/16] and it could also have option 20, and the folder for that would be [~/Images/Products/10/20] and then the folder for both option 16 and option 20 combined would be [~/Images/Products/10/16/20].

There would not be a folder [~/Images/Products/10/20/16] because nested options would only go in ascending order.



## Custom Server Control – ImageList

First step was to produce a server control that served image swatches in a list format, handled click events so that the customer could select the image swatches they desired, and exposed a list control interface.

The new server control inherits Web.UI.WebControls.ListControl and implements IRepeatInfoUser, INamingContainer, and IPostBackDataHandler as is required for data list controls.

<Assembly: TagPrefix("ImageList.Deadline.WebControls", "DalWebCtrl")>

Namespace Deadline.WebControls

<ToolboxData("<{0}:ImageList runat=server></{0}:ImageList>")> \_

Public Class ImageList

Inherits Web.UI.WebControls.ListControl

Implements IRepeatInfoUser, INamingContainer, IPostBackDataHandler

…

End Class

End Namespace

Within this control we implement all of the methods that are required as part of the interface. We also had to make some changes, particularly to the Render() event and the PostBackData events.

The sub CreateChildControls() creates the child control collection, and this is simple an unordered list containing image buttons. Each image button creates a post back event to this custom server control, and we retrieve the associated index number when we get the post back event.

Because we are using Image Buttons the post back event will send back a name of the control including the co-ordinates on the button that were pressed. This causes a problem during post back as ASP.Net does not recognise that our custom server control is the intended target for the post back event. For this reason we have to request that all post back events are passed through this control when it is present on the form.

Private Sub ImageList\_Init(sender As Object, e As EventArgs) Handles Me.Init

Page.RegisterRequiresPostBack(Me)

End Sub

For this application the overhead should not be a problem.

Within LoadPostData() we read in which button was pressed, and then determine what the related options ID was using the function GetDataForControl() which finds the value from a List(of ControlDataValue) which is a custom object for storing this data.

We have custom implementation of SaveViewState() and LoadViewState() which takes the normal viewstate and adds it to an array object. In that array object we also store the attributes for the image list that we have made because the attributes contain what the database ID (option ID) is for each image swatch. Normally, this is not persisted in ViewState.

## Modifying the Kartris Code

After adding references to the ImageList control from within Kartris we then started making the required changes to the existing modules. What follows is the applied changes in chronological order.

Kartris uses letters to determine what sort of option control you want to use in the Options Group Container user control. ‘L’ is for list item, ‘D’ is for dropdown etc. We wanted to use ‘S’ for Swatch.

In SQL Server we added ‘S’ to the CONSTRAINT CK\_OptionGroupsValidation.sql

Use the file called “ADD s to CONSTRAINT CK\_OptionGroupsValidation.sql” to achieve this.

This table explains what modules were changed within the Kartris source code and why each module was changed.

|  |  |
| --- | --- |
| Module | Change Reason |
| \_OptionGroups.ascx | This is where you create and edit option groups. Added ‘s’ as an option for the ImageSwatch option, and added the \_FileUploader.ascx control to the form so that users can upload the image swatches that will be shown in the option control |
| Kartris.vb | Added enumerator enum\_OptionSwatch and also added method to map that enumerator to the Option Groups images folder. |
| \_ItemSorter.ascx | Is now contained within a nested update panel within a data list as such, when you try to move images around, the asynchronous post back does not have the data values (option ID) recorded. Change made to persist image value into viewstate. |
| \_FileUploader.ascx | Persisted folder URLs to viewstate. Added method so that we could pass a sub folder string (e.g. “/12/34/56/78”) instead of just parent and item IDs. |
| Options.ascx | Multiple changes so that the ImageList control is loaded dynamically with the ListType is set to ‘s’ for Swatch.  Also added code to pass the folder location for the option images to the image list control. |
| OptionsContainer.ascx | Bubbling events from the Options.ascx control. |
| ProductVersions.ascx | Bubbling events from the OptionsContainer.ascx control. |
| ProductView.ascx | Change the images that are currently displayed in the ImageViewer.ascx control when the ProductVersions control raises events. Events from ProductVersions are ultimately bubbled up from the ImageList control. |
| Image.aspx | Modified querystring from a number to a string so that sub folder strings can be passed (e.g. “/12/34/56/78”) |
| ImageViewer.ascx | Moved Javascript events for applying presentation and events to the images into a named event InitImageClickEvents() . This way the event can be called when performing partial post backs. |
| \_ProductOptionGroups.ascx | Admin control that is present when editing products. This shows every possible combination of options that images can be applied to, and allows you to upload images in each case. |

What follows now is the change log that was produced while implementing all of the code changes. As you can see there are many changes that were required, it would not be practical to explain every change within this document.

