

OS/2 GUI Tools: Enfin and Easel

Two of the most interesting (and expensive) GUI builders on the market today, Easel and Enfin offer similar capabilities via radically different languages. The elder statesman of the two, Easel first hit the market in 1984 as a DOS product, long before the current OOP craze, and has since been continually developed with an eye to backward compatibility. Enfin was developed from the ground up as an object-oriented language (based on Smalltalk) by Enfin Corp. and hit the market in 1989. Enfin Corp. was bought out by Easel Corp. in 1992. ¹

The Application

To get a feel for the strengths and weaknesses of each, I spec'd a small application with an eye toward exercising in some depth the main features of both products. The application I chose is one I've been thinking about for a while: a personal, total health tracking and management package. The purpose of the package is to allow an individual to enter and analyze all the data that together provides a picture of his or her physical health, a Quicken for the body.

The heart of the application is a database of personal data: meals, workouts, illnesses, treatments, weigh-ins and checkups. The theory is that by looking at the data over time people can get a better feel for how to manage their own health. Of course, no total health package would be complete without drug, anatomy and general health references - all areas where CD-ROM publishers have provided strong products, so all I provide for them is hooks to start up external applications. Figure 1 shows a simplified data model for the application.

The Easel Language

Easel tries hard to be English-like and sort of succeeds. Statements like "*make GenericList_DialogBox visible*" and "*change IngredientName_EditField text to StringVar*" are typical Easel. While this makes Easel extremely readable, there are a couple of problems with it. Take the latter example. The statements "*change text of IngredientName_EditField to StringVar*" and "*Copy StringVar to text of IngredientName_EditField*" are both made up of valid Easel constructs and mean the same thing as "*change IngredientName_EditField text to StringVar*" in English, but in Easel they don't compute. A minor gripe, admittedly, but unlike other languages, no matter how long I work with Easel, I still find myself making those kinds of mistakes.

Easel supports a number of basic data types - **string**, **integer**, **float** and **boolean**, as well as arrays of each type, and structures - regular structures and fixed-length structures suitable for use by external C-DLLs. It also supports a large number of built-in object types; dialog boxes, sense, graphical and dialog regions, fonts, image maps ... All of the standard PM controls also get their own object type; button, entry field, list box etc. Easel is an explicitly event-driven language and thus, all the usual PM suspects, window open and close, button click, as well as program start and end have their own event. Easel supports four different procedure types; **actions**, **subroutines**, **functions** and **response blocks**. **Actions**, **subroutines** and **functions** are all stand-alone pieces of code, roughly corresponding to macros, call-by-reference functions and call-by-value functions in C. Response blocks are explicitly attached in their declaration to specific events from specific objects. Whenever the event occurs, the response block executes. References to built-in object types can be either hard-wired object names or string variables containing a valid object name. Most of the power of Easel comes from combining this bit of abstraction with **response blocks**. Every possible event in the system can, but doesn't have to, have a **response block** associated with it.

Easel supports a suite of familiar control structures: if-then-else, switch, for and while. You can also get outside Easel with relative ease by invoking external executables, DDE messaging or by calling external DLLs. The flow of control in an Easel program goes roughly: object creation-event generation-procedure call. The procedure

¹Dates provided by David Kitchen of Copplethorn & Bellows, Easel's PR firm.

calls in turn can create new objects which generate more events and so on. Figure 2 shows the flow in a typical Easel program.

Enfin Smalltalk

Going into Enfin Smalltalk I feel a little like the American in Paris who is startled to find that "they have a different word for everything over here!". Enfin is a Smalltalk variant and knowing something about Smalltalk would give you a big leg up on getting going with it. Though neither a Smalltalk V nor Smalltalk-80 variant, Enfin promises to converge on ANSI Smalltalk whenever that arrives. Smalltalk is much too big a subject to tackle in depth here, but a short overview is in order.

