

## Multiple-Document Application Template for REALbasic 2006r3

REALbasic makes great Single-Window Applications right out of the box, but when it comes to making a Multiple-Document Application, there's a lot of work to be done. You've got to account for saving, creating, reverting, opening, and changes, among other things. Then you have all the other perks like printing and preferences that should be in every application. That's quite a bit to implement.

We've taken care of all that for you - even added a standard "window" menu and "open recent" menu. You barely need to implement anything!

The project includes many of our smaller projects, such as our TransportableItems classes and Preferences module. The Preferences module requires the TransportableItems classes. And these classes allow you to turn many RB datatypes into transportable XML strings. See the TransportableItems Read Me for more information on these wonderful classes. You'll probably want to use them to save and load your documents.

Also, please read this entire document before you dive into development. There's a few things that need to be covered, or you'll have problems.

### App Object

Your standard "App" object is now a MultipleDocumentApplication instance, which implements a few new Events which you'll want to become familiar with:

#### **GetApplicationIdentifier() As String**

This event NEEDS to return a string in order for Preferences to work properly. An application identifier on Mac OS X typically consists of the organization type, organization name, and product name. The organization type should be either "com" or "org" - for commercial or non-profit, respectively. The other two are entirely up to you, but should be lowercase and may only use a-z and 0-9 characters. Separate each by periods, and you get:

```
com.apple.mail  
com.thezaz.networkclipboard  
com.panic.transmit
```

There's no reason to change the identifier for each platform, but you're welcome to add compiler directives if you feel necessary.

#### **NewDocument(appLaunching As Boolean)**

This is the basic NewDocument event. However, it's called every time the File->New menu item is selected. Therefore, the appLaunching property is set to true only if the event is being called during launch time.

**OpenDocument(Item As FolderItem, appLaunching As Boolean)**

Same notes as the NewDocument event. Item is a valid FolderItem to be opened. We've already taken care of adding the item to the Recent Items menu. This event is only called when a file actually needs to be opened. For example, if a user attempts to open a file that is already open, the event will NOT fire, but the window will be shown instead.

The easiest implementation consists of creating a new window and passing it the "Item" object via the "RestoreItem" method:

```
dim w as window1
w = new window1
w.RestoreItem Item
```

(Window1 is a DocumentWindow, described later)

**SetupOpenDialog(Dialog As OpenDialog)**

This event is called when the File->Open menu item is selected. Dialog is a non-nil OpenDialog object. Simply set properties, like Filter and PromptText. DO NOT RUN DIALOG.

**ShowPreferences()**

This event is fired when the Preferences option is selected from the Application menu (or Settings option in the Edit menu on Windows and Linux) Do what you want with this event.

App also has some new Methods, which you may or may not want to use:

**ApplicationIdentifier() As String**

This method simply triggers the GetApplicationIdentifier event and returns the value. Useful in writing Mac OS X applications

**AskOpenFile()**

Shows an OpenDialog asking the user to open a file. Exact same functionality as the File->Open menu item.

**AttemptQuit()**

Instead of using the built-in Quit function, you are advised to use the AttemptQuit method. If there are documents with unsaved changes, a "Review Changes" dialog will be shown giving the user a chance to cancel the quit. If there is no need for the dialog, or the user dismisses it, the application will be quit.

**CreateDocument()**

Simply triggers the NewDocument() event, so you may implement your own "New" option in a toolbar or wherever.

### **RestoreDocument(Item As FolderItem)**

Will attempt to open the file specified. If it is already open, the window will be shown. If not, the OpenDocument event will be fired with Item.

## **DocumentWindow**

That's everything you need to know about your upgraded App object. Now we need to talk about the DocumentWindow class. Every window you create that should represent a document of some kind, needs to be an instance of this class. By default, Window1 is a DocumentWindow. There are a few Events and Methods you need to become familiar with.

### **Events**

#### **DocumentTypeString() As String**

Simply for cosmetics, DocumentTypeString allows you to return a string to use instead of "Document" when asked to save, or in untitled document windows. By default, a new window will be called "Untitled Document". But if you tend to use the word "Project" instead, for example, you can return "Project" in this event. You will see "Untitled Project" from now on.

#### **PrintFile(g as graphics)**

Just like the paint event. Draw whatever you want into g, and when the method is completed, it'll be printed.

#### **PrintSetup() As Boolean**

Fired before the "Print" dialog is displayed. It'll allow you to display other messages or dialogs to gather additional information. Return true to cancel the print process.

#### **RestoreFile(Item As FolderItem)**

Item is a non-nil FolderItem. The application already knows the file needs to be opened. In this event, you should open the file and set the window's contents accordingly.

