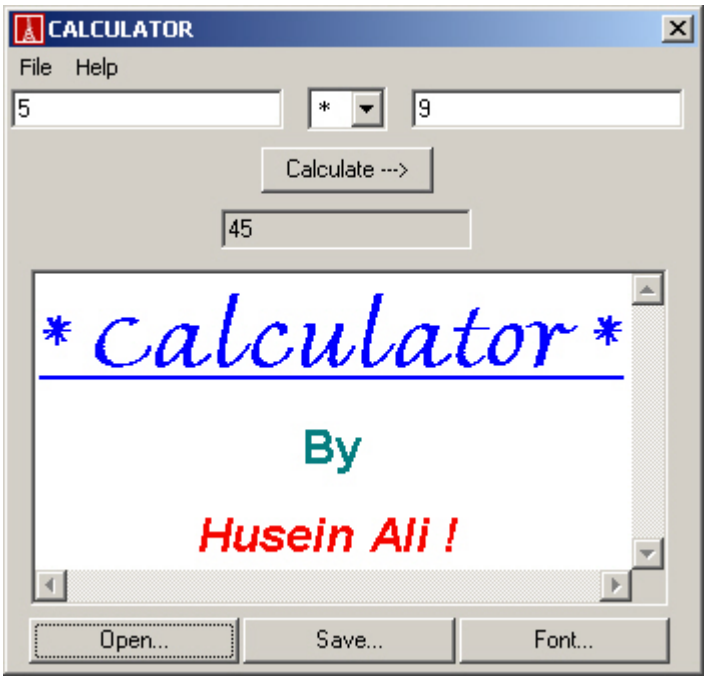
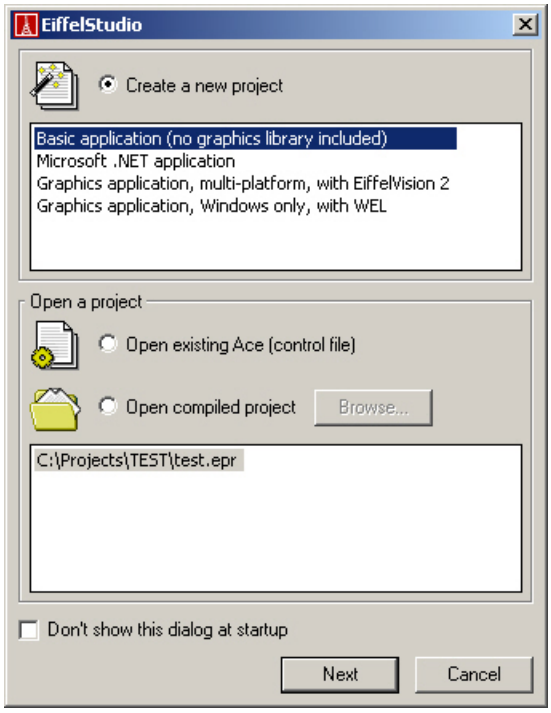
EIFFELVISION 2 TUTORIAL

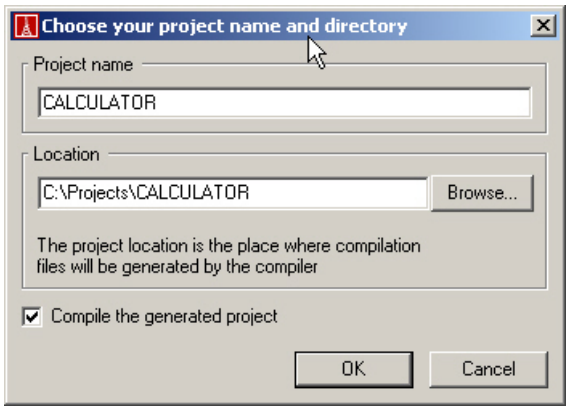
This tutorial will introduce you on how to build GUI applications with the new cross-platform EiffelVision 2 class library. The application we'll work on will be a simple calculator. We'll keep building the calculator incrementally and most importantly *manually*. EiffelStudio has many wizards to automate the application development process, but we'll strictly avoid them. The calculator will look something similar to the figure below:



First off, we'll assume that you already know enough about the Eiffel language i.e. basic OO concepts as it relates to Eiffel. We'll also assume that you're familiar enough to be able to navigate around EiffelStudio. Please refer to the **EiffelStudio: A Guided Tour** manual located at http://www.eiffel.com/doc/online/eiffel50/intro/studio to learn how to use EiffelStudio. We'll make use of the EiffelVision 2 class library, the most recent version released with EiffelStudio. EiffelVision 2 is a portable, object-oriented GUI class library. At the lower level of EiffelVision 2, platform specific libraries like WEL (Windows Eiffel Library) and GEL (GTK Eiffel Library for Unix) cover the graphical mechanisms of Microsoft Windows and X-Windows. As a matter of fact, the new EiffelStudio IDE was built using EiffelVision 2 ☺. The new revised version of this library sports a completely redesigned framework with many new enhancements, the most notable of which is the introduction of ***agents*** for event-handling. We'll go through a quick overview of what agents are later in the tutorial. Let's get started now:

Run *EiffelStudio* and from the *File* menu, select *New Project…* and you'll be prompted with a wizard like dialog box. Select the '**Basic application (no graphics library included)**' option. You can use the '**Graphics application, multi-platform, with EiffelVision2**' option, however, that'll defeat the purpose of this tutorial, as the wizard will generate all the code automatically. If you're comfortable enough, you may want to choose the later option and then customize the resultant generated application, but I suggest you follow the approach I'm taking here. This way, you'll know exactly what is entailed in creating an EiffelVision 2 based application.





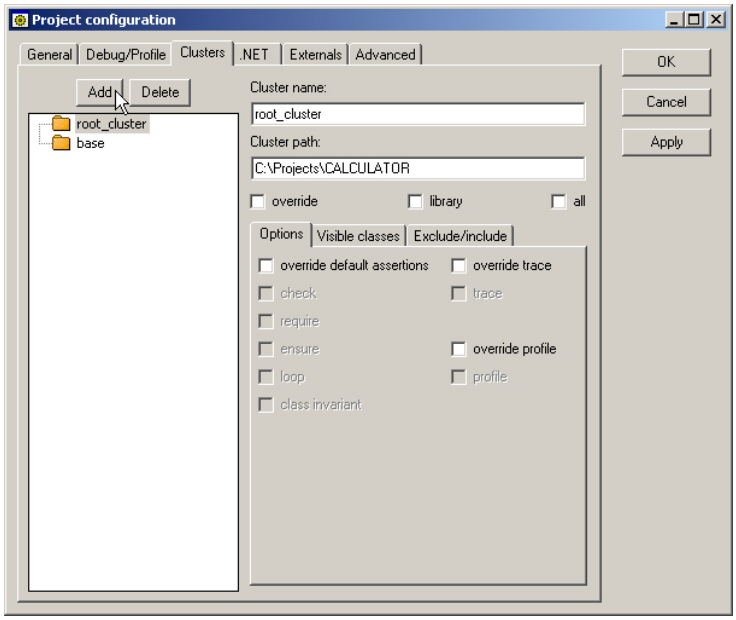
Click on *Next* and you'll be presented with the *project name* dialog box. Set the name to **Calculator** as in the above figure. Click on *OK* and EiffelStudio will automatically create a *ROOT_CLASS* and compile the project for you. The ROOT_CLASS is an empty skeleton that doesn't do anything useful. By default, the two clusters present in our project are **base** and **root_cluster**. Before we start using EiffelVision 2 specific classes, we need to add the **vision2** and the **wel** cluster in our project. Select the **Project → Project settings** menu-item to go to the *Project configuration* dialog box. Select the **cluster** tab and click on the **Add** button and type in **vision2** for the *Cluster name*. Next, select the **vision2** cluster that you just added and set the *Cluster path* to **$ISE_EIFFEL\library\vision2**. Similarly, add the **wel** *Cluster name* with *Cluster path* as **$ISE_EIFFEL\library\wel**. This should work if you've installed EiffelVision 2 correctly during the EiffelStudio setup. Make sure the **library** and **all** checkboxes are ticked while the **vision2** and **wel** cluster is highlighted. Next, select the *Externals* tab and add the following entries for the *Include Path*:

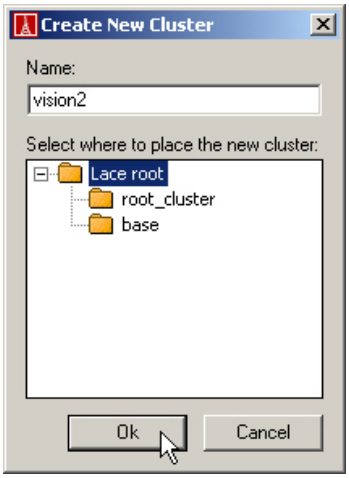
$(ISE_EIFFEL)\library\wel\spec\windows\include
$(ISE_EIFFEL)\library\vision2\spec\include

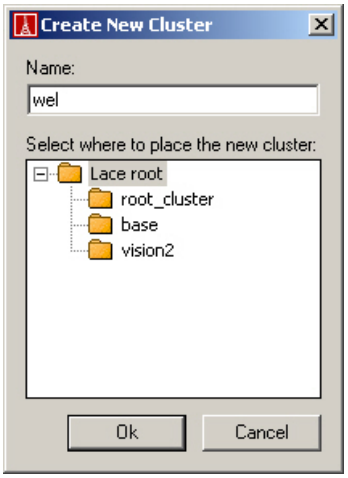
Now add the following entries for the *Object file* path:

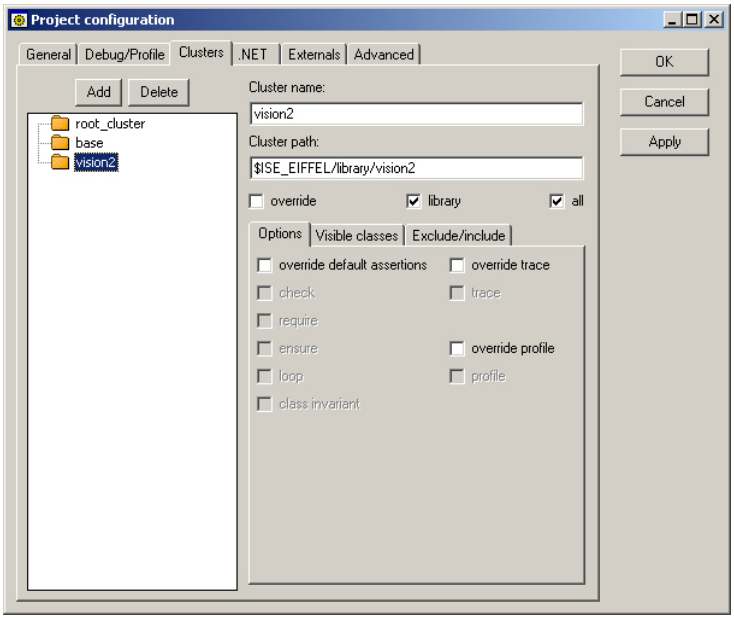
$(ISE_EIFFEL)\library\vision2\spec\$(ISE_C_COMPILER)\lib\vision2.lib
$(ISE_EIFFEL)\library\wel\spec\$(ISE_C_COMPILER)\lib\wel.lib
$(ISE_EIFFEL)\library\vision2\spec\$(ISE_C_COMPILER)\lib\libpng.lib
$(ISE_EIFFEL)\library\vision2\spec\$(ISE_C_COMPILER)\lib\zlib.lib