Where you see ellipses (…) between the start and end of a method such as this:

<asp:PlaceHolder ID="phdNewSwatch" runat="server" Visible="true">

…

</asp:PlaceHolder>

This indicates that the entire section was added, removed, or altered as explained by the text.

## Change Log

\_OptionGroups.ascx line 92 ADD

<asp:ListItem Text="swatches" Value="s" />

\_OptionGroups.ascx line 167 ADD

<asp:ListItem Text="swatches" Value="s" />

SQL SERVER ran "ADD s to CONSTRAINT CK\_OptionGroupsValidation.sql" to add the character 's' to the list of acceptable option group characters.

\_OptionGroups.ascx.vb line 189 ADD

Case "s"

row("Display") = "Swatches"

\_OptionsGroups.ascx line 358 ADD

<asp:PlaceHolder ID="phdNewSwatch" runat="server" Visible="true">

…

</asp:PlaceHolder>

Kartris.vb line 1982 ADD

enum\_OptionSwatch = 6

Create Folder ~/Images/OptionGroups

Kartris.vb line 2009 ADD

Case IMAGE\_TYPE.enum\_OptionSwatch

strFolderURL &= "Images/OptionGroups/" & nItemID& "/"

\_ItemSorter.ascx.vb line 194 & 195 REM'd out and replacement made on line 196

'imgToRemove.ImageUrl = CType(e.Item.FindControl("litImgURL"), Literal).Text

'litImgName3.Text = imgToRemove.ImageUrl.ToString

litImgName3.Text = CType(e.Item.FindControl("litImgURL"), Literal).Text

\_OptionGroups.ascx.vb line 141 ADD

' show the 'Add Image' controls if the option type is 'image swatch'.

If e.Item.DataItem("OPTG\_OptionDisplayType") = "s" Then

CType(e.Item.FindControl("phdNewSwatch"), PlaceHolder).Visible = True

End If

FileUploader.ascx.vb line 188 ADD

Case IMAGE\_TYPE.enum\_OptionSwatch

c\_strFileName = "s" + CStr(c\_numItemID) + "\_"

FileUploader.ascx.vb line 110 ADD

Case IMAGE\_TYPE.enum\_OptionSwatch

Session("tab") = "images"

\_FileUploader.ascx.vb line 115. REM'd some code so that we now used a ViewState persisted control for the path.

' Changed so that w are trying to map to a folder which is persisted in a ViewState managed control.

'If Not Directory.Exists(Server.MapPath(c\_strUploadPath)) Then

' Directory.CreateDirectory(Server.MapPath(c\_strUploadPath))

'End If

If Not Directory.Exists(Server.MapPath(litInfo.Text)) Then

Directory.CreateDirectory(Server.MapPath(litInfo.Text))

End If

\_FileUploader.ascx.vb line 222 REM'd some code so that we now used a ViewState persisted control for the path.

' Changed so that we are trying to map to a folder which is persisted in a ViewState managed control.

'existingFiles = Directory.GetFiles(Server.MapPath(c\_strUploadPath))

existingFiles = Directory.GetFiles(Server.MapPath(litInfo.Text))

\_FileUploader.ascx.vb line 237 REM'd some code so that we now used a ViewState persisted control for the path.

' Changed so that we are trying to map to a folder which is persisted in a ViewState managed control.

'If Not File.Exists(Server.MapPath(c\_strUploadPath & strTempName)) Then

If Not File.Exists(Server.MapPath(litInfo.Text & strTempName)) Then

\_UC\_UploaderPopup.SaveFile(Server.MapPath(litInfo.Text & strTempName), I, I < (FileNames.Count - 1))

\_FileUploader.ascx.vb line 242 REM'd some code so that we now used a ViewState persisted control for the path.

' Changed so that we are trying to map to a folder which is persisted in a ViewState managed control.

