BARB Yann
MOHR Alexandre

TX20

Report

Responsible for TX20 : Eric Bachard

Spring 2006

Acknowledgment

We want to thank here all the people who help us in the redaction of this report. Eric Bachard for his precious help, his patience and his lead all along this semester. Apple for the machines which have been very useful. Michele Valenza for his participation to the oral presentation of our report. And finally, we thank to all the people on the channel IRC, fr.openoffice.org and openoffice.org

Introduction

Nowaydays, the project OpenOffice.org is one of the biggest project of the opensource community. Available for the principle OS, like Linux, Mac OS X and Windows, it is a perfect example of what the opensource community is capable of. But, as any other sofware there is always something to do in order to improve it. For instance, let's take the case of the Mac OS X version of OpenOffice.org. Today, to install this version, we have to first install what we call X11. X11 is a graphic server, responsible of all the communication between the OS and the different display devices (screen, graphic card). Without X11, OpenOffice.org won't running at all. Of course, for a qualified person, this is not a impossible task to install X11, but for the simple user who want only have to install this program, it is completely different and may be he will prefer and much simpler software to use. By this simple observation, some people of the community have decided to create a version of OpenOffice.org running without using X11 but using instead the graphic server of Mac OS X called Quartz.

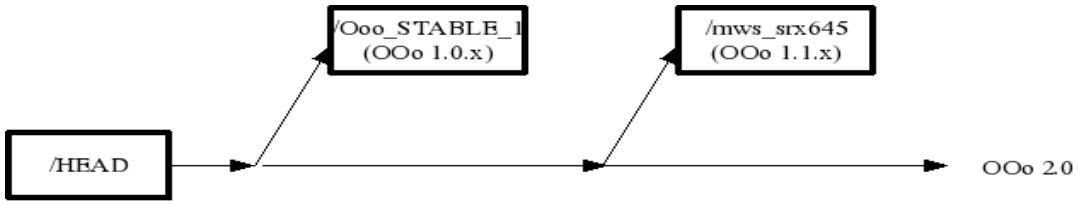
This report has been divided into two parts. In a first time, we are going to see the organisation of the OpenOffice.org project. Most particulary, we will see in detail the process of creation of each new release by using CVS, and the organisation of the different modules of this project. In the second part, we will talk about a specific module called VCL, for Visual Class Libray. In a nutshell, VCL is the graphics engine of OpenOffice.org. Without it, you have nothing on your screen. So we can easily understand that if we are interested by porting OpenOffice.org on Mac OS X, this is what we have to update in order to get a version working under Quartz. As we will see later, is has been decided to use Carbon to create the Quartz Version of OpenOffice.org. So we will see what is Carbon, how we use it and how it is implemented in the VCL module.

Summary

I) Overview of the OpenOffice.org build project

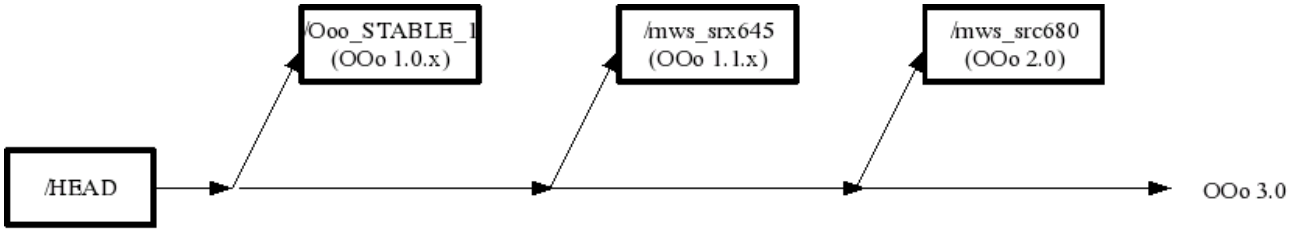
OpenOffice.org project uses **Concurrent Versions System** (**CVS**). CVS is a program who keep track of all work and all changes in a set of files, typically the source code of a software project, and allows several developers to collaborate on the same project. CVS has become very popular in the open-source world. It is released under the GNU General Public licence.