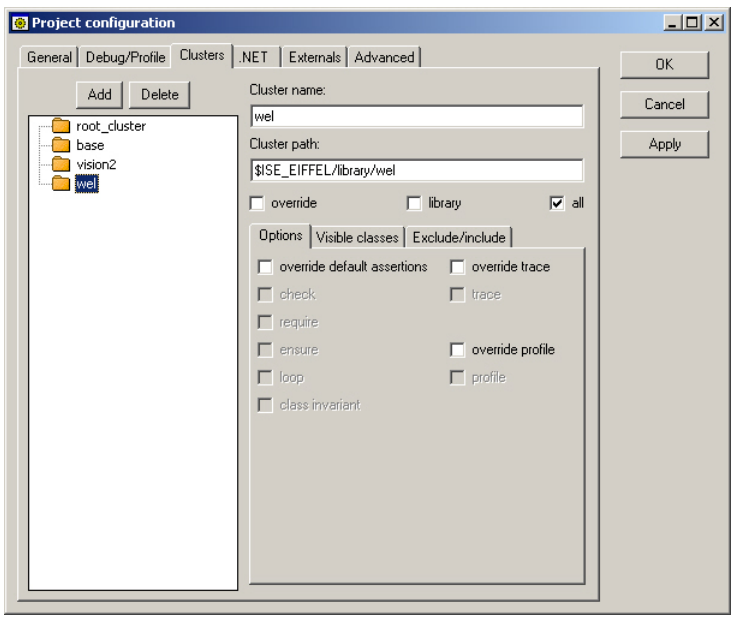
In addition, you'll also need to make sure certain entries are included in the *Exclude/include* tab list. Adding all those entries is cumbersome; to relieve you of extraneous effort, I've posted the project *ace* file for you to copy. The project configuration dialog box is really nothing more than a visual mapping of the project *ace* file. Anything you add to the *ace* file is parsed and displayed here and vice-versa. Did we really have to go through all these steps to set up the library dependencies? Absolutely NO ☹. Can we simply copy the *ace* file and forget setting this and that dependencies manually? Absolutely YES ☺. It's really up to you! Follow these screen-shots to see how it's done visually:











Next, press OK to dismiss the *Project configuration* dialog box. You MUST **compile** the project now for the classes to be visible on the clusters tree-view pane in EiffelStudio. After having done that, you'll be able to navigate through all the classes that are available in the **EiffelVision 2** and the **wel** class libraries. At this point, your ace file should read like this:

```
system
      calculator

root
      ROOT_CLASS: make

default
      assertion(require)
      arguments(" ")
      disabled_debug(yes)
      disabled_debug("dlg_dispatcher")
      disabled_debug("gdi_count")
      disabled_debug("vision2_windows")
      disabled_debug("vision2_windows_gdi")
      disabled_debug("wel")
      disabled_debug("wel_gdi_count")
```

```
	disabled_debug("wel_gdi_references")
	disabled_debug("win_dispatcher")
	debug(no)
	line_generation(no)
	profile(no)
	trace(no)
	il_verifiable(yes)
	msil_generation_type("exe")
	check_vape(yes)
	console_application(no)
	address_expression(no)
	array_optimization(no)
	dead_code_removal(yes)
	dynamic_runtime(no)
	exception_trace(no)
	inlining(no)
	multithreaded(no)

-- EiffelBase

cluster
	root_cluster:		"."


	all base:		"$ISE_EIFFEL\library\base"
		exclude
			"table_eiffel3"; "desc";
		end

	all vision2:		"$ISE_EIFFEL\library\vision2"
		exclude
			"gtk"; "EIFGEN"; "tmp"; "temp"; "release"; "obsolete"; "CVS";
		end

	all wel:		"$ISE_EIFFEL\library\wel"

external

	include_path:
		"$(ISE_EIFFEL)\library\wel\spec\windows\include",
		"$(ISE_EIFFEL)\library\vision2\spec\include"

	object:
		"$(ISE_EIFFEL)\library\vision2\spec\$(ISE_C_COMPILER)\lib\vision2.lib",
		"$(ISE_EIFFEL)\library\wel\spec\$(ISE_C_COMPILER)\lib\wel.lib",
		"$(ISE_EIFFEL)\library\vision2\spec\$(ISE_C_COMPILER)\lib\libpng.lib",
		"$(ISE_EIFFEL)\library\vision2\spec\$(ISE_C_COMPILER)\lib\zlib.lib"

end
```

So far, we've only done preliminary work. Let's start using the EiffelVision 2 classes now ☺.

Right now the *ROOT_CLASS* that was automatically generated looks like this:

```
indexing
	description: "System's root class"
	note: "Initial version automatically generated"

class
	ROOT_CLASS

create
	make
```

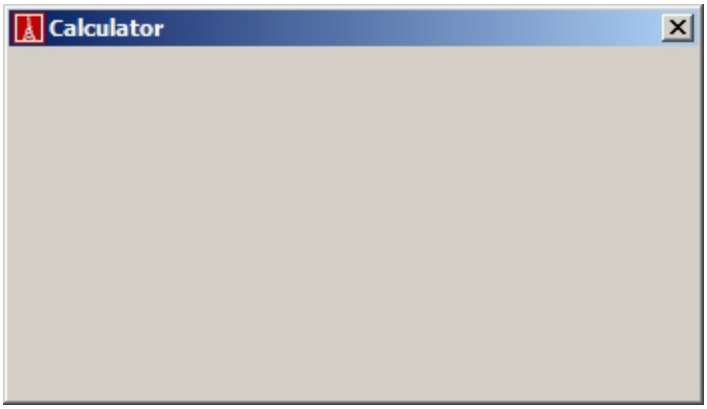
```
feature -- Initialization

	make is
		--| Creation procedure.
		do
		end

end -- class ROOT_CLASS
```

Not very interesting, huh? Let's modify our ROOT_CLASS to create a simple main window for our calculator. But first, a little chip about class **EV_APPLICATION**, the most important class in the EiffelVision2 framework. Every EiffelVision2 based application root class **MUST** either *descend* from the **EV_APPLICATION** class or use it as a *client* (instance variable/feature). The EV**_APPLICATION** class is the main entry point of ALL EiffelVision2 applications; think of it as being like the root class for all EiffelVision2 based applications. It's primarily responsible for initializing the underlying operating system specific graphical toolkit and for starting the application main event-loop, which in fact is the heart of an EiffelVision2 based application. We'll pick the former choice and descend from **EV_APPLICATION**. The following code snippet will create a simple main window for our calculator:

```
indexing
	description: "System's root class"
	note: "Initial version automatically generated"

class
	ROOT_CLASS

inherit
	EV_APPLICATION

create
	make

feature -- Initialization

	make is
		--| Creation procedure
		local
			main_window: EV_TITLED_WINDOW
		do
			default_create

			-- Create main window
			create main_window.make_with_title ("Calculator")

			main_window.set_size (350, 200)
			main_window.disable_user_resize
			main_window.close_request_actions.extend (agent destroy)
			main_window.show

			launch
		end

end -- class ROOT_CLASS
```

Note: All EiffelVision2 classes start with the prefix ***EV_*** to distinguish them from other classes.

If you run the above application, this is what you get:



Let's understand what the above code accomplishes. By now, you must know that ALL Eiffel classes implicitly descend from class **ANY**. If this sounds unfamiliar, please consult an Eiffel language reference for details. We'll quickly skim through this background area. Class **ANY** provides core features that perform type conformance tests, cloning, deep/shallow copy, equality tests etc. Class **EV_ANY**, which descends from **ANY** is the EiffelVision2 version of **ANY** for all classes found in the EiffelVision2 class library i.e. every class in the EiffelVision2 library ultimately descends from class **EV_ANY**. Class **EV_ANY** augments class **ANY** with additional features, the most important of which is probably ***default_create***. This feature is responsible for correctly initializing an EiffelVision2 class object. A rule of thumb with ANY EiffelVision2 class is that you MUST, either directly or indirectly call ***default_create*** during object creation. But then how does it work internally ? How can one single feature correctly anticipate and initialize all possible descendent class objects ? You don't need to understand this section for the rest of the tutorial; you may very well skip it if you wish.

Do you remember the word **polymorphism**, the magic at the heart of OO programming ? The creation sequence for ALL EiffelVision2 objects is such: feature ***default_create*** calls feature **intitialize** which is redefined (overridden) for any descendent specific initialization. It's that simple ☺. In addition, there are other helper features that take care of platform specific initialization. The normal pattern is that ***default_create*** will produce a properly initialized default object and any special convenience creation features will call ***default_create*** then do their extra work. The important thing to remember about class **EV_ANY** is that it provides a common interface for all EiffelVision2 classes across all platforms. Platform implementation specific delegate classes are managed by **EV_ANY** transparently behind the scenes, but that's something we really don't care about.

We're now prepared to analyze the above code snippet. Our **ROOT_CLASS** descends from class **EV_APPLICATION** as indicated before so no surprise there. The creation feature ***make***, declares a local variable of type **EV_TITLED_WINDOW**. This class is exactly what its name suggests. It encapsulates a window with a title caption. The first line of the ***make*** feature invokes feature ***default_create*** inherited from **EV_APPLICATION**. Remember what we just said: ***default_create*** MUST be called for ALL EiffelVision2 class objects. The next line creates an object of type **EV_TITLED_WINDOW**. This creates a window titled *Calculator*. The next line calls feature ***set_size*** to set the pixel width/height of the window.
Next, feature ***disable_user_resize*** is invoked; this will prevent the user from resizing our window (very user-friendly ☺). For now, ignore the next line that reads *close_request_actions.extend* (**agent** *destroy*).
It's an event handler mechanism that requires an understanding of ***agents*** which we'll cover shortly.

Although we've created the window, it only exists in memory. We need to DISPLAY it to the user. Feature ***show*** does precisely that. It displays the window on the screen. Now that we have the window on the screen, we need to instruct our **ROOT_CLASS** to enter an *event-loop*. Every GUI application typically has an event-loop. Events like mouse-clicks, keyboard strokes etc. that are of interest are constantly fired during an applications lifetime. These events are trapped by the application *event-loop* and further dispatched to the appropriate window and usually results in a feature execution appropriate for the event that was fired. The loop runs continuously in the background awaiting events until the application is closed. Feature ***launch*** starts the application *event-loop* running.

At this point, I'm quite certain that you're probably wondering where the heck is ***default_create*** being called for the *main_window* object. After all, I did mention that this feature needs to be called for EVERY EiffelVision2 class object ? You're quite right. However, this time, the feature is being called *indirectly* for us.

As it turns out, creation feature ***make_with_title*** in class **EV_TITLED_WINDOW** calls ***default_create*** automatically.