Enfin Smalltalk is object-oriented (perhaps object-obsessed would be a better term) from the ground up. Things you might consider basic data types, integer and float for example, are mere subclasses of more abstract classes. There are really only four basic things to think about in Enfin: classes, methods, objects and messages. A class implements a method. An object provides an instance of a class. A message sent to that object selects a method for that object to execute. The statement:

```
System beepFrequency: 400 duration 100.
```

sends the message `beepFrequency:duration:` to the object named `System`. As an object obsessed language, Enfin is almost completely untyped. The statements:

```
X := 1.
```

```
X beepFrequency: 400 duration: 100.
```

compile even though `X` is a number. `X`'s object type isn't determined until run-time, at which point it generates an error "instance method `beepFrequency:duration:` not found in class `smallInteger`". Figure 3 shows the flow of control for a Smalltalk program.

Since Enfin Smalltalk is nothing but objects and methods, the power of the language lies in the supplied classes and the methods they support. Most of the usual basic data types (`bool`, `int`, `float` ...) are supplied and they support the usual operators. Some of our favorite control flow is available, conditional exec, loop on expression, and loop on index as well as loop over collection. An important point to remember with control structures is that they are implemented as messages with blocks of code as arguments. Thus the statement:

```
X ifTrue: ['yes' out].
```

is more properly read as *send the `ifTrue` message to `X` with `['yes' out]` as the code-block argument*. Thus, you can easily implement your own control flow by writing a method with a block argument.

User Interface

I chose to use different style interfaces for the two implementations to reflect the strengths and the 'Zen' of the tools. The Workbench implementation goes with a traditional PM/Windows style with a hierarchy of menus available from a menu bar off the main screen. The Enfin implementation strives, foreign as it is to me, to implement a drag and drop style interface. Figures 4 and 5 show the first two levels of the app in both versions. In the Enfin version, any of the icons in the top two lines can be dragged and dropped onto the Edit or New icons in order to create a new item, or edit an existing item.

This choice has important implications in Enfin. To implement the traditional, hierarchical model I would have created a **Controller**, with a single **Form** containing a menu and multiple **Subforms**. Instead, we have a **Controller** with a single **Form**, which contains only a series of **WorkplaceObjects**. These

WorkplaceObjects appear within the main form as icons and have forms attached to them. The **WorkplaceObjects** are direct substitutes for the menu items that would populate the menu bar of a hierarchical app. When one **WorkplaceObject** is dropped on another, the open method for the receiver is called. The default WorkplaceObject>>open method opens the **Form** associated with the receiving **WorkplaceObject**.

In the Easel version, you pick a type of item to work on from the menu, then, from the second level dialog, either select an existing item to edit or create a new one. Each menu item gets its own response block which gets called automatically when the user selects the item. The **response block** queries the database, formats the displayable list of items, then calls a generic list dialog box. When the user hits New or Edit from the generic list box, the **response block** written for these buttons loads the dialog named by the parameter.

The flow of the application from main menu to edit-item dialog combines many of the most powerful features of Easel: parameters, object access-by-name and item classes. All the menu items are collected into a single class (it's probably easier to think of Easel classes as arrays of objects rather than OOP classes). This class gets a single response block. Choosing one of the items in the class causes the class response block to run. Each menu item also has a unique parameter. Calling parameter within the **response block** gets the **parameter** of the menu item that triggered the response. The **response block** constructs the names of the **actions** (unique to each menu item) to call using the parameter. The unique action blocks format the data for the list box. Listing 1 shows the **primary region, menu bar, menu item, item class, and response block** code for a new **Workout** item.

Database Access

External database access is one of the key features of these packages. I chose IBM's DB2/2 single-user, version 1 for this project since it's supported by both packages. For database creation and initialization (with sample data) I wrote a simple C program.

Though previous versions of Workbench relied on 'local app' access to databases, where you had to start up a command line OS/2 shell and read and write it's standard IO, both packages now supports seamless, DLL access to a number of vendor's SQL databases. Workbench provides automated SQL database linkage through a package called DB/Assist, a separate executable. It allows you to construct and test SQL statements and link them to Easel variables. These statements are turned into Easel action blocks and inserted into the project. The Easel program calls OPEN_XXX, FETCH_XXX and CLOSE_XXX (where XXX is the name of the SQL statement) and the data from the **select** magically appears in the linked variables. From there, the coder has to write code to enter these values into the dialog box. There is no auto creation of data entry forms.