The major releases of OpenOffice.org are implemented on CVS branches. The development of the next major releases is developed on HEAD of the CVS tree, the maintenance of older versions also happens on branches.

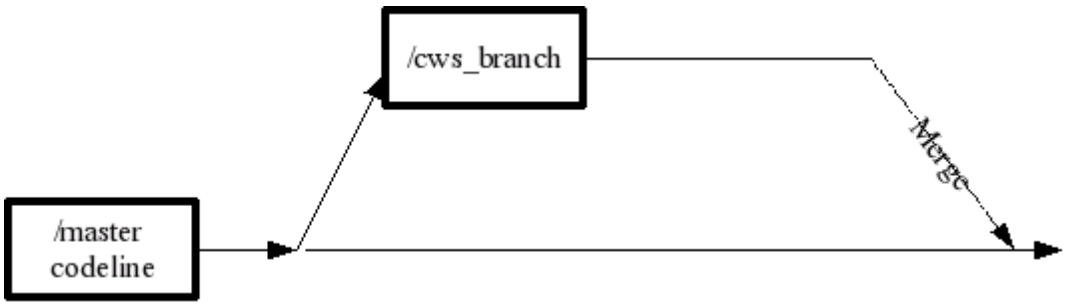


In the OpenOffice.org environment these branches are often called codelines or master. A master workspace represents the road to a product. For the 1.1.x codeline this result is in a cvs branch tag called mws_srx645 which represents the latest status of this codeline. Several more tags represents a specific milestone on this codeline (SRX645_m34, SRX645_m40). The same scheme is applied to the 2.0 codeline (mws_src680 for the latest status, SRC680_m36 for a specific milestone). A **milestone** is a MWS (Master WorkSpace) build on the way to a product. Usually every week there will be a new milestone.

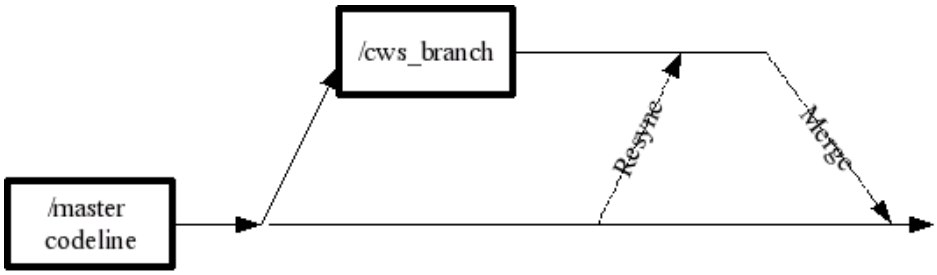
As soon as the OOo 2.0 comes close to release, a new branch for this codeline will be created so that concurrent development of the next release can be started.



OpenOffice.org is developed by more than 100 developers, it builds for more than 10 platforms, for more than 25 languages and it has more than 7 million lines of code with a plenty of build time needed for a complete recompile. So the risk of breaking something is pretty high, even if the comitters is sure about his changes. Due to this complexity and to the will to have at all times a milestone available, the concept of doing any feature development and bug fixes on cvs branches have been developped. This means that a new feature has to be developed on a cvs branch, until the feature is complete and has been tested on at least two major platforms (usually on a Unix derivate and Windows). The same applies for bugs fixes.