Ok, let's move on now to create a couple of widgets on the main window. Look at Fig 1. We need to create menus and add some buttons, edit boxes and combo boxes on the main window. Before we do that, I'd like you to pause and consider this: We've used the **EV_TITLED_WINDOW** class as a client in our **ROOT_CLASS**. That's perfectly legitimate. However, for reasons that will apparent later, it's actually better to create our own custom class that inherits from **EV_TITLED_WINDOW** with focused functionality. We'll now rewrite what we have so far and use the **EV_TITLED_WINDOW** in an inheritance relationship as opposed to a client-supplier relationship. At this point, we'll have TWO classes (and hence two source files):

```
indexing
        description: "System's root class"
        note: "Initial version automatically generated"

class
        ROOT_CLASS

inherit
        EV_APPLICATION

create
        make

feature -- Initialization

        make is
                -- Creation procedure
                local
                        main_window: CALCULATOR_WINDOW
                do
                        -- Initialize application object
                        default_create

                        -- Create main window
                        create main_window
                        main_window.show

                        -- Start event-loop
                        launch
                end

end -- class ROOT_CLASS

indexing
        description: "Calculator window class"
        author: "Husein Ali"
class
        CALCULATOR_WINDOW

inherit
        EV_TITLED_WINDOW
                redefine
                        initialize
                end
create
        default_create

feature -- Initialization

        initialize is
                        -- Build the interface for this window.
                do
                        Precursor {EV_TITLED_WINDOW}
                        close_request_actions.extend (agent close_window)
```

```
                        set_title (window_title)
                        set_size (window_width, window_height)
                        disable_user_resize
                end

feature {NONE} -- Implementation, Close event

        close_window is
                        -- The user wants to close the window.  Return reference to 'destroy' feature.
                do
                        (create {EV_ENVIRONMENT}).application.destroy
                end

feature {NONE} -- Implementation / Constants

        Window_title: STRING is "CALCULATOR"
                        -- Title of the window.

        Window_width: INTEGER is 350
                        -- Initial width for this window.

        Window_height: INTEGER is 200
                        -- Initial height for this window.

end -- class CALCULATOR_WINDOW
```

So how is this approach different from the previous one we had? Look carefully at class **ROOT_CLASS**. In the former approach, all the window specific initialization was performed in the *make* feature of class **ROOT_CLASS** by calling features on the **EV_TITLED_WINDOW** object, namely *main_window*. This time around however, all the initialization is performed in the *initialize* feature of our custom class **CALCULATOR_WINDOW** which inherits from class **EV_TITLED_WINDOW**. The *make* feature in class **ROOT_CLASS** only creates the **CALCULATOR_WINDOW** object and calls feature *show*. The rest of the work (initialization) is done automatically inside the **CALCULATOR_WINDOW**. This is a good example of encapsulation, an important feature of OO programming.

Let's now see how **CALCULATOR_WINDOW** functions. The creation feature of class **CALUCLATOR_WINDOW** is delegated to feature ***default_create***, inherited from **EV_TITLED_WINDOW**. Feature ***default_*create** as discussed before calls feature *initialize* which we override. A recurring question again: where is ***default_create*** called? Look carefully again: The creation feature for this class is ***default_create***; it's *automatically* called when an object of this class type is instantiated. Yikes, gotcha there! The statement `Precursor {EV_TITLED_WINDOW}` is necessary because we need to correctly initialize all the ancestors of this class. Next, features *set_size*, *set_title* and *disable_user_resize* are exactly as discussed before. The expression reading `close_request_actions.extend (agent close_window)` handles the main windows **close** event and is similar to the previous `close_request_actions.extend (agent destroy)` which we had ignored. Since event handling is a very common task in EiffelVision2, let's first gain an understanding of **agents**, a mechanism via which EiffelVision2 handles *events*. If you're already accustomed to working with function pointers in C/C++, procedural types in Object Pascal or delegates under the Microsoft .NET framework, then you already know what agents entail. Agents are however much more than that but those are details that aren't relevant to our discussion. We'll only consider an oversimplified case with enough rigor to be able to use them effectively with EiffelVision2 without concerning ourselves too much with details.

In the OO world, one of the cornerstone concepts is the notion of objects and operations i.e. object instances represent information equipped with operations (features in Eiffel) that can be invoked using the object instance qualifier. This draws a clear line of distinction between an object and any operations that may be invoked on the object. Think of an object as a separate entity distinct from all its operations. Simply stated, an object is NOT an operation and an operation is NOT an object. This is the conventional form of decoupling objects from its associated operations. This division works for most cases, however, there are cases in which it makes intuitive sense to treat operations as objects and pass them around to software elements which can then use them to execute the operation the object encapsulates. This scheme separates the place of an operations *definition* from the place of its *execution*. As such, the definition can be *incomplete*, since the missing details are provided at the instant of any particular execution. Confusing? Let me elaborate: Say

we're developing an extremely powerful generic feature that can integrate a large class of functions of type *f(x)*. Our problem is that the integrate feature has no prior knowledge of the specific nature of *f(x)*. It could be a simple function like $f(x) = x$, a quadratic function like $f(x) = x^2$ or a complicated polynomial. This forces us to conclude that the integrate feature should not be tied to any specific *f(x)* and that the actual computation of *f(x)* at any *x* is decoupled from the integrate feature i.e. the computation of *f(x)* is performed OUTSIDE of the integrate feature. The feature corresponding to *f(x)* can be a feature in *any* class that has a signature with input parameter type corresponding to the function domain type *x* (probably a **floating-point**) and a return type corresponding to the function *f(x)* range type (probably a **floating-point** as well). Ok, so far so good. Now, the question that arises next is how will the integrate function actually call the feature that corresponds to *f(x)* to compute *f(x)* at a given value of *x*. This is where agents come into play. An agent object encapsulates a feature (operation) within a given objects scope and stores enough information about it such that the agent may then be used to call upon the feature it encapsulates. In our case, we'd like to write an expression that only *describes* the calls intended for the actual *f(x)* computation and *execute* that description later on at any time. Think of this technique as **delayed calls**.

Let us work through an example. Suppose our integrate feature is called *integrate* and that *f'(x)* is the *anti-derivative* of *f(x)*. Let β = an **agent class type** encapsulating a feature that computes *f'(x)* (the *anti-derivative* of *f(x)*). The signature of the feature that computes *f'(x)* MUST have a single input double type parameter with return type double e.g. ***feature_X (in: DOUBLE): DOUBLE***. This is what the pseudo-code for the implementation of feature *integrate* looks like:

```
integrate(function_to_call: β, lower_bound: INTEGER , upper_bound: INTEGER): DOUBLE is
require
        meaningful_interval: lower_bound <= upper_bound
local
    upper_value, lower_value: DOUBLE
do
        -- CALL feature represented by agent object passing upper_bound as a parameter.
        -- This has the effect of calling f'(upper_bound).
        upper_value := function_to_call(upper_bound)

        -- Call feature represented by agent object passing lower_bound as a parameter.
        -- This has the effect of calling f'(lower_bound).
        lower_value := function_to_call(lower_bound)

        -- Return result of the integrate computation.
        Result := upper_value – lower_value
end
```

Note: The syntax for the β declaration and the call *function_to_call*(…) has been simplified in the above example. In actual usage, the **FUNCTION/PROCEDURE** and the **TUPLE** class along with related syntactic elements are employed, but that's a complication that we skip here altogether.

Next, assume that we have the following features for functions *f'(x)* and *g'(x)*:

function_X(x: DOUBLE): DOUBLE
function_G(x: DOUBLE): DOUBLE

Note: *function_X* and *function_G* represent the *anti-derivative* versions of the functions *f(x)* and *g(x)*.

These features may reside in the scope of a **single** object of the **same** class type or **two** different objects of a **single** class type or possibly **two** different objects of **two** different class types. The location of the feature is irrelevant as long as it satisfies the specified agent type signature β. We'd now like to integrate these functions from bounds **zero** to **one**. The call will be implemented as such:

```
        value := integrate(agent function_X, 0, 1)
        value := integrate(agent function_Y, 0, 1)
```

Note: We assume *function_X* and *function_G* are currently in scope. If not, an object instance will be necessary to specify a fully qualified feature reference.

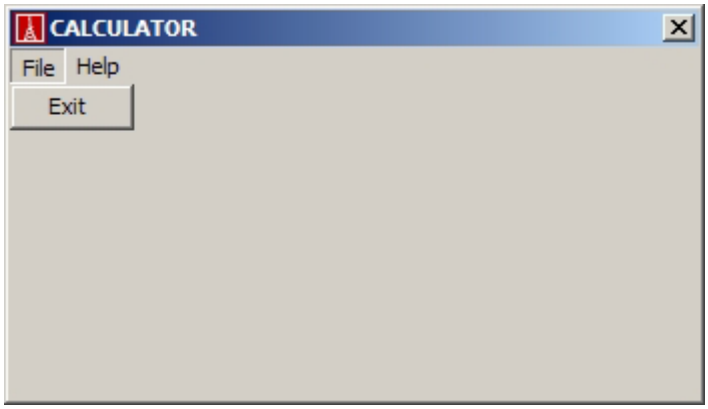
Nothing magical; really quite simple as you can see. The keyword agent creates an agent object that refers to the feature that follows it, in this case, *function_X* and *function_G*. Notice how the *integrate* feature accommodates BOTH the functions by decoupling its implementation from the actual implementation of the function being integrated. Obviously, passing an agent object that refers to an *anti-derivative* feature makes the *integrate* feature really trivial to implement but I hope the core concept of one possible use of agents has sunk in you head ☺ If not, go back and reread this section ☹. Furthermore, in the above example, the value of the *integrate* feature is returned before execution continues with the next line. In other words, execution is **synchronous**. It's possible to pass agent objects and then continue execution and expect the agent to be used at ANY time thereafter also known as **asynchronous** execution. In the case of *event-handling* with EiffelVision2, we register an agent object that will be executed when a particular event of interest happens. For example, we can register an agent object for a button widget by making use of feature *select_actions* of class **EV_BUTTON**. After the agent is registered, any clicks on the button will result in the routine (feature) execution encapsulated by the agent registered previously.

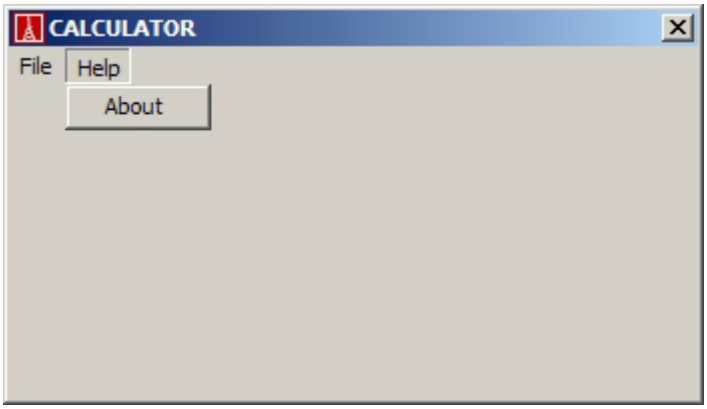
In passing, one last item before we conclude agents: It's very much possible to supply some parameters and leave others open during the agent object construction for a feature and then fill them just before the actual execution takes place. In particular, you may build an agent from a routine with more arguments than expected in the final call, and you may set the values of some arguments at the time you define the agent. This provides for a very powerful programming model but is unfortunately outside the scope of this tutorial so we won't get into it. I thought I'd just mention it for the benefit of the die-hard EiffelVision 2 fans ☺. Finally, the bottom line about an event associated agent object is that it's nothing but a triple **[Control X, Event Y, Operation Z]** tuple. Simply stated, some EiffelVision2 **Control X** object receives an **Event Y** signal which results in execution of some **Operation Z**. Period ☺. For detailed information on agents, please see Bertrand Meyers article on *Agents, iteration and introspection* from the ISE website (check the URL http://www.eiffel.com/doc/manuals/language/agent/page.html).