Enfin, on the other hand, provides automatic creation of data entry forms via the databases tool. Pick a database table, choose **data entry form** and voila, a pretty good first hack at the dialog box. Enfin also comes with a lightweight, internal SQL database that allows you to develop and test your application without actually running the external DB; a nice touch.

Enfin provides automatic links into the database, where you can specify a view into the database and that view is automatically updated whenever the data behind it changes. Listing 2 shows the initialization code for the listbox containing the list of workouts, where you can delete one, or choose one to edit. Two lines of code link the TabularListBox object to the database table. Two more lines link the edit field in the dialog box to the field in the record. Rows fetched from the database appear as Record objects and the fields within a Record object can be obtained by name. HEALTH never needs that kind of access. All db operations are handled by the provided classes.

External Links

In Enfin, the three external applications, Anatomy, General Health and Drug references are setup by subclassing the WorkplaceObject with our own ExecWPO class. The default WorkplaceObject behavior is to 'open'

the object associated with the icon, so our ExecWPO provides `setSessionName` and `setFileName` methods that setup the session title and executable file name and a new open method that uses the String class method `startSessionProgram` to execute the program. Listing 3 shows the new ExecWPO class. The Easel version uses a straightforward `start application` call from the menu item response block.

Source Code and the IDE

Easel Workbench keeps the source for your project in a binary .EWB file. Using an outside editor to change source involves exporting it from the IDE then importing it back in, an awkward process. The IDE does maintain a map of source to source-file (project views) so that exporting is relatively pain-free. You can include Easel source files through the use of a simple **include** statement. Easel applications compile into an Easel Binary (.EBI) file. In order to run the application, users have to have a run-time version of Easel installed. Easel run-time licenses carry a substantial price-tag.

Enfin operates directly on ASCII text .CLS files. You can edit outside the Enfin environment as long as you remember to reload a changed source file. You include Enfin Smalltalk files in your application by adding the file name to the application file. Figure 6 shows the source file organization of a three-form Enfin application. Designer, the interface source code generator very capably gets you through the look and feel phase of program building. Like most code-generators, Designer writes code its own way, and if you modify files outside the system you have to be careful where you do it. The Controller method, initialize, seems to be reserved for Designer-generated code and if you throw something in there, then modify an object through Designer, it can get lost. The best policy is to not modify Designer-generated methods but to write your own methods and simply 'call' them from the Designer-generated method. Enfin applications compile into an image (.IMG) file. Again, users need the run-time Enfin file (a single executable) but the Enfin run-time can be distributed royalty-free.

Results

The final Enfin application comes out to about 1700 lines (including whitespace) of original code. Of that, approximately 1400 is instantiation of Form objects (essentially dialog box definitions). The Form initializations are mostly uninteresting, except for the code setting up links to the database.

I created a separate Form for each 'list' of database records, with a hard-wired link to a database table simply because it was so easy using Enfin Data Manager's automatic default data entry form facility. I could, as I did in the Easel version, have used a single 'list' Form by sub-classing Form. This subclass would have to maintain variables for the table name and the edit form that follows when the user hits Edit or double-clicks an item in the listbox. The subclass open method would take the new table name, remove any existing table link then add a link to the new table and update it.

The Easel application comes out to about 2500 lines of original code. Of that, about 1400 is form initialization, 800 DB/Assist-generated SQL, and 300 is code I entered manually. This is a little deceptive, though. The only way I got away with so little manual code in Easel was by taking care in how I named the objects and actions. If I'd created a different form for each list of items (as I did in Enfin) the line count would have inflated exponentially.

