# FGThumbnailCanvas

**Version 1.0**

# About the class

FGThumbnailCanvas is a REALbasic drop-in control that is intended to provide the functionality of the thumbnail canvas found in many Apple Macintosh applications such as iPhoto and Aperture. It is extremely flexible out-of-the box but is also customisable if you have a unique look you are trying to achieve.

When might I use it? FGThumbnailCanvas provides a gorgeous-looking user interface for browsing through anything that can be represented as an image. Common applications include photos, album covers, video stills and product pictures.

FGThumbnailCanvas will compile to all three platforms supported by REALbasic (Mac, Windows and Linux). It has been thoroughly tested on Mac (both Cocoa and Carbon) and Windows. It's written entirely in native REALbasic code and requires **no** additional plugins to function.

The FGThumbnailCanvas control comprises five components, only two of which you are likely to need to interact with directly - the canvas itself (FGThumbnailCanvas) and the base class for the thumbnails displayed on that canvas (FGThumbnailClass). FGThumbnailCanvasSuper is the parent class of FGThumbnailCanvas (it handles drawing the drag selection rectangle) and BoundsRectangle is a helper class for the canvas to determine collision detection. A single class Interface is also included (FGThumbnail) that **must** be implemented by all thumbnail classed and subclasses.

A little note about using images

When assigning images to your thumbnails, try to bear in mind that when REALbasic loads an image into RAM for manipulation as a Picture object that it's memory footprint will be **much** greater than the size of the image on disk. Apple knows this and this is why if you look inside your iPhoto or Aperture library you'll notice a 'Thumbnails' folder. In this folder there are low-resolution or down-scaled versions of the actual image that you want to display to the user. This is good practice and is highly recommended. The speed improvements can be enormous by using a smaller (say 400 x 400 pixel JPEG) image for your thumbnails than the original several megapixel image.

# Adding the class to your projects

To use FGThumbnailCanvas in a REALbasic project, you must add all of its required components to the 'Project' tab in the REALbasic IDE.  The easiest way to do this is to copy the folder entitled 'FGThumbnailCanvas Components' (found in your download) into the same directory as your project and then drag it from there onto the 'Project' tab in the REALbasic IDE.  This will add the required classes and interfaces to your project.

To add a FGThumbnailCanvas to your project simply drag an instance of one onto a window of your choosing in your project.  You can find the FGThumbnailCanvas control in the window editor by clicking on the 'Project Controls' popup menu on the left-hand side of the editor.

The default settings in the properties inspector for the FGThumbnailCanvas approximate the appearance of the iPhoto thumbnail canvas and will probably suffice for most users but feel free to change them.

If you have any problems regarding the use of the FGThumbnailCanvas control or would like to make suggestions or file a bug report then please either email us at support@madebyfiga.com or visit our support forums at www.madebyfiga.com/forums.

# The structure of the thumbnail canvas

FGThumbnailCanvas manages an array of FGThumbnailClass subclasses.  FGThumbnailClass is the base class from which all of your custom thumbnail classes should inherit from. FGThumbnailClass implements some basic (but critical) functionality that all thumbnails need (such as collision detection) so you don't have to worry about it.

Before you can add thumbnails to your newly instantiated FGThumbnailCanvas control (using the *AddThumbnail* method) you must define your own FGThumbnailClass subclass for use in your application.  Why? Why can't I just use FGThumbnailClass?  Well, mainly this is to provide you with maximum power and flexibility on how each thumbnail is presented.  You must define a subclass and implement the methods specified in the FGThumbnail Interface.  If you're not familiar with REALbasic's powerful Interface capabilities then fear not - they are very simple.  When a class implements an interface, it simply "promises" to implement the methods specified in that interface. By implementing the FGThumbnail interface in your FGThumbnailClass subclasses, you are providing your own implementation of the methods that FGThumbnailCanvas needs each thumbnail to do.

Getting started subclassing FGThumbnailClass

Create a new class in the IDE and set it's Super to FGThumbnailClass and tell it to implement the FGThumbnail interface (this is done via the property inspector on the right-hand side of the 'Project' tab in the IDE).  Name your class.

Double-click this new class and add the four FGThumbnail interface methods listed below:

| Image( Selected as Boolean = False ) As Picture |
|---|
| Update(CellSize as Integer, ForceUpdate as Boolean = False) |
| Visible() as Boolean |
| Visible(Assigns Value as Boolean) |

How you implement these methods is up to you.  These methods are called at predictable points by FGThumbnailCanvas:

Image() is called whenever FGThumbnailCanvas is going to draw that thumbnail to the screen. It's up to you to return a REALbasic Picture object to the canvas of the correct size.