Let's now go back to feature *initialize* in the **CALCULATOR_WINDOW** class. The expression `close_request_actions.extend (agent close_window)` requests the window object to call the `close_window` feature whenever a close event is received i.e. an attempt is made to close the window. A slight detour before we explain feature `close_window`: All EiffelVision2 applications create an **EV_ENVIRONMENT** class type object upon startup. This class has facilities for inspecting global environment information i.e. environment, application and system wide global variables are stored here. Remember, every EiffelVision2 based application root class **MUST** either *descend* from the **EV_APPLICATION** class or use it as a *client* (instance variable/feature). In either case, the **EV_APPLICATION** conformant object instance created by/in the **ROOT_CLASS** is cached in the **EV_ENVIRONMENT** object (created automatically upon initialization). The usefulness of having an **EV_ENVIRONMENT** object is that we can access our **EV_APPLICATION** object instance even when we're NOT in the scope of our **ROOT_CLASS** anymore. Class **EV_APPLICATION** has one very important feature called *destroy* which simply exits an application. We would like to exit our application when a close event is signalled/received in our window object. Therefore, we need to construct an agent that encapsulates the *destroy* feature of class **EV_APPLICATION** so that our application is exited when the close event is detected. The only way to access the **EV_APPLICATION** conformant object instance inside the **CALCULATOR_WINDOW** class, which is outside the SCOPE of the **ROOT_CLASS**, is to use an **EV_ENVIRONMENT** object. The `close_window` feature constructs such an instance and then via the *application* feature (which in fact is the **EV_APPLICATION** object instance we're after) returns a reference to the *destroy* feature. This reference is returned by the `close_window` feature and then used for the agent object construction.

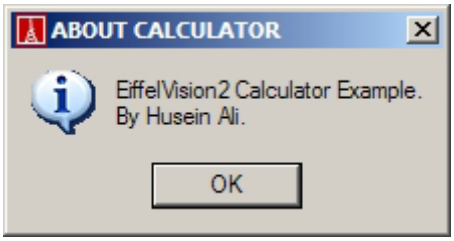
This concludes the first phase of the calculator. We now have the core basic skeleton of our calculator working by using the **EV_TITLED_WINDOW** in an inheritance relationship. The next sections will only augment class **CALCULATOR_WINDOW** to provide for more functionality.

Our next task is to add menu-items in the main window. The visual changes that need to be incorporated to the main window are as such:





Selecting the **Exit** menu-item should close the application.... Duh ☺. Selecting the **About** menu-item should bring up this dialog box:



The following is the updated source for the **CALCULATOR_WINDOW** class:

```
indexing
        description: "Calculator window class"
        author: "Husein Ali"

class
        CALCULATOR_WINDOW

inherit
        EV_TITLED_WINDOW
                redefine
                        initialize
                end

create
        default_create



feature -- Initialization

        initialize is
                        -- Build the interface for this window.
                do
                        Precursor {EV_TITLED_WINDOW}

                        -- Create and add the menu bar.
                        build_menu_bar
                        set_menu_bar (standard_menu_bar)

                        -- Execute `close_window' when the user clicks
                        -- on the cross in the title bar.
                        close_request_actions.extend (agent close_window)

                        -- Set the title of the window
                        set_title (window_title)
```

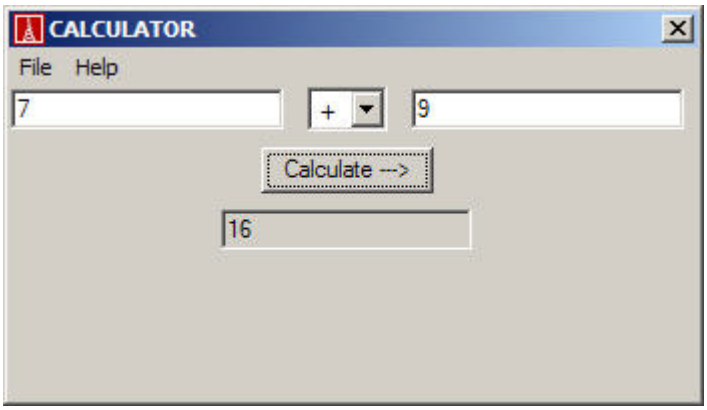
```
                        -- Set the initial size of the window
                        set_size (window_width, window_height)

                        -- Prevent user from resizing calculator
                        disable_user_resize
                end

feature {NONE} -- Implementation, Close event

        close_window is
                        -- The user wants to close the window
                do
                        (create {EV_ENVIRONMENT}).application.destroy
                end

        show_about_box is
                        -- Display an about box.
                local
                        about_dialog: EV_INFORMATION_DIALOG
                do
                        create about_dialog.make_with_text ("EiffelVision2 Calculator
                                                            Example.%NBy Husein Ali.")
                        about_dialog.set_title ("ABOUT CALCULATOR")
                        about_dialog.show_modal_to_window (Current)
                end

feature {NONE} -- Menu implementation

        standard_menu_bar: EV_MENU_BAR
                        -- Standard menu bar for this window.

        file_menu: EV_MENU
                        -- "File" menu for this window (contains Exit...)
        help_menu: EV_MENU
                        -- "Help" menu for this window (contains About...)

        build_menu_bar is
                        -- Create and populate standard_menu_bar.
                require
                        menu_bar_not_yet_created: standard_menu_bar = void
                local
                        menu_item: EV_MENU_ITEM
                do
                        -- Create the menu bar.
                        create standard_menu_bar


                        -- Add the "File" menu
                        create file_menu.make_with_text ("&File")
                        create menu_item.make_with_text ("E&xit")
                        file_menu.extend (menu_item)
                        standard_menu_bar.extend (file_menu)

                        -- Close application when 'Exit' menu-item selected.
                        menu_item.select_actions.extend (agent close_window)

                        -- Add the "Help" menu
                        create help_menu.make_with_text ("&Help")
                        create menu_item.make_with_text ("&About")
                        help_menu.extend (menu_item)
                        standard_menu_bar.extend (help_menu)


                        -- Display ABOUT box when 'About' menu-item selected.
                        menu_item.select_actions.extend (agent show_about_box)
```

```
                ensure
                        menu_bar_created: standard_menu_bar /= void and then
                                          not standard_menu_bar.is_empty
                end

feature {NONE} -- Implementation / Constants

        Window_title: STRING is "CALCULATOR"
                        -- Title of the window.

        Window_width: INTEGER is 350
                        -- Initial width for this window.

        Window_height: INTEGER is 200
                        -- Initial height for this window.

end -- class CALCULATOR_WINDOW
```

Feature *initialize* has been updated to create a main-main for our window. The actual task of creating the menu is delegated to feature *build_menu_bar*. The main-menu object is feature *standard_menu_bar* of class type **EV_MENU_BAR**. This class type allows one to build a main-menu bar for a window by hierarchically adding/arranging sub menu/item objects of class type **EV_MENU**, **EV_MENU_ITEM** etc. In our case, we need two sub-menus namely ***File*** and ***Help***. Feature *build_menu_bar* starts off by creating the main **EV_MENU_BAR** object. Next, an **EV_MENU** object to represent the *File* sub-menu is created. A temporary local variable of type **EV_MENU_ITEM** is used to create the ***Exit*** menu-item object. Notice how the menu-item is then *extended* on to the ***File*** sub-menu followed by the ***File*** sub-menu itself being extended on to the main-menu. This is how the menu hierarchy is built. Also notice how we register a menu-item click/select event. In the above example, after having created the ***File*** sub-menu and the ***Exit*** menu-item, we specify that feature *close_window* be called whenever a click/select event takes places for the ***Exit*** menu-item. Remember, at this point of execution, local variable *menu_item* contains a reference to the ***Exit*** menu-item that was just created a few lines above. Next, the same technique is employed to build the ***Help*** sub-menu containing the ***About*** menu-item. In this case however, the click/select event-handler for the ***About*** menu-item is feature *show_about_box*. All this feature does is display an about dialog box to the user. This is easily accomplished by making use of class **EV_INFORMATION_DIALOG**. Check feature *show_about_box*; it's really trivial to follow ☺.

After the menu is constructed, we make use of feature *set_menu_bar* to set the constructed menu object hierarchy as the main-menu for our window. Since feature *standard_menu_*bar is the root of the menu constructed, it's passed as a parameter to *set_menu_bar*. Note: There can only be **one** main-menu active at any given time for a given window. Calling *set_menu_bar* will remove the previously set main-menu, if any.