Documentation

I worked with Enfin 2.90 and an early copy of the 4.0 documentation. The 2.90 documentation was riddled with grammatical errors and worse, was not current with the product itself, talking about screens and buttons that didn't exist. In 4.0, which should be in production by the time you read this, the documentation is very slick, but that said, it may not be exactly what you expect. The package supplies about 400 classes with 6000 methods. Most of these methods are VERY short, a few lines maybe. This makes the traditional, one function per page of paper documentation style clearly inappropriate. Enfin takes the programming by example approach, where

they try to give you a general idea of which classes and methods to use and it's up to you to look at the actual class code via the Browser to see how they work. A hard-copy list of classes, with a brief description of each is provided, but is actually virtually worthless.

The Easel Workbench used was version 3.00.0. The doc is complete, as far as it goes, but at least in the reference, you really have to know what you're looking for. For instance, the keyword **text** can appear in either a **change text to** or **text of** statement and if you find a reference to one, it won't point you to the other. This is a weakness of the language though, not the doc.

Conclusion

Both packages are highly capable, complete application production environments. The classes provided with Enfin, in general provide more functionality than the built-in objects of Easel. For example, **Form** fields can have validation attached, and database item attachment to dialog controls is handled automatically by the **TableLink** class, both things requiring you to write more code in Easel. The power of Enfin is somewhat muted, though, by the IDE. Though you can sub-class interface classes, when you do so, you lose the ability to modify them within Enfin Designer. When I tried to modify one of my **ExecWPO** objects through the Designer, the code generator turned it back into a **WorkplaceObject**. The choice between them comes down to whether you're object-obsessed or object-tolerant. While object-obsessed Enfin is clearly the wave of the future, Easel provides similar functionality in a more traditional package.