Update() is called whenever FGThumbnailCanvas thinks that this thumbnail should update itself. This might be because the thumbnail's available size has changed (the new size is CellSize x CellSize) or because the user has forced a redraw.

The Visible() methods are called by FGThumbnailCanvas when this thumbnail's visibility state has changed.  One use for this method would be to delete any stored images for this thumbnail in RAM to free up memory when the thumbnail is no longer visible.

The demo project contains an example subclass called PhotoThumbnail.  If you carefully examine this class you should get a clearer idea of the above concepts.

# FGThumbnailCanvas

# FGThumbnailCanvas Events

Although the FGThumbnailCanvas inherits from the built-in Canvas class (actually it inherits from the FGThumbnailCanvasSuper class which itself inherits from the Canvas class), the following events have been **removed**:
- ConstructContextualMenu
- ContextualMenuAction
- MouseDown
- MouseUp

Some events are unique (or are modified from REALbasic's canvas):
- ClickedThumbnail
- MaxScrollValueChanged
- MouseDrag
- ScrollValueChanged
- SelectionChanged
- ThumbnailCountChanged
- Update

## ClickedThumbnail

Whenever a thumbnail is clicked by the user in the canvas (even if multiple thumbnails are selected), this event is fired.

> ClickedThumbnail(Thumbnail as FGThumbnailClass, ContextualClick as Boolean = False, X as Integer = -1, Y as Integer = -1) As Boolean

Parameters
*Thumbnail*
A pointer to the actual thumbnail object clicked. Note that this is an instance of FGThumbnailClass (that your thumbnails will inherit from). If you want to access properties on this object that are part of your custom class you will need to *cast* this object to your subclass.

*ContextualClick*
Set to true if the user contextual-clicked the thumbnail or false if they left-clicked it.

*X, Y*
The global X, Y coordinates of the click. The main purpose of these properties is to let you know where to display a contextual menu (if desired).

Notes
If you return true then FGThumbnailCanvas will not draw a drag selection rectangle - returning false means it will. Generally, if *Thumbnail* is not nil, you should return true to permit drag and drop functionality.

**MaxScrollValueChanged**

Called whenever the size of the FGThumbnailCanvas itself has changed or the individual thumbnail size is altered.  It's purpose is to allow you to update a scrollbar with new parameters.

MaxScrollValueChanged(NewMaximum as Integer, NewPageStep)

Parameters
*NewMaximum*
The new maximum value for the scrollbar

*NewPageStep*
The new page step value for the scrollbar

**MouseDrag**

The user is dragging the mouse on the canvas.

MouseDrag(X as Integer, Y as Integer, Thumbnails() as FGThumbnailClass)

Parameters
*X, Y*
The mouse has been dragged to the local coordinates X, Y.

*Thumbnails()*
An array containing the FGThumbnailClass subclass objects currently being dragged by the user. Remember to *cast* these objects to your subclass.

**ScrollValueChanged**

Called whenever the FGThumbnailCanvas is scrolled or the thumbnail cell size is altered.  It provides you with the opportunity to update an optional scrollbar you may have linked to the FGThumbnailCanvas.

ScrollValueChanged(NewValue as Integer)

Parameters
*NewValue*
The new value of the scroll bar.

**SelectionChanged**

The currently selected thumbnails in the FGThumbnailCanvas have changed.

SelectionChanged()

**ThumbnailCountChanged**

The number of thumbnails being managed by FGThumbnailCanvas has changed.

ThumbnailCountChanged(NewCount as Integer)

Parameters
*NewCount*
The new number of thumbnails being managed.

# FGThumbnailCanvas Methods

## AddThumbnail

Adds the passed object to the canvas' array of managed thumbnails.

```
AddThumbnail(Thumbnail as FGThumbnailClass)
```

Parameters
*Thumbnail*
The thumbnail object to manage.

Notes
Remember that *Thumbnail* should be your own subclass of FGThumbnailClass.

## Clear

Removes all managed thumbnails from the canvas and redraws.

```
Clear()
```

## Clear(Integer)

Removes the managed thumbnail at the specified index in FGThumbnailCanvas' managed array and redraws.

```
Clear(Index as Integer)
```

Parameters
*Index*
The (zero-based) index of the thumbnail to remove.

## ClearSelection

Removes from FGThumbnailCanvas' managed array all thumbnail objects that are currently selected.

```
ClearSelection()
```

## DeselectAll

Deselects any thumbnail that is currently selected.

```
DeselectAll()
```

## Scroll

Scrolls the canvas by the specified number of rows.

```
Scroll(RowsToScroll as Integer)
```

Parameters
*RowsToScroll*
The number of rows to scroll.  If positive we scroll down, if negative we scroll up.