Now let's add a couple of widgets on the main window so that we may use the calculator to actually calculate something ☺. The updated **CALCULATOR_WINDOW** class along with the corresponding visual changes follows:



```
indexing
        description: "Calculator window class"
        author: "Husein Ali"

class
        CALCULATOR_WINDOW

inherit
        EV_TITLED_WINDOW
                redefine
                        initialize
                end

create
        default_create

feature -- Initialization

        initialize is
                        -- Build the interface for this window.
                do
                        Precursor {EV_TITLED_WINDOW}

                        -- Create and add the menu bar.
                        build_menu_bar
                        set_menu_bar (standard_menu_bar)

                        -- Create the operator/operand primitives
                        build_primitives

                        -- Execute `close_window' when the user clicks
                        -- on the cross in the title bar.
                        close_request_actions.extend (agent close_window)

                        -- Set the title of the window
                        set_title (window_title)

                        -- Set the initial size of the window
                        set_size (window_width, window_height)

                        -- Prevent user from resizing calculator
                        disable_user_resize
                end

feature {NONE} -- Implementation events

        close_window is
                        -- The user wants to close the window
                do
                        (create {EV_ENVIRONMENT}).application.destroy
                end

        show_about_box is
                        -- Display an about box.
                local
                        about_dialog: EV_INFORMATION_DIALOG
                do
                        create about_dialog.make_with_text ("EiffelVision2 Calculator
                                                            Example.%NBy Husein Ali.")
                        about_dialog.set_title ("ABOUT CALCULATOR")
                        about_dialog.show_modal_to_window (Current)
                end

feature {NONE} -- Menu implementation
```

```
        standard_menu_bar: EV_MENU_BAR
                        -- Standard menu bar for this window.

        file_menu: EV_MENU
                        -- "File" menu for this window (contains Exit...)

        help_menu: EV_MENU
                        -- "Help" menu for this window (contains About...)

        build_menu_bar is
                        -- Create and populate standard_menu_bar.
                require
                        menu_bar_not_yet_created: standard_menu_bar = void
                local
                        menu_item: EV_MENU_ITEM
                do
                        -- Create the menu bar.
                        create standard_menu_bar


                        -- Add the "File" menu
                        create file_menu.make_with_text ("&File")
                        create menu_item.make_with_text ("E&xit")
                        file_menu.extend (menu_item)
                        standard_menu_bar.extend (file_menu)

                        -- Close application when 'Exit' menu-item selected.
                        menu_item.select_actions.extend (agent close_window)

                        -- Add the "Help" menu
                        create help_menu.make_with_text ("&Help")
                        create menu_item.make_with_text ("&About")
                        help_menu.extend (menu_item)
                        standard_menu_bar.extend (help_menu)

                        -- Display ABOUT box when 'About' menu-item selected.
                        menu_item.select_actions.extend (agent show_about_box)
                ensure
                        menu_bar_created: standard_menu_bar /= void and then
                                          not standard_menu_bar.is_empty
                end

feature {NONE} - Misc. primitives implementation

        enclosing_box: EV_FIXED
                        -- Invisible Primitives Container

        first_operand: EV_TEXT_FIELD

        second_operand: EV_TEXT_FIELD
                        -- Operator primitives

        result_text: EV_TEXT_FIELD
                        -- Calculation result text box.

        operator: EV_COMBO_BOX
                        Operator combo-box primitive

        calculate_btn: EV_BUTTON

        calculate_result is
                        -- Calculate the result after the user clicks
                        -- on the 'Calculate --->' button.
                local
                        value: REAL
                        error_dialog: EV_INFORMATION_DIALOG
```

```
		do
			-- Make sure the operands are valid real numbers
			if (first_operand.text_length > 0) and (first_operand.text.is_real)
                                                and
			   (second_operand.text_length > 0) and (second_operand.text.is_real)
			then
				-- Clear result edit text box contents.
				result_text.remove_text

				-- Calculate :-)
				if operator.selected_item.text.is_equal (" +") then

					value := first_operand.text.to_real +
						second_operand.text.to_real

				elseif operator.selected_item.text.is_equal (" -") then

					value := first_operand.text.to_real -
						second_operand.text.to_real

				elseif operator.selected_item.text.is_equal (" *") then

					value := first_operand.text.to_real *
						second_operand.text.to_real
				elseif operator.selected_item.text.is_equal (" /") then
					-- Deal with Divide By Zero error.
				    if second_operand.text.to_real = 0 then

					create error_dialog.make_with_text ("Divide By Zero!");
					error_dialog.set_title ("Error");
					error_dialog.set_pixmap(default_pixmaps.error_pixmap);
					error_dialog.show_modal_to_window (Current)

				    else
					value := first_operand.text.to_real /
						second_operand.text.to_real
				    end

				end
				-- Set calculated result value
				result_text.set_text (value.out)
			end
		end

	build_primitives is
			-- Create the operand/operator primitives
		require
			enclosing_box_not_yet_created: enclosing_box = void
			first_operand_not_yet_created: first_operand = void
			second_operand_not_yet_created: second_operand = void
			result_text_not_yet_created: result_text = void
			operator_not_yet_created: operator = void
			calculate_btn_not_yet_created: calculate_btn = void
		do
			-- Avoid flicker on some platforms
			lock_update

			-- Cover entire window area with a primitive container.
			create enclosing_box
			extend (enclosing_box)

			-- Add first operator text edit box primitive
			create first_operand
			first_operand.set_minimum_width (138)
			enclosing_box.extend (first_operand)
```

```
                        -- Add operator combo-box primitive
                        create operator
                        operator.set_minimum_width (40)
                        operator.extend (create {EV_LIST_ITEM}.make_with_text (" +"))
                        operator.extend (create {EV_LIST_ITEM}.make_with_text (" -"))
                        operator.extend (create {EV_LIST_ITEM}.make_with_text (" *"))
                        operator.extend (create {EV_LIST_ITEM}.make_with_text (" /"))
                        operator.disable_edit
                        enclosing_box.extend (operator)
                        enclosing_box.set_item_x_position (operator, 149)

                        -- Add second operator text edit box primitive
                        create second_operand
                        second_operand.set_minimum_width (138)
                        enclosing_box.extend (second_operand)
                        enclosing_box.set_item_x_position (second_operand, 200)

                        -- Add 'calculate' button primitive
                        create calculate_btn.make_with_text ("Calculate --->")
                        calculate_btn.set_minimum_width (86)
                        calculate_btn.select_actions.extend (agent calculate_result)
                        enclosing_box.extend (calculate_btn)
                        enclosing_box.set_item_x_position (calculate_btn, 126)
                        enclosing_box.set_item_y_position (calculate_btn, 30)

                        -- Add result text edit box primitive
                        create result_text
                        result_text.set_minimum_width (127)
                        result_text.disable_edit
                        enclosing_box.extend (result_text)
                        enclosing_box.set_item_x_position (result_text, 105)
                        enclosing_box.set_item_y_position (result_text, 60)

                        -- Allow screen refresh on some platforms
                        unlock_update
                ensure
                        enclosing_box_created: enclosing_box /= void
                        first_operand_created: first_operand /= void
                        second_operand_created: second_operand /= void
                        result_text_created: result_text /= void
                        operator_created: operator /= void and then operator.count = 4
                        calculate_btn_created: calculate_btn /= void
                end

feature {NONE} -- Implementation / Constants

        Window_title: STRING is "CALCULATOR"
                        -- Title of the window.

        Window_width: INTEGER is 350
                        -- Initial width for this window.

        Window_height: INTEGER is 200
                        -- Initial height for this window.

end -- class CALCULATOR_WINDOW
```

Whew, I hope that wasn't a heavy initial dosage ☺. Relax, I assure you the changes aren't really that frightening as they seem to appear. Ok, take a deep breath and let's get our feet wet!

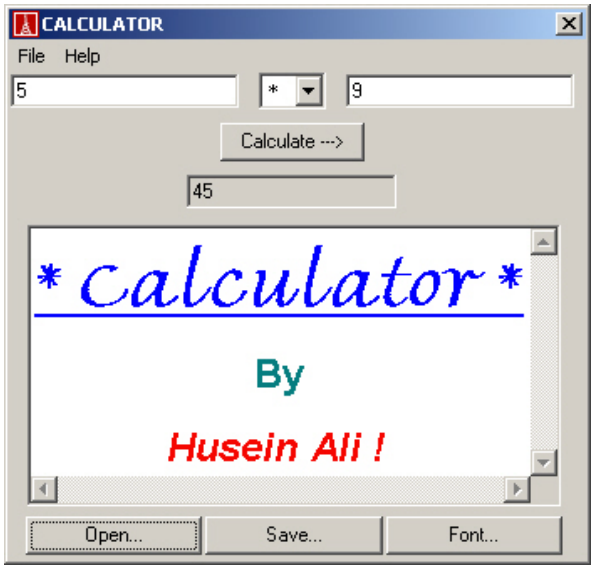
Feature *initialize* is updated again to build the user-interface primitives following the main-menu creation by making use of feature *build_primitives*. For our calculator, we need two text-edit widgets to enter the operands and a list to select the operator (either +, -, * or /) needed to compute the desired result. In addition, we also need a text-edit widget to show the result of the calculation and a button to trigger the

computation. There's a slight complication here that we need to understand before me move on. As is, a top-level **EV_TITLED_WINDOW** object can serve as a container for only **ONE immediate** child widget. Now, stop scratching your head and please don't ask me why that's the case because I don't have an answer. Esteemed engineers at ISE, are you listening? So, then how do we add our widgets (> ONE) on to the main window. The answer is to use a suitable child container widget (remember only **ONE** is permitted on the main window) that'll cover the whole window client area and allow us to add > one child widget on to it. Effectively, our components will be the children of this container which in turn will be the only immediate child of the main window. This is possible because a container is really just another widget that can be nested within another container. Ahhh, problemo solved. Don't we all just love to outsmart a computer ☺. The magical container that we'll use is class **EV_FIXED**. This is what feature *build_primitives* first starts off with. We create an **EV_FIXED** object called *enclosing_box*. Next, we **extend** it to the list of child controls on the main window, although, in this case, there can really be only ONE child control for the main window. Next, we create an object of type **EV_TEXT_FIELD** for the first operand and extend it to the list of child controls of the *enclosing_box*. Further, we call feature *set_width*, which resizes the width of the widget in pixel-units. Next, we instantiate an object of type **EV_COMBO_BOX** to create a list of operators. Notice how we also create instances of **EV_LIST_ITEM** and extend them as children of the **EV_COMBO_BOX** object just created. The syntax we employed for creating the list items is one where the list item objects are not assigned to any feature/variable. This is justified since we really don't need to access these item objects directly after they are created, hence, we employ this syntactic sugar. We also call feature *disable_edit* to disallow the user to edit the contents of the list. The list is positioned 149 pixels from the left of the *enclosing_box*. Immediately following that, we create a second **EV_TEXT_FIELD** object for the second operand and position it 200 pixel-units away from the left-hand side of *enclosing_box* (effectively, the main window because *enclosing_box* covers the whole window anyways!). This layout makes the operator list positioned in between the two operand widgets, which, you'll agree is intuitive. Following that, we create an **EV_BUTTON** object that encapsulates a button widget and place it underneath the operator list. Notice we register a click/select event-handler for the button. The event-handler is feature *calculate_result*, which we'll cover shortly. Finally, we create an **EV_TEXT_FIELD** object for the result of the computation which is positioned right beneath the button object.

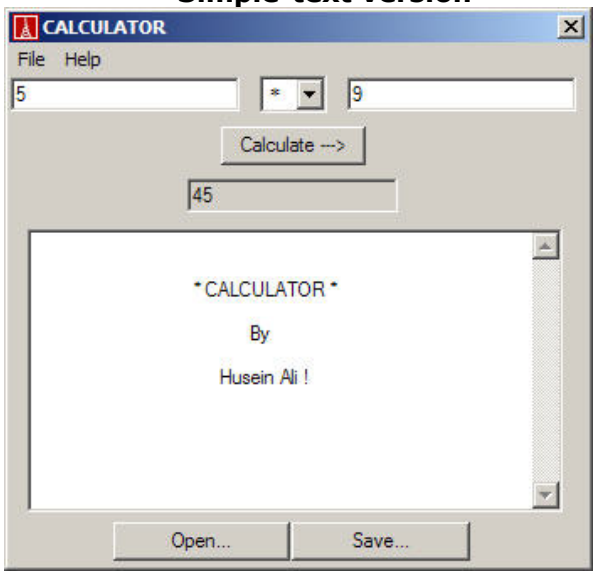
Feature *calculate_result* is undoubtedly the heart of our calculator; after-all, this is precisely where the calculation takes place! The first step in this feature is to check the contents of the two operand text-edit box widgets for valid floating-point numbers. The first conditional statement accomplishes just that. It checks for some input (operand string length > 0) and valid real number (via feature *is_real*). Once we've verified for valid floating-point operands, we clear the contents of the result text-edit box widget of any previous value. Next, we interrogate the operator list to check which operator is currently active/selected based on which, we either add, subtract, multiply or divide. Following that, we simply set the result text-edit widget with the newly computed value. Before we conclude this section, check the divide operation; it's a little involved. Can you make out what it does? See what happens when you attempt to divide by zero? Hint: This dreaded dialog box will appear:

In case you've not realized it yet, you've just finished creating your first EiffelVision 2 application! The good news is that we now have a complete working calculator, BUT, I really hate to hastily conclude a tutorial with such a trivial ending. We didn't have a chance to really crank out our gray matter with EiffelVision 2 ☺. To pique your interest, let's add something more to our calculator; just a few finishing touches that'll spice it up. Let's add a simple *text-editor* to our calculator; something similar to notepad. We would like to open, save and edit the contents of a text file. A really cool feature to add is the ability to change the font displayed for the text file contents. This last feature is something that's easily implemented in windows. The Windows OS comes with a default system wide rich-text edit window class that can be created using the windows API. This window class is distributed in file **Riched32.dll**, usually found in your windows system sub-directory. Unfortunately, I don't know of any easy way to edit rich-text-format files in UNIX. So, for the benefit of both type of audiences, I'll provide a version of the calculator that'll incorporate rich-text editing capabilities and another one that'll only cater to pure text files with NO font-formatting capabilities. The changes made to the **CALCULATOR_WINDOW** class will result in the following visual representations of the calculator window:

Rich-text version



Simple-text version



The rich-text version allows you to open, edit and save rich-text files. In addition, the editor allows you to highlight portions of the text and apply font-formatting only to the selected piece of text. Although color may also be specified for each character in the rich-text version, we skipped that implementation deliberately to keep the example easy to follow. You might wonder how I managed to get that fancy blue, teal and red colors for the sample text in the snapshot above? ☺ Hint: Try using *WordPad* and Ctrl-X, Ctrl-C and Ctrl-P on the rich-text edit control! The rich-text version of the calculator will only work on the Windows platform. The simple-text version allows you to only open and save pure text files with NO text-formatting capabilities; this version will work on any platform.