'If IsNumeric(strCompressQuality) AndAlso strCompressQuality > 0 AndAlso strCompressQuality < 100 Then CompressImage(Server.MapPath(c\_strUploadPath & strTempName), CLng(strCompressQuality))

If IsNumeric(strCompressQuality) AndAlso strCompressQuality > 0 AndAlso strCompressQuality < 100 Then CompressImage(Server.MapPath(litInfo.Text & strTempName), CLng(strCompressQuality))

\_ItemSorter.ascx.vb line 221 MODIFY

' Simplification of the code by removing uneccessary control.

strFileName = litImgName3.Text ' imgToRemove.ImageUrl

\_FileUploader.ascx.vb Changed Public Property ItemID() to use ViewState rather than a local variable. Changed all references to c\_numItemID in code and pointed to ItemID() instead.

Added reference to ImageList.dll

Options.ascx line 2 ADDED

<%@ Register Assembly="ImageList" Namespace="ImageList.Deadline.WebControls" TagPrefix="DalWebCtrl" %>

Options.ascx.vb line 16 ADDED

Imports DlaWebCtrls = ImageList.Deadline.WebControls

Options.ascx.vb line 90 ADDED

Case "s"

\_OptionType = "Swatch"

\_Control = New DlaWebCtrls.ImageList

\_ControlPrice = New CheckBoxList

AddHandler CType(\_Control, DlaWebCtrls.ImageList).SelectedIndexChanged, AddressOf indxChanged

CType(\_Control, DlaWebCtrls.ImageList).CssClass = "imagelist"

Options.ascx.vb line 176 ADD

If \_Control.GetType = GetType(DlaWebCtrls.ImageList) Then

' The control is an image list.

Dim FilePaths As String() = Directory.GetFiles(Server.MapPath(CkartrisImages.CreateFolderURL(CkartrisImages.IMAGE\_TYPE.enum\_OptionSwatch, pItemValue)))

If FilePaths.Count > 0 Then

itemListOptions.Attributes.Add("ImageUrl", CkartrisImages.CreateFolderURL(CkartrisImages.IMAGE\_TYPE.enum\_OptionSwatch, pItemValue) & Path.GetFileName(FilePaths(0)))

End If

End If

Options.ascx.vb line 336 ADD

Case "Swatch"

If \_Control.SelectedIndex < 0 Then

Return False

End If

Options.ascx.vb line 217 ADD

Public ReadOnly Property GetControlType As String

…

End Property

OptionsContainer.ascx.vb line 285 ADD

If CType(sender, Options).GetControlType = "Swatch" Then

RaiseEvent Event\_OptionSwatchChanged(CType(sender, Options).GetOptionGroupID, CType(sender, Options).GetSelectedValues, CType(sender, Options).GetSelectedText)

End If

OptionsContainer.ascx.vb line 336 ADD

Public Event Event\_OptionSwatchChanged(ByVal OptionParentID As Integer, OptionId As Integer, OptionText As Integer)

ProductVersions.ascx.vb line 796 ADDED

Protected Sub OptionSwatchChanged()

…

End Sub

ProductVersions.ascx.vb line 801 ADDED

Public Event OptionsSwatchChanged(ByVal OptionParentID As Integer, OptionId As Integer, OptionText As Integer)

ProductView.ascx.vb line 186 ADDED

Private Sub OptionSwatchChanged()

…

End Sub

Image.aspx line 46 MODIFIED

' Modified from number to string by Craig Moore 31-10/2016 so that option groups within the images folder could be

' defined and loaded.

Dim strItem As String = ""

Try

strItem = Request.QueryString("numItem") 'ID number of product/category/version.

Catch

End Try

ProductView.ascx.vb line 32 ADDED

Private \_ProductOptionID As Integer

Public ReadOnly Property ProductOptionId As Integer

Get

Return \_ProductOptionID

End Get

End Property

ProductView.ascx.vb line 146 MODIFY

Dim strProductFolder As String = String.Empty ' Folder where images are located.

strProductFolder = CStr(ProductID)

If \_ProductOptionID > 0 Then

' If a product option (swatch) has been selected in an option control, we need to append its details to the

' image folder that the image control is going to go looking for the new images.

' E.g. Product 5 with swatch option 11 would be looking in /Images/Products/5/11.

strProductFolder = strProductFolder & "/" & CStr(ProductOptionId)

End If

ProductView.ascx.vb line 150 MODIFY

strProductFolder, \_

ProductView.ascx.vb line 175MODIFY

strProductFolder,

ImageViewer.ascx.vb line 75 ADD

litMainImage.Text = ""

Below are a couple of related changed made in one go.

