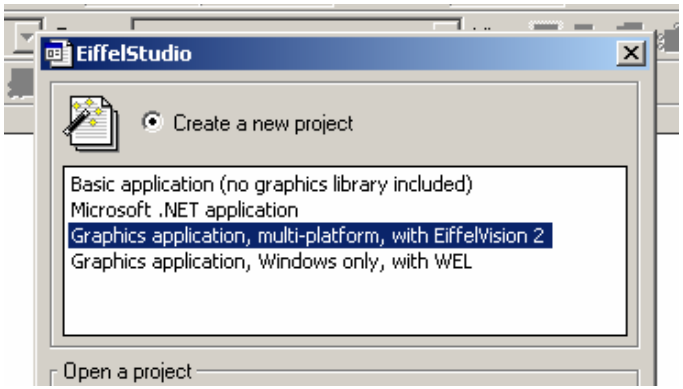
Glückliche Benutzer dank Geheimdienst

Die Vision in Eiffel

In Eiffel werden grafische Benutzeroberflächen bzw. GUIs (engl. *graphic user interface*) entweder mit WEL oder EiffelVision hergestellt. Da WEL windowsabhängig ist, wollen wir einmal nur EiffelVision weiter betrachten.

Mit EiffelVision[1] kann man Fenster, Knöpfe, Listen, Toolbars, Schieber etc. so zusammenstellen, damit man eine Benutzeroberfläche für das eigene Programm bekommt. Normalerweise entwickelt man aber zuerst nach wie vor eine Anwendung mit Klassen, Objekten etc. und testet das Ganze mit einer primitiven ROOT_CLASS, welche Informationen statt auf der Konsole mit Meldungsfenstern ausgibt. Meistens erst dann macht man ein GUI nach den Vorstellungen, die man schon ganz zu Beginn gemacht hat!



Eiffel bietet visionäre Möglichkeiten

[1] Die vielen Klassen, deren Namen mit *EV_* beginnen, haben etwas mit EiffelVision zu tun…

Meldung auf die Schnelle

Wenn ihr einfach eine kleine Meldung ausgeben wollt, dann müsst ihr lediglich Folgendes machen:

```
local
        info: EV_INFORMATION_DIALOG
do
        create info.make_with_text("Meine Meldung!")
        info.set_title("Wichtig")
        info.show
end
```

Mein erstes Fenster

Zum Üben kann man einfach einmal ein neues Projekt erstellen (siehe Abbildung). Nach einer langen Kompilierphase stehen dann auch alle Hilfsmittel zur Verfügung. Wir schreiben also als erstes eine ROOT_CLASS, die von EV_APPLICATION vererbt (das ist wichtig, damit wir die graphischen Fähigkeiten von EiffelVision überhaupt haben.) ein Feature *make*, welches unser allererstes Fenster erstellt:

```
class ROOT_CLASS
inherit
        EV_APPLICATION
create
        make
feature
        make is
                local
                        mein_fenster: EV_TITLED_WINDOW
                do
                        default_create
                        create mein_fenster.make_with_title("Mein Fenster")
                        mein_fenster.set_size(400, 300)
                        mein_fenster.show
                        launch
                end
end
```

Das Feature *make* initialisiert also zuerst die Applikation und EiffelVision mit *default_create*, dann wird ein Fenster mit Titel erstellt. Bevor es auf dem Bildschirm angezeigt wird, stellen wir noch die Grösse ein. (Für jetzt nicht so wichtig ist, dass auf der letzten Zeile wird *launch* ausgeführt wird).

Wenn wir ein eigenes Fenster erstellen wollen, das einen Knopf enthält, schreiben wir eine neue Klasse:

```
class MEIN_FENSTER
inherit EV_TITLED_WINDOW
        redefine
                initialize
        end
create
        default_create
feature
        knopf: EV_BUTTON
        initialize is
                do
                        Precursor {EV_TITLED_WINDOW}
                        create knopf.make_with_text("Klick mich!")
                        enclosing_box.extend (calculate_btn)
                        enclosing_box.set_item_x_position (calculate_btn, 126)
                        enclosing_box.set_item_y_position (calculate_btn, 30)
                end
end
```

Der *knopf* ist ein Attribut und im Feature *initialize* wird der Knopf erstellt und im Fenster platziert. *enclosing_box* ist quasi der Inhalt des Fensters, die Fläche, die in Windows grau ist.

Wenn ihr weitere Knöpfe oder auch andere Widgets einfügen möchtet, dann seht unter *vision2.interface.widgets* nach, was es alles gibt. Knöpfe und andere Bedienelemente findet man dort unter *primitives*, Fenster und andere Elemente, die weitere Widgets enthalten können (und deshalb ein *enclosing_box* Feature haben) sind unter *containers* eingeordnet.

Meldungsfenster jeglicher Art wie im vorherigen Beispiel findet man unter *dialogs*.

Passiv und faul, oder Event-driven

GUIs sind ziemlich passiv und machen nichts von sich aus. Sie warten eigentlich nur auf den langsamen Menschen, der das Programm bedient. Wie wir ein Fenster mit einem Knopf erstellen, haben wir gesehen. Aber was macht denn unser *knopf: EV_BUTTON*, wenn er gedrückt wird? Eine Aktion, aber diese muss erst noch zum Ereignis hinzugefügt werden.

Aktionen sind im Prinzip nichts anderes als Features, die nach dem Eintreffen eines Ereignisses ausgeführt wird. Aber was ist ein Ereignis? Nichts anderes als ein Feature Aufruf. Aber dieser Feature Aufruf is spezieller: man kann sich anmelden, dass man auch aufgerufen werden möchte. Beim Knopf geht das so:

```
knopf.select_actions.extend(agent meine_reaktion )
```

knopf ist vom Typ *EV_BUTTON*, *select_actions* ist vom Typ *EV_NOTIFY_ACTION_SEQUENCE* und *meine_reaktion* ist ein Feature ohne Parameter und ohne Rückgabewert. Eine Notifikationsaktions-sequenz ist eine Sequenz (ähnlich einer Liste) von Aktionen. Man kann Aktionen mit dem Feature *extend* anhängen. Dann wird man immer informiert, wenn das Ereignis *select* (deshalb heisst das Feature *select_actions*) eintrifft.

Spitzel in Eiffel

Komischerweise wird im oberen Beispiel ein Feature *meine_reaktion* als Parameter dem Feature *extend* übergeben. Aber dieses *meine_reaktion* ist gar keine Query! Wir wollen eigentlich ja auch keinen Wert der Notifikationsaktionssequenz übergeben, sondern das Feature selbst. Dies können wir, indem wir einen Agenten vom Feature erstellen, welcher bei Bedarf

das Feature aufrufen geht. *agent meine_reaktion* macht so etwas wie einen Zeiger auf das Feature.

Referenzen

Agenten – englische Dokumentation – http://docs.eiffel.com/eiffelstudio/general/guided_tour/language/tutorial-12.html

Wie man einen Taschenrechner in EiffelVision macht – http://www.cs.yorku.ca/eiffel/vision2/Eiffel%20Vision%20Calculator.pdf

EiffelVision 2 Referenz mit Beispielen – http://docs.eiffel.com/eiffelstudio/libraries/vision2/index.html