This is the source listing for the version of the calculator with rich-text formatting capabilities.

```
indexing
        description: "Calculator window class"
        author: "Husein Ali"

class
        CALCULATOR_WINDOW

inherit
        EV_TITLED_WINDOW
                redefine
                        initialize
                end

create
        default_create

feature -- Initialization

        initialize is
                        -- Build the interface for this window.
                do
                        Precursor {EV_TITLED_WINDOW}

                        -- Create and add the menu bar.
                        build_menu_bar
                        set_menu_bar (standard_menu_bar)

                        -- Create the operator/operand primitives
                        build_primitives

                        -- Execute `close_window' when the user clicks
```

```
                        -- on the cross in the title bar.
                        close_request_actions.extend (agent close_window)

                        -- Set the title of the window
                        set_title (window_title)

                        -- Set the initial size of the window
                        set_size (window_width, window_height)

                        -- Prevent user from resizing calculator
                        disable_user_resize
                end

feature {NONE} -- Implementation events

        close_window is
                        -- The user wants to close the window
                do
                        (create {EV_ENVIRONMENT}).application.destroy
                end

        show_about_box is
                        -- Display an about box.
                local
                        about_dialog: EV_INFORMATION_DIALOG
                do
                        create about_dialog.make_with_text ("EiffelVision2 Calculator
                                                            Example.%NBy Husein Ali.")
                        about_dialog.set_title ("ABOUT CALCULATOR")
                        about_dialog.show_modal_to_window (Current)
                end

feature {NONE} -- Menu implementation

        standard_menu_bar: EV_MENU_BAR
                        -- Standard menu bar for this window.

        file_menu: EV_MENU
                        -- "File" menu for this window (contains Exit...)
        help_menu: EV_MENU
                        -- "Help" menu for this window (contains About...)

        build_menu_bar is
                        -- Create and populate standard_menu_bar.
                require
                        menu_bar_not_yet_created: standard_menu_bar = void
                local
                        menu_item: EV_MENU_ITEM
                do
                        -- Create the menu bar.
                        create standard_menu_bar

                        -- Add the "File" menu
                        create file_menu.make_with_text ("&File")
                        create menu_item.make_with_text ("E&xit")
                        file_menu.extend (menu_item)
                        standard_menu_bar.extend (file_menu)

                        -- Close application when 'Exit' menu-item selected.
                        menu_item.select_actions.extend (agent close_window)

                        -- Add the "Help" menu
                        create help_menu.make_with_text ("&Help")
                        create menu_item.make_with_text ("&About")
                        help_menu.extend (menu_item)
```

```
                        standard_menu_bar.extend (help_menu)

                        -- Display ABOUT box when 'About' menu-item selected.
                        menu_item.select_actions.extend (agent show_about_box)
                ensure
                        menu_bar_created: standard_menu_bar /= void and then
                                        not standard_menu_bar.is_empty
                end

feature {NONE} -- Misc primitives implementation

        enclosing_box: EV_FIXED
                        -- Invisible Primitives Container

        first_operand: EV_TEXT_FIELD
        second_operand: EV_TEXT_FIELD
                        -- Operator primitives

        result_text: EV_TEXT_FIELD
                        -- Calculation result text box.

        operator: EV_COMBO_BOX
                        -- Operator combo-box primitive

        calculate_button: EV_BUTTON

        rich_text_edit: WEL_RICH_EDIT

        open_button: EV_BUTTON
        save_button: EV_BUTTON
        font_button: EV_BUTTON

        calculate_result is
                        -- Calculate the result after the user clicks
                        -- on the 'Calculate --->' button.
                local
                        value: REAL
                        error_dialog: EV_INFORMATION_DIALOG
                do
                        -- Make sure the operands are valid real numbers
                        if (first_operand.text_length > 0) and (first_operand.text.is_real)
                                                        and
                           (second_operand.text_length > 0) and (second_operand.text.is_real)
                        then
                                -- Clear result edit text box contents.
                                result_text.remove_text

                                -- Calculate :-)
                                if operator.selected_item.text.is_equal (" +") then

                                        value := first_operand.text.to_real +
                                                second_operand.text.to_real

                                elseif operator.selected_item.text.is_equal (" -") then

                                        value := first_operand.text.to_real –
                                                second_operand.text.to_real

                                elseif operator.selected_item.text.is_equal (" *") then

                                        value := first_operand.text.to_real *
                                                second_operand.text.to_real
                                elseif operator.selected_item.text.is_equal (" /") then
                                        -- Deal with Divide By Zero error.
                                    if second_operand.text.to_real = 0 then
```

```
                                create error_dialog.make_with_text ("Divide By Zero!");
                                error_dialog.set_title ("Error");
                                error_dialog.set_pixmap(default_pixmaps.error_pixmap);
                                error_dialog.show_modal_to_window (Current)

                            else
                                value := first_operand.text.to_real /
                                        second_operand.text.to_real
                            end

                        end
                        -- Set calculated result value
                        result_text.set_text (value.out)
                    end
            end

    open_file is
                    -- Open a rich-text file
            local
                    file_open_dialog: EV_FILE_OPEN_DIALOG
                    f: RAW_FILE
            do
                    create file_open_dialog
                    file_open_dialog.show_modal_to_window (Current)

                    if file_open_dialog.selected_button.is_equal((create
                                                {EV_DIALOG_CONSTANTS}).ev_ok) then

                            create f.make_open_read_write (file_open_dialog.file_name)
                            rich_text_edit.load_rtf_file (f)
                    end
            end

    save_file is
                    -- Save a rich-text file
            local
                    file_save_dialog: EV_FILE_SAVE_DIALOG
                    f: RAW_FILE
            do
                    create file_save_dialog
                    file_save_dialog.show_modal_to_window (Current)

                    if file_save_dialog.selected_button.is_equal((create
                                                {EV_DIALOG_CONSTANTS}).ev_ok) then

                            create f.make_open_write (file_save_dialog.file_name)
                            rich_text_edit.save_rtf_file (f)
                    end
            end

    change_font is
                    -- Change current font type/size
            local
                    font_dialog: EV_FONT_DIALOG
                    font_format: WEL_CHARACTER_FORMAT
                    font: EV_FONT
                    font_const: EV_FONT_CONSTANTS
            do
                    create font_dialog
                    font_dialog.show_modal_to_window (Current)

                    if font_dialog.selected_button.is_equal((create
                                                {EV_DIALOG_CONSTANTS}).ev_ok) then
                            font := font_dialog.font
```

```
                        create font_format.make
                        font_format.set_face_name (font.name)
                        font_format.set_height (font.height)
                        create font_const

                        if font.weight = font_const.weight_regular then
                                font_format.unset_bold
                        elseif font.weight = font_const.weight_bold then
                                font_format.set_bold
                        end

                        if font.shape = font_const.shape_regular then
                                font_format.unset_italic
                        elseif font.shape = font_const.shape_italic then
                                font_format.set_italic
                        end

                        rich_text_edit.set_character_format_selection (font_format)
                end
        end

build_primitives is
                -- Create the operand/operator primitives
        require
                enclosing_box_not_yet_created: enclosing_box = void
                first_operand_not_yet_created: first_operand = void
                second_operand_not_yet_created: second_operand = void
                result_text_not_yet_created: result_text = void
                operator_not_yet_created: operator = void
                calculate_btn_not_yet_created: calculate_btn = void
        local
                parent_window: WEL_WINDOW
        do
                -- Avoid flicker on some platforms
                lock_update

                -- Cover entire window area with a primitive container.
                create enclosing_box
                extend (enclosing_box)

                -- Add first operator text edit box primitive
                create first_operand
                first_operand.set_minimum_width (138)
                enclosing_box.extend (first_operand)

                -- Add operator combo-box primitive
                create operator
                operator.set_minimum_width (40)
                operator.extend (create {EV_LIST_ITEM}.make_with_text (" +"))
                operator.extend (create {EV_LIST_ITEM}.make_with_text (" -"))
                operator.extend (create {EV_LIST_ITEM}.make_with_text (" *"))
                operator.extend (create {EV_LIST_ITEM}.make_with_text (" /"))
                operator.disable_edit
                enclosing_box.extend (operator)
                enclosing_box.set_item_x_position (operator, 149)

                -- Add second operator text edit box primitive
                create second_operand
                second_operand.set_minimum_width (138)
                enclosing_box.extend (second_operand)
                enclosing_box.set_item_x_position (second_operand, 200)

                -- Add 'calculate' button primitive
                create calculate_btn.make_with_text ("Calculate --->")
                calculate_btn.set_minimum_width (86)
```

```
                        calculate_btn.select_actions.extend (agent calculate_result)
                        enclosing_box.extend (calculate_btn)
                        enclosing_box.set_item_x_position (calculate_btn, 126)
                        enclosing_box.set_item_y_position (calculate_btn, 30)

                        -- Add result text edit box primitive
                        create result_text
                        result_text.set_minimum_width (127)
                        result_text.disable_edit
                        enclosing_box.extend (result_text)
                        enclosing_box.set_item_x_position (result_text, 105)
                        enclosing_box.set_item_y_position (result_text, 60)

                        -- Add rich-text edit control
                        parent_window ?= enclosing_box.implementation
                        create rich_text_edit.make (parent_window, "", 10, 90, 320, 170, - 1)

                        -- Add the rest of the buttons
                        create open_button.make_with_text ("Open...")
                        open_button.set_minimum_width (105)
                        enclosing_box.extend (open_button)
                        enclosing_box.set_item_x_position (open_button, 10)
                        enclosing_box.set_item_y_position (open_button, 265)
                        open_button.select_actions.extend (agent open_file)

                        create save_button.make_with_text ("Save...")
                        save_button.set_minimum_width (106)
                        enclosing_box.extend (save_button)
                        enclosing_box.set_item_x_position (save_button, 117)
                        enclosing_box.set_item_y_position (save_button, 265)
                        save_button.select_actions.extend (agent save_file)

                        create font_button.make_with_text ("Font...")
                        font_button.set_minimum_width (105)
                        enclosing_box.extend (font_button)
                        enclosing_box.set_item_x_position (font_button, 225)
                        enclosing_box.set_item_y_position (font_button, 265)
                        font_button.select_actions.extend (agent change_font)

                        -- Allow screen refresh on some platforms
                        unlock_update
                ensure
                        enclosing_box_created: enclosing_box /= void
                        first_operand_created: first_operand /= void
                        second_operand_created: second_operand /= void
                        result_text_created: result_text /= void
                        operator_created: operator /= void and then operator.count = 4
                        calculate_btn_created: calculate_btn /= void
                end

feature {NONE} -- Implementation / Constants

        Window_title: STRING is "CALCULATOR"
                        -- Title of the window.

        Window_width: INTEGER is 350
                        -- Initial width for this window.

        Window_height: INTEGER is 335
                        -- Initial height for this window.

end -- class CALCULATOR_WINDOW
```

If you try to compile this, you'll get an error. Don't panic yet. Just read on. The rich-text version of the calculator revolves around feature *rich_text_edit*: *WEL_RICH_EDIT*. Oooops, did we read that right?