## ScrollDown

Scrolls the canvas down the specified number of rows.

```
ScrollDown(RowsToScroll as Integer = 1)
```

Parameters
*RowsToScroll*
The number of rows to scroll.  Defaults to one.

## ScrollToRow

Scrolls the canvas to the specified row number.

```
ScrollToRow(RowToScrollTo as Integer)
```

Parameters
*RowToScrollTo*
The row number to scroll to.  The first row is zero.

## ScrollUp

Scrolls the canvas up the specified number of rows.

```
ScrollUp(RowsToScroll as Integer = 1)
```

Parameters
*RowsToScroll*
The number of rows to scroll.  Defaults to one.

## SelectAll

Select all thumbnails in the canvas.

```
SelectAll()
```

**SelectedItemIndices**

Returns a (sorted) array containing the indices of the thumbnail objects currently selected.

```
SelectedItemIndices() as Integer()
```

**SelectedThumbnail**

Returns the currently selected thumbnail object.

```
SelectedThumbnail() as FGThumbnailClass
```

**SelectedThumbnails**

Returns an array of the currently selected thumbnail objects.

```
SelectedThumbnails() as FGThumbnailClass()
```

**Update**

Tells the canvas to redraw.

```
Update(ForceUpdate as Boolean = False)
```

Parameters
*ForceUpdate*
Tells every visible thumbnail that the user wants them to update, even if they don't deem it necessary.

# FGThumbnailCanvas Properties

**AllowMouseWheelScrolling**

```
AllowMouseWheelScrolling as Boolean
```

*Read-write*.  If true then the mouse wheel will cause scrolling of the canvas.

**BackgroundColor**

```
BackgroundColor as Color
```

*Read-write*. The colour of the canvas' background.

**CellSize**

```
CellSize as Integer
```

*Read-writ*e.  The current size (in pixels) of each thumbnail cell.

**CurrentThumbnailUnderMouseIndex**

```
CurrentThumbnailUnderMouseIndex as Integer
```

*Read-only*.  The (zero-based) index of the current thumbnail under the mouse.  Returns -1 if nil.

**DownKeyCode**

```
DownKeyCode as Integer
```

*Read-write*.  The keyboard keycode for the key to be used to scroll down through the thumbnails.

**DragSelectionBorder**

```
DragSelectionBorder as Integer
```

*Read-write*.  The thickness (in pixels) of the border of the selection box that appears when drag-selecting.

**DragSelectionBorderColor**

```
DragSelectionBorderColor as Color
```

*Read-write*.  The colour of the border of the selection box that appears when drag-selecting.

### DragSelectionColor

```
DragSelectionColor as Color
```

*Read-write. Mac-only.*  The fill colour of the selection box that appears when drag-selecting.

### DragSelectionOpacity

```
DragSelectionOpacity as Integer
```

*Read-write. Mac-only.*  The opacity of the fill colour of the selection box that appears when drag-selecting.

### DrawSelectionRectangle

```
DrawSelectionRectangle as Boolean
```

*Read-write.*  If true then selected thumbnails in the canvas will have a rounded rectangle drawn around them.

### ForceUpdateOnMouseMove

```
ForceUpdateOnMouseMove as Boolean = True
```

*Read-write.*  If true then the canvas will redraw whenever the mouse moves over it.  This is useful if you would like to supply a different thumbnail image if the mouse is hovering over a thumbnail. Defaults to true.

### InFocus

```
InFocus as Boolean
```

*Read-only.*  Returns true if the canvas has the focus, false if it doesn't.

### InvertMouseWheelScrollDirection

```
InvertMouseWheelScrollDirection as Boolean = True
```

*Read-write.*  Inverts the direction of scrolling when using the mouse wheel.  Defaults to true.

Notes
Only takes effect if AllowMouseWheelScrolling = True.

### LeftKeyCode

```
LeftKeyCode as Integer
```

*Read-write.*  The keyboard keycode for the key to be used to scroll left through the thumbnails.

### MaxCellSize

```
MaxCellSize as Integer
```

*Read-write*.  The maximum permissible size (in pixels) of each thumbnail.

### MaxColumnPadding

```
MaxColumnPadding as Integer
```

*Read-write*. The maximum number of pixels that should pad each column.

### MaxRowPadding

```
MaxRowPadding as Integer
```

*Read-write*.  The maximum number of pixels that should pad each row.