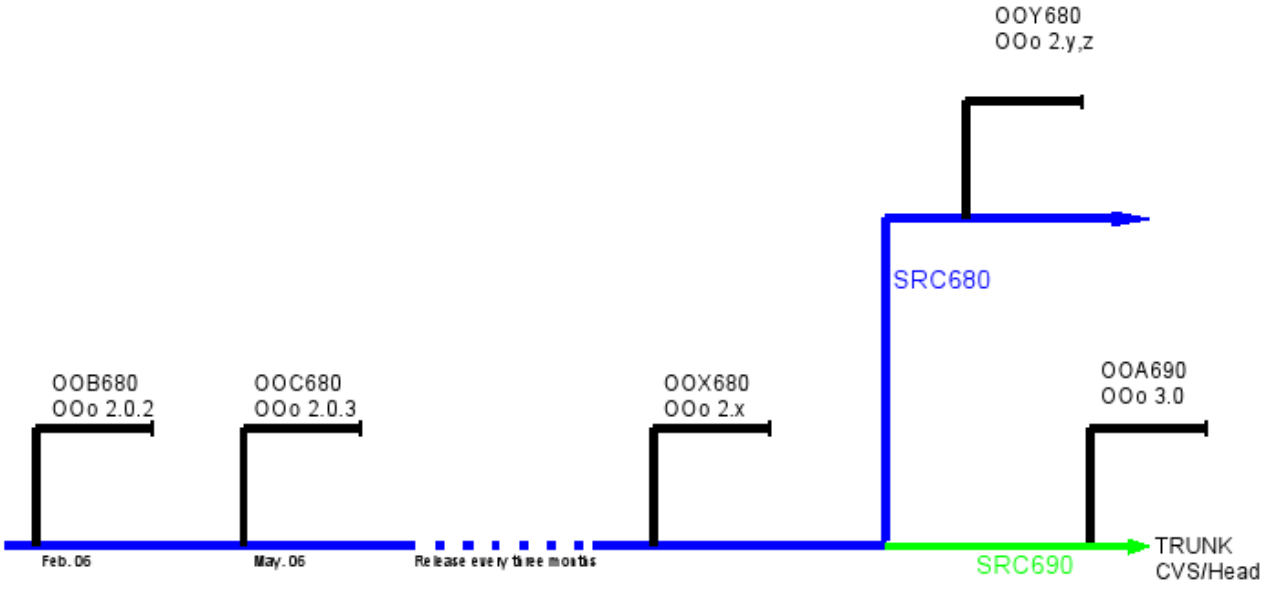
These branches are called in the OpenOffice.org environment **child workspaces** (**CWS**). Since it is possible to create many child workspaces in parallel, some additional processes has been developed to make life more easy and secure on these child workspaces. For example, the problem of “repeated merges” is quite common in cvs. Indeed, this situation occurs frequently when developpers are dealing with many branches in parallel.



Many of the “repeated merge” problems can be solved before the merge back to the master branch if you were able to bring your copy of the cvs branch up to date of the latest know stable version of your master. For this the resynchronization action (cwsresync) command has been introduced.

The resync mechanism is able to deal with repeated operation, so that this process can be executed frequently. In case of a conflict has to be resolved, this will happen in the child workspace, so the risk of having a broken master workspace is less than in the classical approach, when a branch will be merged back to the master with the usual 'cvs update' command.

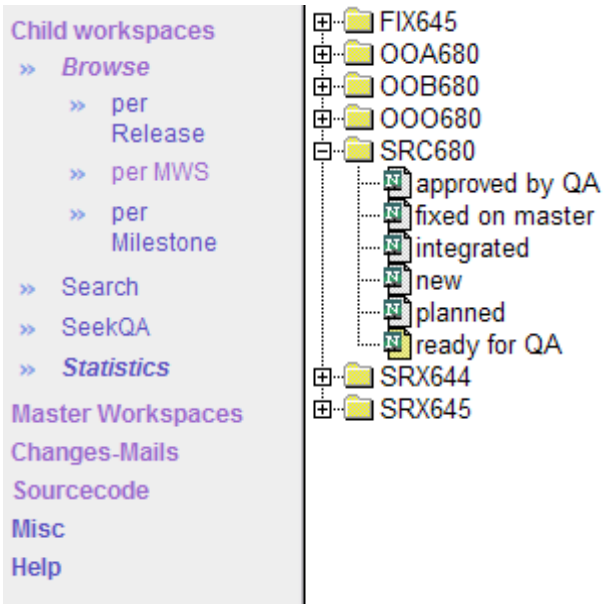
Here is a view more general on the OpenOffice.org project tree:



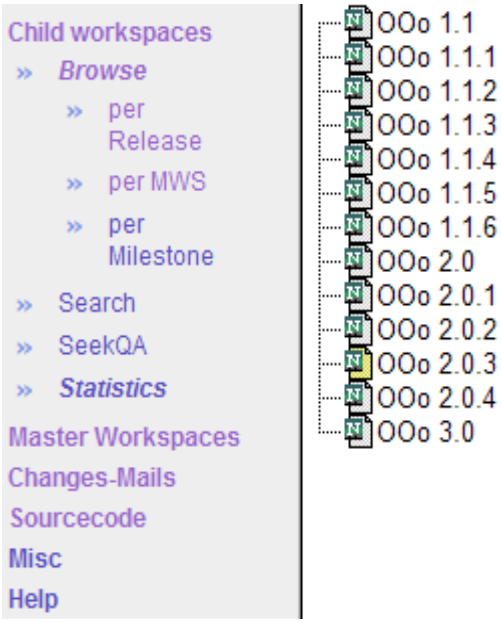
2) Environment Information System (EIS)

It exist a web frontend where we can view the list of all child workspaces and their status (http://eis.services.openoffice.org/EIS2/servlet/GuestLogon). This web frontend is called **Environment Information System** (**EIS**). EIS is in fact a database where MWS and CWS data is stored. EIS offers an friendly interface which provides a easy way to browse all the information about the existing CWSs. It offers several views on the CWSs, sorted either by master workspaces, milestones or releases. It's possible to search for a CWS and to view overall statistics. A click on the CWS name will lead to a detailed overview of the selected CWS. The status is displayed, the members of development and QA working on it and which tasks have been assigned to the CWS. After integration of a CWS there is also a detailed list of changed files with revisions, task-ids and authors. Another important piece of information are the 'creation milestone', 'current milestone' and the 'integration milestone' entries. Additional fields for comments complete the view on the CWS.

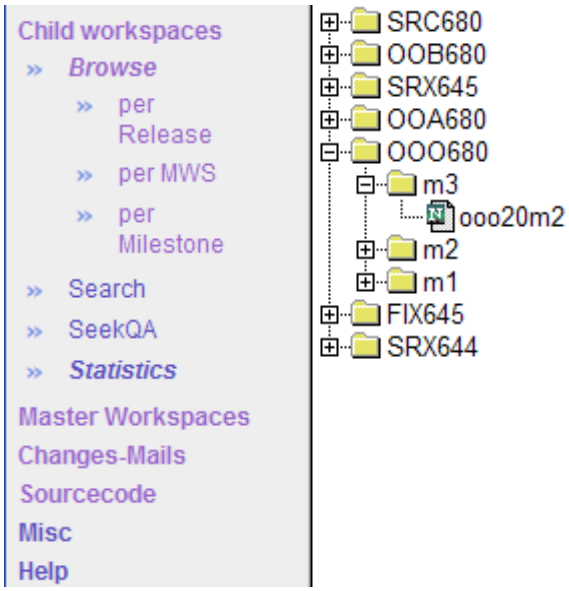
Here are some screenshots of the web interface:



View of the different Master Workspaces on EIS



View of the different Releases on EIS



View of the different Milestones on EIS

Some terms definitions :

1. `planned`: A CWS with this state is planned, but not yet physically existent. Thus no code has been changed, it's not even yet decided from which milestone the CWS will be created. Having this state available is useful for long term planning, resource acquisition etc.

2. `new`: CWSs with this state have been created, they do have a physical representation somewhere. All development is done while the CWS is in this state.