WEL_RICH… What is a **WEL** class doing here??? Didn't we mention that all classes in EiffelVision 2 begin with the *EV_* prefix, so why are we using the WEL class library here? Remember, some time ago (first page of this tutorial!), we mentioned that "at the lower level of EiffelVision 2, platform specific libraries like WEL (Windows Eiffel Library) and GEL (GTK Eiffel Library for Unix) cover the graphical mechanisms of Microsoft Windows and X-Windows"…… Unfortunately, there's NO primitive in the EiffelVision 2 library that allows editing rich-text, so we resorted to using **WEL_RICH_EDIT**, a WEL class library that encapsulates a rich-text edit window. Fortunately, the esteemed engineers at ISE had fully anticipated that the EiffelVision 2 library (or any other class library for that matter) can never be too generic to cover all possible implementation scenarios so they made sure the library is open-ended to account for such cases. The only problem with resorting to a platform specific implementation library is that your code will end up working for only ONE platform, contrary to what the EiffelVision 2 library was supposed to relieve you of in the first place ☹. Using an underlying implementation library like WEL directly has its own set of problems e.g. when it comes to coordinating/binding it with EiffelVision 2 based code (you'll cast your eyes across such an example in a minute!). This is not the right place to review the WEL class library; please consult my previous tutorials for details on how to use the WEL class library. The rest of the tutorial will assume you know a little about how the WEL class library functions. Sorry folks, but there really is no easy way around this; that is, if you choose to follow the version of the calculator with rich-text editing capabilities. If you think this is getting a little heavy, skip this section and try the other simpler version of the calculator first. BTW, this **IS** the place where our gray matter will be cranked fully as promised ☺

So, exactly what problem are we facing by using a WEL class directly? Actually, I exaggerated a little more than I should have back there! In our case, there's really only ONE problem and unfortunately, it's an issue that the esteemed engineers at ISE are responsible for. Starting to feel relieved already ☺ See if this explanation makes any sense. A little while back, we said that the EiffelVision 2 library exhibits a common interface for all its classes across all platforms via the **EV_ANY** class and that platform implementation specific delegate classes are managed by **EV_ANY** transparently behind the scenes, but, that was something we really didn't care about, at-least not until now. Pretty much every EiffelVision 2 class has a corresponding implementation class(es) (WEL based on the Windows platform) that's mapped and managed automatically by the library. Simply stated, an arbitrary **EV_X** EiffelVision 2 class usually has a corresponding **EV_X_IMP** platform specific implementation class. This **EV_X_IMP** class makes use of one or more platform specific classes (either via inheritance and/or client-supplier relationship). In our case, the platform specific classes are all WEL based. The **EV_X_IMP** object is accessed via the *implementation* feature of class **EV_X**. As it turns out, we'll need to directly access this underlying implementation WEL object to be able to use the *rich_text_edit* control correctly. The gist of the problem is such:

The *make* feature in class **WEL_RICH_EDIT** requires a *parent* window object of class type **WEL_WINDOW**. This is the window that'll *contain* the rich-text edit widget. Ahhhhh, speaking of *containing*, remember our magical main window container *enclosing_box* of class type **EV_FIXED**? (If this is rusty, go back and re-read the relevant section). The implementation class **EV_FIXED_IMP** for class **EV_FIXED** descends from class **WEL_CONTROL_WINDOW** which in turn descends from class **WEL_WINDOW**. Realizing this, we try to access the *implementation* feature of the **EV_FIXED** container object (feature *enclosing_box*) inside our **CALCULATOR_WINDOW** class. This would allow us to access the **EV_FIXED_IMP** object (type conformant to **WEL_WINDOW**) that we could happily pass on to the *make* feature of class **WEL_RICH_EDIT**. Take a look at the following statements in feature *build_primitives*:

```
parent_window ?= enclosing_box.implementation
rich_text_edit.make (parent_window, "", 10, 90, 320, 170, - 1)
```

If you try to compile the latest version of the source listing, you'll get this error:

Error code: VUEX(2)
Error: feature of qualified call is not available to client class.
What do do: make sure feature after dot is exported to caller.

So why can't we access feature *implementation* of class **EV_FIXED** inside the **CALCULATOR_WINDOW** class? Go ahead and check the *implementation* feature export clause in class **EV_FIXED**; this is what it reads:

```
feature {EV_ANY_I} -- Implementation

        implementation: EV_FIXED_I
                -- Responsible for interaction with native graphics toolkit.

Feature {NONE} -- Implementation

        create_implementation is
                        -- See `{EV_ANY}.create_implementation'.
                do
                        create {EV_FIXED_IMP} implementation.make (Current)
                end
```

Bingo ☺. Clearly, the *implementation* feature is only exported to class **EV_ANY_I**. In other words, the feature that exposes the underlying native implementation object (WEL based in our case) can't be accessed from class **CALCULATOR_WINDOW** (or any class except **EV_ANY** for that matter). The obvious solution is to modify the export clause for feature *implementation* and manually ADD **CALCULATOR_WINDOW**. Once you make this change, the latest listing will compile and run correctly. Is this the right approach though? Is there another clean way of accessing the underlying native implementation object in EiffelVision 2? Specifically, how else do we access the **WEL_WINDOW** implementation object that corresponds to the **EV_FIXED** object? The simple answer is a NO, there's no other alternative. Currently, there's no way around this hack. It seems the engineers at ISE overlooked this problem ☹. I've brought this matter to their attention and fortunately, they are reviewing this access method and hope to improve upon it for future versions of EiffelVision 2.

The only changes made in this implementation are adding features *open_button*: *EV_BUTTON*, *save_button*: *EV_BUTTON*, *font_button*: *EV_BUTTON*, *open_file*, *save_file*, *change_font* and updating feature *build_primitives*. Feature *build_primitives*, adds a local *parent_window*: *WEL_WINDOW* variable, the purpose of which was already discussed. Augmenting to the previous *buid_primitive* code, the statements

```
        parent_window ?= enclosing_box.implementation
        rich_text_edit.make (parent_window, "", 10, 90, 320, 170, - 1)
```

were also just discussed. These lines create a new rich-text multi-line edit widget and positions it at the specified coordinates. Remember, the rich-text edit control is enclosed by the main window container *enclosing_box*. The next set of lines create three new push buttons captioned '*Open…*', '*Save…*' and '*Font…*' for the file open/save and font-formatting operations. These buttons are positioned immediately below the rich-text edit widget. The click event-handlers for these buttons are tied with features *open_file*, *save_file* and *change_font*. Feature *open_file* makes use of class **EV_FILE_OPEN_DIALOG** to display a file select dialog box. After retrieving the selected file name from the user, the feature opens the file using class **RAW_FILE** requesting read/write permissions and then delegates the task of loading the rich-text edit control with the file contents to feature *load_rtf_file* already implemented in class **WEL_RICH_EDIT**. The *save_file* feature works along the same pattern as *open_file* except here, class **EV_FILE_SAVE_DIALOG** is used instead and feature *save_rtf_file* is used in place of *load_rtf_file*. Feature *change_font* is relatively a bit more involved. A select font dialog box (of class type **EV_FONT_DIALOG**) is first displayed for the user to select a font and set any attributes like size, weight etc. Next, the user selected font is then saved in a local *font* variable of class type **EV_FONT**. An instance of class type **WEL_CHARACTER_FORMAT** by the name of *font_format* is then created. This class contains information about character formatting in a rich-edit control. Following that, the font face-name, height, weight and shape attributes selected by the user are then applied to *font_format*. Class **EV_FONT_CONSTANTS** is simply used to enhance code readability. We could have very well materialized strange looking constants out of nowhere and everyone would be totally lost. But then, that's not what we want ☹ Besides, it's an ugly coding practice that we should all avoid (not always ☺). The final statement calls feature *set_character_format_selection* and passes the font-formatting attributes contained in *font_format* as a parameter. This sets the **currently selected text** with the font attributes contained in *font_format*.

This concludes the rich-text version of the calculator. The following final segment of this tutorial will discuss the simpler version of the calculator that'll work both on the Windows as well as the Unix platform. You may conclude right here with the tutorial if you're using the Windows platform ☺. If you're on the Unix platform, or simply wish to garnish your EiffelVision 2 repertoire, then please read on!