Within ProductView.ascx.vb. Removed the image loading functionality from LoadProduct() and put it into its own function BuildImageViewer(), then within ProductView.ascx.vb LoadProduct Line 139 we add a call to this method.

ProductView.ascx.vb Line 139 ADDED

BuildImageViewer()

ProductView.ascx.vb OptionSwatchChanged() added two new calls to update the image display. line 219 ADDED

' refresh the image display

BuildImageViewer()

updImages.Update()

ImageViewer.ascx

Placed all inline Javascript (line 31) inside a wrapper method InitImageClickEvents()

ImageViewer.ascx.vb Line 378 ADDED

Protected Sub Page\_PreRender(sender As Object, e As EventArgs) Handles Me.PreRender

' Register the click events on startup, required because partial page postbacks from selecting image swatches do

' not trigger the script on this page that sets up the click events otherwise.

ScriptManager.RegisterStartupScript(Page, Me.GetType(), "InitClickEvents", "InitImageClickEvents();", True)

End Sub

ProductView.ascx.vb line 227 ADDED

' trigger the Foundation script to run again if we are performing a partial page postback with updated images.

' This is because the foundation script only runs on page load, not on partial postback, and we need it to run to format the carousel for

' the images.

' The script is here, and not in the image viewer control because the image viewer control gets redrawn in the code behind even if it is not

' being posted to the client, and it is a waste of resources to trigger the script every postback. It is only from within this current method

' that we know why the redraw is occuring and we know that we are updating the image update panel.

ScriptManager.RegisterStartupScript(Page, Me.GetType(), "FoundationClearing", "$(document).foundation();", True)

Deleted \_ProductOptionId from productview.ascx.vb as it is now not needed

Deleted its reference from OptionSwatchChanged() too

Added ProductView.RecordOptionSelection()

Added ProductView.SwatchOptions()

ProductView.BuildImagePath() refractored out and referenced at line 163.

Added tabOptionImages to \_ProductOptionGroups.ascx

\_ProductOptionGroups.ascx.vb line 180 ADDED

Private Sub GetOptionImages()

…

End Sub

\_ProductOptionGroups.ascx.vb line 238ADDED

GetOptionImages()

\_ProductOptionGroups.ascx.vb line 981 ADDED

Protected Sub dlOptionImages\_ItemCommand()

…

End Sub

\_ProductOptionGroups.ascx.vb line 181 ADDED

GenerateOptionImageCombinations()

\_ProductOptionGroups.ascx

Added reference to \_FileUploader at top of control

\_FileUploader.ascx.vb Line 98 ADDED

Public Property ImagePath() As String

…

End Property

\_FileUploader.ascx.vb Line 113 MODIFIED

If ImagePath = String.Empty Then

c\_strUploadPath = CreateFolderURL(c\_enumImageType, CStr(ItemID), CStr(ParentID))

Else

c\_strUploadPath = CreateFolderURL(c\_enumImageType, CStr(ItemID)) & ImagePath

If c\_strUploadPath.Length > 0 Then

If Not Right(c\_strUploadPath, 1) = "/" Then

c\_strUploadPath = c\_strUploadPath & "/"

End If

End If

End If

\_ItemSorter.ascx.vb

Changed all direct calls to c\_FolderPath to the property FolderPath - Fuck knows why this wasn't done before.

\_ItemSorter.ascx.vb Line 27 MODIFIED to cache value into Viewstate

Public Property FolderPath()

…

End Property

\_ItemSorter.ascx.vb Line 26 MODIFIED to add default value and comments.

Dim c\_FolderPath As String = String.Empty

\_ProductOptionGroups.ascx.vb Line 1036 ADDED

Protected Sub dlOptionImages\_ItemDataBound

…

End Sub

\_ProductOptionGroups.ascx.vb Line 501 ADDED

GetOptionImages() ' Update the option image display

\_OptionsGroups.aspx.vb line 26 ADDED

' Struggled to get this set from the file uploader control, no idea why.

' As a result, I have added this part here so that the page works on first postback of the upload control.

Page.Form.Enctype = "multipart/form-data"

OptionsGroups.ascx.vb Line 64 ADDED

If Page.Form.Enctype <> "multipart/form-data" Then

' In this form there is a repeater with a file uploader. Becuase of the lifecycle of this page, the file upload control

' cannot set the form encoding type at the correct time. For this reason we set the encoding type to Multipart at this point.

' which is the required encoding type for file uploads.

Page.Form.Enctype = "multipart/form-data"

End If