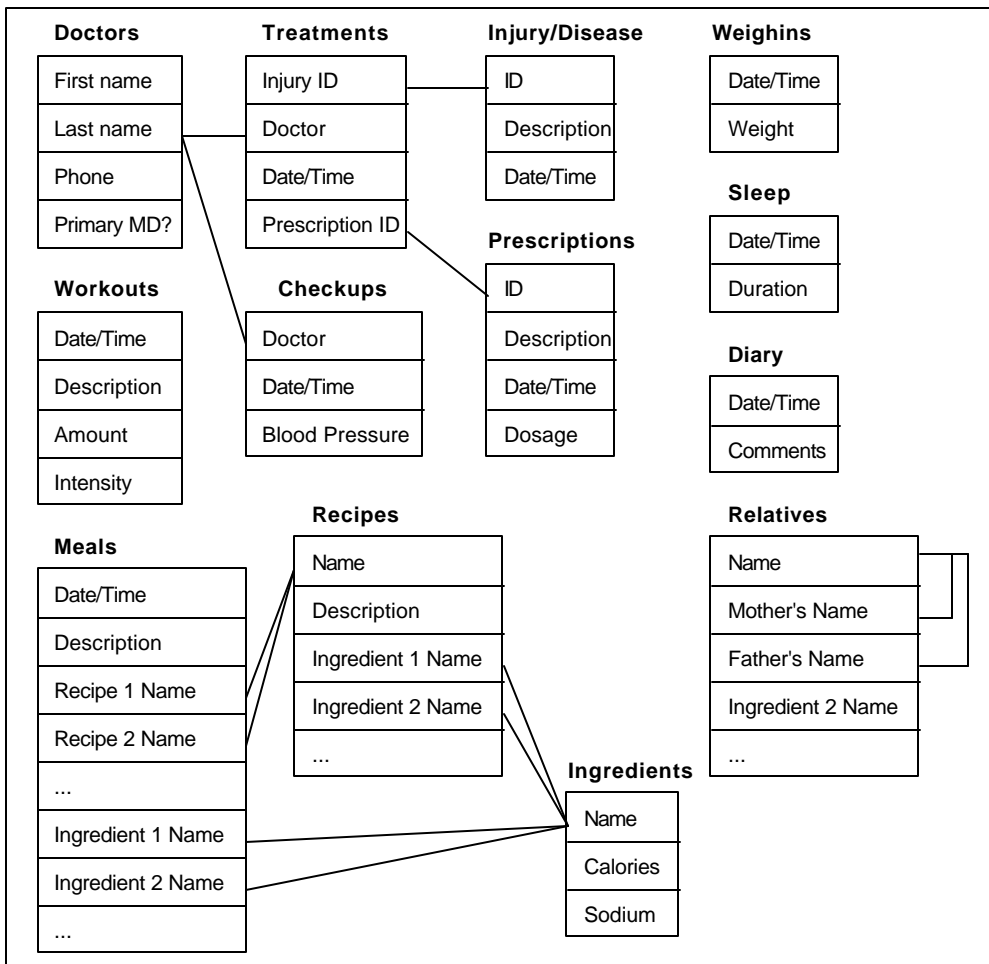


Figure 1: Simplified data model for test application.

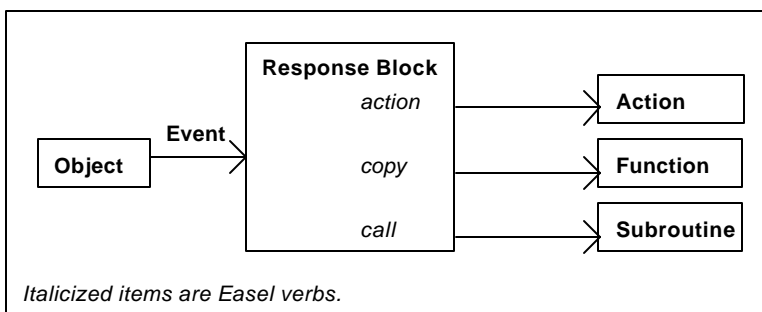


Figure 2: Control flow in an Easel program.

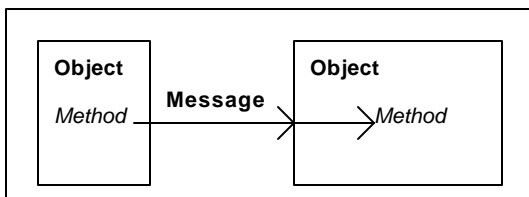


Figure 3: Control flow in an Enfin Smalltalk program

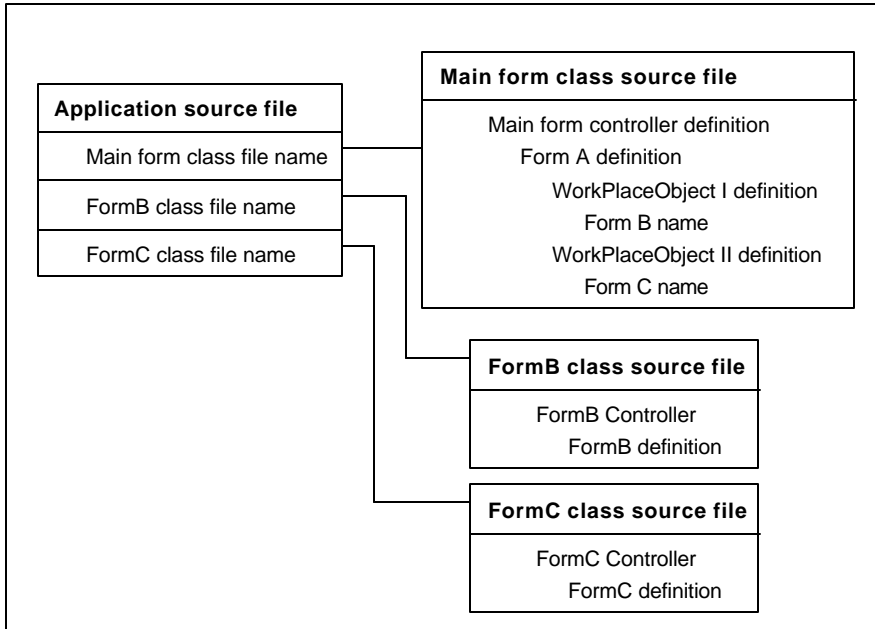


Figure 6: Source file structure of an Enfin application with 3 forms.