#### **SaveFile(Item As FolderItem)**

Item is a non-nil (but possibly nonexistent) FolderItem. The dialog has already been displayed and the application knows the document needs to be saved here. The contents of this event should store the document to a item.

#### **SetupSaveDialog(Dialog As SaveAsDialog)**

Just like SetupOpenDialog in the App object, Dialog is a non-nil SaveAsDialog that should be populated. DO NOT RUN DIALOG

### **Methods**

#### **AskSaveAs() As Boolean**

Displays a standard "Save As..." dialog. SetupSaveDialog and SaveFile will be fired as usual. Returns true on a successful save, false on failure.

### **DocumentName() As String**

Gets the name of the document. In the case of an unsaved file, it'll return "Untitled Document" or the filename in the case of a saved file.

### **Print()**

Starts the print process.

### **RestoreItem(Item As FolderItem)**

Fires the RestoreFile event. Thus, opens item.

### **Save() As Boolean**

Just like AskSaveAs(), saves the file. If the document has never been saved, a "Save As" dialog will be shown. If it has, the file is simply saved. Returns true on success.

### **SetTitle()**

Call to refresh the title of the window.

## **Properties**

### **CanBeModified As Boolean**

Defaults to true. Set to false to prevent the Modified property from being set. If Modified is true and CanBeModified becomes set to false, Modified will also become false.

### **CanBePrinted As Boolean**

Defaults to false. Some documents don't get printed. Set to true to enable printing.

### **File As FolderItem**

Defaults to Nil. This is the file of the document.

### **Modified As Boolean**

Defaults to false. Every time a change is made to the document, set Modified to true. You should never need to set it to false, that is taken care of for you. On Mac OS X, it displays the dark circle over the close box. If Modified is not set, the application will not ask about saving changes, nor will it properly allow saving of opened documents.

DocumentWindow windows will also automatically get a draggable proxy icon alias in their title bar on Mac OS X.

## **Preferences**

There's not much to worry about in regards to preferences. They are loaded and

saved automatically, so you only need to worry about using them - which is really simple. It would be a wise idea to read the TransportableItems Read Me before you continue.

Every Method and Property is protected. So you'll need to call them using Preferences.Save - for example. The idea is to treat Preferences like an object.

Strings, Integers, Doubles, Colors, Booleans, and FolderItems can be saved in the Preferences without any conversion at all. Even better, FolderItems properly use GetSaveInfo and NOT AbsolutePath. When requesting data, an optional second parameter can be passed which will be returned if the key does not exist.

Saving arrays and dictionaries are also possible, but require a little more work. That will not be fully covered in this document, but if you read the TransportableItems Read Me, you'll figure it out quickly.

## Methods

### **Preferences.Save()**

Simply save the preferences. You should never need to call this. It is done for you.

### **Preferences.Load()**

Loads the preferences. Again, should never need to be called.

## Properties

### **Preferences.BooleanValue(Key As String, Default As Boolean = False) As Boolean**

Get or set value for Key as a boolean:

```
if not Preferences.BooleanValue("Has Launched",false) then
  msgbox "Welcome!"
end
Preferences.BooleanValue("Has Launched") = true
```

### **Preferences.ColorValue(Key As String, Default As Color = &c000000) As Color**

Get or set value for Key as a color:

```
self.backcolor = Preferences.ColorValue("Document Background Color")
```

You get the idea, so we'll just list the rest

### **Preferences.DoubleValue(Key As String, Default As Double = 0) As Double**

### **Preferences.FolderItemValue(Key As String, Default As FolderItem = Nil) As FolderItem**

### **Preferences.IntegerValue(Key As String, Default As Integer**

```
= 0) As Integer  
Preferences.StringValue(Key As String, Default As String =  
"") As String
```

Oh, and one last property:

```
Preferences.ObjectValue(Key As String) As  
TransportableItem
```

This property allows you to get or set any of the TransportableItem classes to the Preferences. Because there are TransportableArray and TransportableDictionary classes available, you would use this property to save arrays or dictionaries in your Preferences. Caution: if the key does not exist, nil will be returned. Do not attempt to use without first consulting the TransportableItems Read Me.

From here on out, you should be all set to write yourself a beautiful multiple-document application. To make your life really easy, move this file to the "Project Templates" folder inside your REALbasic application folder. This way, whenever you start a new project, you have the Multiple-Document Application as an option!

I'd like to thank both Kevin Ballard and Seth Willits. Kevin Ballard's Carbon Declare Library helped with the proxy icons in Mac OS X, and Seth Willits wrote the Window Menu code.

Please do not distribute this file at all. If you would like to send it to a friend, please link them to the original site, as a new version may have been released.

```
<thom@thezaz.com>  
<http://www.thezaz.com/>
```