### MinCellSize

```
MinCellSize as Integer
```

*Read-write*.  The minimum permissible size (in pixels) of each thumbnail.*MinColumnPadding*

```
MinColumnPadding as Integer
```

*Read-write*.  The minimum number of pixels that should pad each column.

### MinRowPadding

```
MinRowPadding as Integer
```

*Read-write*.  The minimum number of pixels that should pad each row.

### RightKeyCode

```
RightKeyCode as Integer
```

*Read-write*.  The keyboard keycode for the key to be used to scroll right through the thumbnails.

## SelectedBorderColor

SelectedBorderColor as Color

*Read-write*.  The colour of the border of the selection box that surrounds selected thumbnails.

Notes
The thumbnail selection box is only drawn if DrawSelectionRectangle = True.

## SelectedBorderThickness

SelectedBorderThickness as Integer

*Read-write*.  The thickness (in pixels) of the selection box that surrounds selected thumbnails.

Notes
The thumbnail selection box is only drawn if DrawSelectionRectangle = True.

## SelectionCount

SelectionCount as Integer

*Read-only*.  The number of thumbnails currently selected in the canvas.

## Selecting

Selecting as Boolean

*Read-only*. Set to true if the user is currently drag-selecting, false if they are not.

## UpKeyCode

UpKeyCode as Integer

*Read-write*.  The keyboard keycode for the key to be used to scroll up through the thumbnails.

# FGThumbnail Interface

# FGThumbnail Interface Methods

All of the following methods must be implemented in your FGThumbnailClass subclasses.

## Image

> Image(Selected as Boolean = False) As Picture

Parameters
*Selected*
This thumbnail is currently selected in the canvas.

*Return value*
The REALbasic picture object representing this thumbnail for its current size.

## Update

> Update(CellSize as Integer, ForceUpdate as Boolean = False)

Parameters
*CellSize*
The new size of the thumbnail in the canvas.

*ForceUpdate*
If true then the canvas is informing this thumbnail that it should update regardless of whether or not it deems it necessary.

Notes
It is recommended that in this method, you create the picture that you will subsequently return in the Image() method.  It's probably a good idea to store the CellSize in a local property as well for later reference.

## Visible

> Visible() As Boolean

Parameters
*Return value*
You should return whether this thumbnail is visible or not.

Notes
It is **highly** recommended that in Visible(Assigns Value as Boolean) you store locally the visible state of the thumbnail and return that value in this method.

## Visible(Boolean)

Visible(Assigns Value as Boolean)

<u>Parameters</u>
*Value*
Set to true if FGThumbnailCanvas has determined that this thumbnail is currently visible, false if it's not.

# FGThumbnailClass

# FGThumbnailClass Methods

## Bounds

Returns this thumbnail's BoundsRectangle.

---
Bounds() As BoundsRectangle

---

## Bounds(Integer, Integer, Integer, Integer)

Sets the current bounds of this thumbnail.

---
Bounds(X as Integer, Y as integer, W as Integer, H as Integer)

---

Parameters
*X, Y*
The top-left coordinates of this thumbnail.

*W*
The width of this thumbnail's bounds.

*H*
The height of this thumbnail's bounds.

## Hit

Returns whether or not the passed coordinates fall within this thumbnail's bounds.

---
Hit(X as Integer, Y as Integer) As Boolean

---

Parameters
*X, Y*
The coordinates to test.

*Return value*
Returns true if (x, y) lies within our bounds or false if it doesn't.

## Inside

Returns whether or not this thumbnail is within the passed BoundsRectangle.

---
Inside(TestRectangle as BoundsRectangle) As Boolean

---

Parameters
*TestRectangle*
The BoundsRectangle that we want to check to see if we are within.

*Return value*
Returns true if this thumbnail is wholly contained within *TestRectangle* and false if it is not.

**Intersects**

Returns whether or not this thumbnail intersects any part of the passed BoundsRectangle.

Intersects(TestRectangle as BoundsRectangle) As Boolean

Parameters
*TestRectangle*
The BoundsRectangle that we want to check to see if we are within.

*Return Value*
Returns true if any part of this thumbnail intersects with *TestRectangle* and false if it does not.

# FGThumbnailClass Properties

**Selected**

| Selected as Boolean = False |
| --- |

*Read-write.*  Set by FGThumbnailCanvas to true if this thumbnail is currently selected and false if it is not.  Defaults to False.

**UnderMouse**

| UnderMouse as Boolean = False |
| --- |

*Read-write.* Set by FGThumbnailCanvas to true if this thumbnail is currently underneath the mouse and false if it is not.  Defaults to false.

# <u>Release Notes</u>

<u>Version 1.0 (15/07/11)</u>

First public release