The following is the source listing for the version of the calculator with simple text-editing capabilities:

```
indexing
        description: "Calculator window class"
        author: "Husein Ali"

class
        CALCULATOR_WINDOW

inherit
        EV_TITLED_WINDOW
                redefine
                        initialize
                end

create
        default_create

feature -- Initialization

        initialize is
                        -- Build the interface for this window.
                do
                        Precursor {EV_TITLED_WINDOW}

                        -- Create and add the menu bar.
                        build_menu_bar
                        set_menu_bar (standard_menu_bar)

                        -- Create the operator/operand primitives
                        build_primitives

                        -- Execute `close_window' when the user clicks
                        -- on the cross in the title bar.
                        close_request_actions.extend (agent close_window)

                        -- Set the title of the window
                        set_title (window_title)

                        -- Set the initial size of the window
                        set_size (window_width, window_height)

                        -- Prevent user from resizing calculator
                        disable_user_resize
                end

feature {NONE} -- Implementation events

        close_window is
                        -- The user wants to close the window
                do
                        (create {EV_ENVIRONMENT}).application.destroy
                end

        show_about_box is
                        -- Display an about box.
                local
                        about_dialog: EV_INFORMATION_DIALOG
                do
                        create about_dialog.make_with_text ("EiffelVision2 Calculator
                                                            Example.%NBy Husein Ali.")
                        about_dialog.set_title ("ABOUT CALCULATOR")
                        about_dialog.show_modal_to_window (Current)
                end
```

```
feature {NONE} -- Menu implementation

        standard_menu_bar: EV_MENU_BAR
                        -- Standard menu bar for this window.

        file_menu: EV_MENU
                        -- "File" menu for this window (contains Exit...)
        help_menu: EV_MENU
                        -- "Help" menu for this window (contains About...)

        build_menu_bar is
                        -- Create and populate standard_menu_bar.
                require
                        menu_bar_not_yet_created: standard_menu_bar = void
                local
                        menu_item: EV_MENU_ITEM
                do
                        -- Create the menu bar.
                        create standard_menu_bar


                        -- Add the "File" menu
                        create file_menu.make_with_text ("&File")
                        create menu_item.make_with_text ("E&xit")
                        file_menu.extend (menu_item)
                        standard_menu_bar.extend (file_menu)

                        -- Close application when 'Exit' menu-item selected.
                        menu_item.select_actions.extend (agent close_window)

                        -- Add the "Help" menu
                        create help_menu.make_with_text ("&Help")
                        create menu_item.make_with_text ("&About")
                        help_menu.extend (menu_item)
                        standard_menu_bar.extend (help_menu)

                        -- Display ABOUT box when 'About' menu-item selected.
                        menu_item.select_actions.extend (agent show_about_box)
                ensure
                        menu_bar_created: standard_menu_bar /= void and then
                                          not standard_menu_bar.is_empty
                end
feature {NONE} -- Misc primitives implementation

        enclosing_box: EV_FIXED
                        -- Invisible Primitives Container

        first_operand: EV_TEXT_FIELD

        second_operand: EV_TEXT_FIELD
                        -- Operator primitives

        result_text: EV_TEXT_FIELD
                        -- Calculation result text box.

        operator: EV_COMBO_BOX
                        -- Operator combo-box primitive

        calculate_button: EV_BUTTON

        text_edit: EV_TEXT

        open_button: EV_BUTTON

        save_button: EV_BUTTON
```

```
	calculate_result is
			-- Calculate the result after the user clicks
			-- on the 'Calculate --->' button.
		local
			value: REAL
			error_dialog: EV_INFORMATION_DIALOG
		do
			-- Make sure the operands are valid real numbers
			if (first_operand.text_length > 0) and (first_operand.text.is_real)
                                                and
			   (second_operand.text_length > 0) and (second_operand.text.is_real)
			then
				-- Clear result edit text box contents.
				result_text.remove_text

				-- Calculate :-)
				if operator.selected_item.text.is_equal (" +") then

					value := first_operand.text.to_real +
						second_operand.text.to_real

				elseif operator.selected_item.text.is_equal (" -") then

					value := first_operand.text.to_real –
						second_operand.text.to_real

				elseif operator.selected_item.text.is_equal (" *") then

					value := first_operand.text.to_real *
						second_operand.text.to_real
				elseif operator.selected_item.text.is_equal (" /") then
					-- Deal with Divide By Zero error.
				    if second_operand.text.to_real = 0 then

					create error_dialog.make_with_text ("Divide By Zero!");
					error_dialog.set_title ("Error");
					error_dialog.set_pixmap(default_pixmaps.error_pixmap);
					error_dialog.show_modal_to_window (Current)

				    else
					value := first_operand.text.to_real /
						second_operand.text.to_real
				    end

				end
				-- Set calculated result value
				result_text.set_text (value.out)
			end
		end

	open_file is
			-- Open a text file
		local
			file_open_dialog: EV_FILE_OPEN_DIALOG
			f: PLAIN_TEXT_FILE
		do
			create file_open_dialog
			file_open_dialog.show_modal_to_window (Current)

			if file_open_dialog.selected_button.is_equal ((create
                                                {EV_DIALOG_CONSTANTS}).ev_ok) then
				create f.make_open_read_write (file_open_dialog.file_name)

				-- Clear previous contents of edit text-box.
				text_edit.remove_text
```

```
                        -- Read file contents and update edit text-box
                        f.read_stream (f.count)
                        text_edit.set_text (f.last_string)
                        f.close
                end
        end

    save_file is
                -- Save a text file
        local
                file_save_dialog: EV_FILE_SAVE_DIALOG
                f: PLAIN_TEXT_FILE
        do
                create file_save_dialog
                file_save_dialog.show_modal_to_window (Current)

                if file_save_dialog.selected_button.is_equal ((create
                                                {EV_DIALOG_CONSTANTS}).ev_ok) then
                        create f.make_open_write (file_save_dialog.file_name)

                        -- Write to file contents of edit text-box
                        f.put_string (text_edit.text)
                        f.flush
                        f.close
                end
        end

    build_primitives is
                -- Create the operand/operator primitives
        require
                enclosing_box_not_yet_created: enclosing_box = void
                first_operand_not_yet_created: first_operand = void
                second_operand_not_yet_created: second_operand = void
                result_text_not_yet_created: result_text = void
                operator_not_yet_created: operator = void
                calculate_btn_not_yet_created: calculate_btn = void
        do
                -- Avoid flicker on some platforms
                lock_update

                -- Cover entire window area with a primitive container.
                create enclosing_box
                extend (enclosing_box)

                -- Add first operator text edit box primitive
                create first_operand
                first_operand.set_minimum_width (138)
                enclosing_box.extend (first_operand)

                -- Add operator combo-box primitive
                create operator
                operator.set_minimum_width (40)
                operator.extend (create {EV_LIST_ITEM}.make_with_text (" +"))
                operator.extend (create {EV_LIST_ITEM}.make_with_text (" -"))
                operator.extend (create {EV_LIST_ITEM}.make_with_text (" *"))
                operator.extend (create {EV_LIST_ITEM}.make_with_text (" /"))
                operator.disable_edit
                enclosing_box.extend (operator)
                enclosing_box.set_item_x_position (operator, 149)

                -- Add second operator text edit box primitive
                create second_operand
                second_operand.set_minimum_width (138)
                enclosing_box.extend (second_operand)
                enclosing_box.set_item_x_position (second_operand, 200)
```

```
                        -- Add 'calculate' button primitive
                        create calculate_btn.make_with_text ("Calculate --->")
                        calculate_btn.set_minimum_width (86)
                        calculate_btn.select_actions.extend (agent calculate_result)
                        enclosing_box.extend (calculate_btn)
                        enclosing_box.set_item_x_position (calculate_btn, 126)
                        enclosing_box.set_item_y_position (calculate_btn, 30)

                        -- Add result text edit box primitive
                        create result_text
                        result_text.set_minimum_width (127)
                        result_text.disable_edit
                        enclosing_box.extend (result_text)
                        enclosing_box.set_item_x_position (result_text, 105)
                        enclosing_box.set_item_y_position (result_text, 60)

                        -- Add the multi-line simple text-edit control
                        create text_edit
                        text_edit.set_minimum_width (320)
                        text_edit.set_minimum_height (170)
                        enclosing_box.extend (text_edit)
                        enclosing_box.set_item_x_position (text_edit, 10)
                        enclosing_box.set_item_y_position (text_edit, 90)

                        -- Add the rest of the buttons
                        create open_button.make_with_text ("Open...")
                        open_button.set_minimum_width (105)
                        enclosing_box.extend (open_button)
                        enclosing_box.set_item_x_position (open_button, 62)
                        enclosing_box.set_item_y_position (open_button, 265)
                        open_button.select_actions.extend (agent open_file)

                        create save_button.make_with_text ("Save...")
                        save_button.set_minimum_width (106)
                        enclosing_box.extend (save_button)
                        enclosing_box.set_item_x_position (save_button, 168)
                        enclosing_box.set_item_y_position (save_button, 265)
                        save_button.select_actions.extend (agent save_file)

                        -- Allow screen refresh on some platforms
                        unlock_update
                ensure
                        enclosing_box_created: enclosing_box /= void
                        first_operand_created: first_operand /= void
                        second_operand_created: second_operand /= void
                        result_text_created: result_text /= void
                        operator_created: operator /= void and then operator.count = 4
                        calculate_btn_created: calculate_btn /= void
                end

feature {NONE} -- Implementation / Constants

        Window_title: STRING is "CALCULATOR"
                        -- Title of the window.

        Window_width: INTEGER is 350
                        -- Initial width for this window.

        Window_height: INTEGER is 335
                        -- Initial height for this window.

end -- class CALCULATOR_WINDOW
```

This version of the calculator is much simpler than the previous one. The primitive we'll use is a multi-line text edit control class called **EV_TEXT**. We add features *text_edit*: *EV_TEXT*, *open_button*: *EV_BUTTON* and *save_button*: *EV_BUTTON*. These are the exact analogs of the rich-text version of the calculator, with the exception that we no longer have a *font_button* feature because it's not applicable here (the reason for which will be clear shortly). Unlike the rich-text analog, the **EV_TEXT** class unfortunately doesn't expose features to load/save from a text file or change the font displayed for the text. In this case, we do have an EiffelVision 2 class here that we can use, however, it's quite primitive (no pun intended!). So, does that mean we'll have to resort to some WEL based class in favor of an EiffelVision 2 class again, as encountered in the rich-text version of the calculator? Better yet, what about accessing the underlying *implementation* WEL object for **EV_TEXT**? Fortunately, the *implementation* WEL object working underneath **EV_TEXT** (which happens to be **WEL_MULTIPLE_LINE_EDIT**) provides a plethora of needed functionality. BUT (a really BIG one), isn't this second version of the calculator supposed to work on ANY platform. This clearly excludes using both the above possibilities, so what do we do? (Ouch, we're a duck stuck in the middle of the ocean with no where to go!). Well, the only obvious way out of this is to do what most of us do when we can't find something off the shelf…….either craft it ourselves (unless you're a magician) or simply ignore it altogether ☺. We'll take a middle stance; we'll implement the task of loading/saving from a text-file but ignore the font-formattting options. You might have noticed that this version of the calculator doesn't have a *font* button in the snapshots above (remember, we don't have a *font_button* feature!); you now know why. Unfortunately, there's no way around this, at-least not without running into platform dependency issues. Hopefully, the next release of EiffelVision 3 will have all these extra bells and whistles built-in so we don't have to remedy our way out with platform specific code. ISE, are you listening??

In any case, let's wrap this tutorial quickly. The current implementation of **CALCULATOR_WINDOW** augments features *open_file* and *save_file*. These are the exact analogs of the rich-text version; we'll quickly skim through them again though for the benefit of those Unix savvy conglomerate that choose to ignore the other (superior!) version of the calculator (sincerely no offense ☺). Feature *open_file* starts off by making use of class **EV_FILE_OPEN_DIALOG** to display a file open dialog box. The scaring looking conditional

```
if file_open_dialog.selected_button.is_equal ((create {EV_DIALOG_CONSTANTS}).ev_ok) then
```

checks whether the user clicked on the *OK* button. If so, we open the user selected file via class **PLAIN_TEXT_FILE** followed by a call to feature *remove_text* to clear the previous contents of the text-edit box. Next, we use feature *read_stream* to read in the content of the whole file into one single string (feature *last_string*). Finally, feature *set_text* is used to set the contents of the text-edit box with the newly read string and the opened file then closed. Feature *save_file* works along the same lines as *open_file* but in reverse. We first request the name of the file the user wishes to save to (via the **EV_FILE_SAVE_FILE_DIALOG** class dialog box). Next, the scary conditional is revisited again to ensure the *OK* button was clicked. We then use feature *make_open_write* (in class **PLAIN_TEXT_FILE**) to either open the user selected file (if it already exists) or create a new one (if it doesn't exist). In either case, the next set of lines write out the contents of the text-edit box to the file. Beware that in the former case, the current contents of the existing file are completely *wiped* and *replaced* with the contents of the text-edit box. This is followed by a file close statement which ends the whole process of saving to a file. Feature *build_primitives* is updated once again (I promise the last time ☺) to add the text-edit box, open button and save button widgets on the main window. The **click** event-handlers for the two buttons are tied with features *open_file* and *save_file*.

Whew, this finally concludes the tutorial! Go ahead and tweak the calculator; try to add or make changes for fun! I realize the learning process has been a long one and probably very involved too. Hey, I did promise to crank out that grey matter, right ☺. The level of detail might have been intimidating but I hope it was an approachable and novel introduction. Although the design of EiffelVision 2 is solid, flexible and clean for the most part, there are a few quirks here and there but then, I know of no comparable, multi-platform, Eiffel based GUI class library out there that can beat it ☺