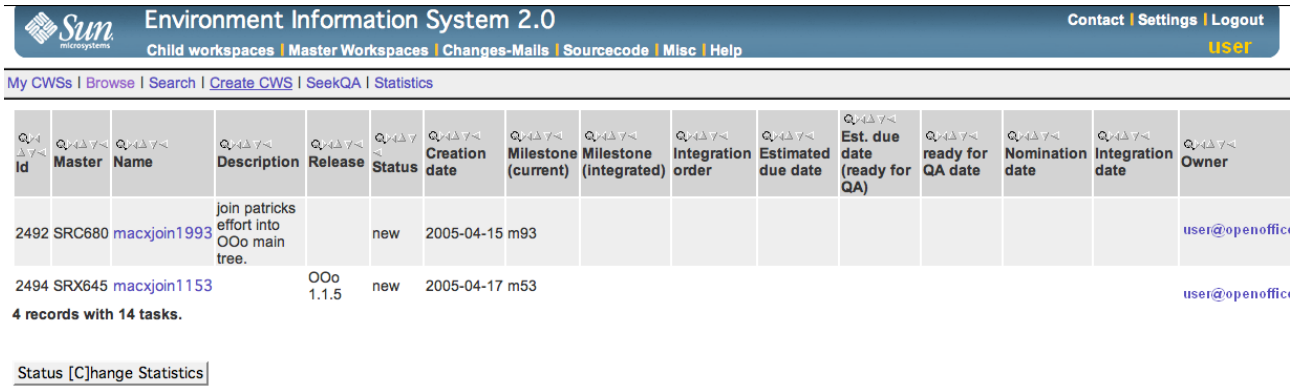
3. `ready for QA`: The developers think they are ready. They have prepared installation sets and submitted them to QA. If QA accepts the CWS the state is usually advanced to `nominated`. If they find bugs in the new stuff or even regressions they will set the state back to `new`.

4. `approved by QA`: A special intermediate state used only for bug fix releases which need an even more controlled approach. QA approves a CWS but program management has the final say if and when something goes into the master.

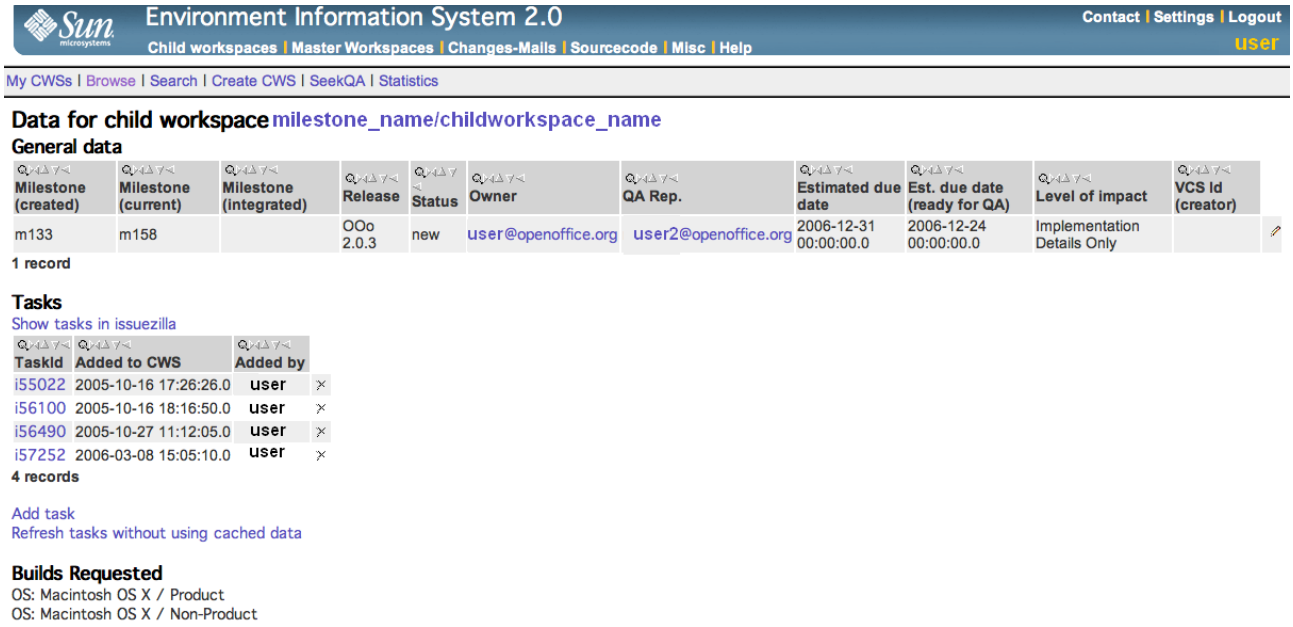
5. `nominated`: Set by QA after accepting a CWS. This is the point where release engineering will take the CWS and integrate it into the master.

6. `integrated`: Set by release engineering after the integration. A CWS in this state is considered done.

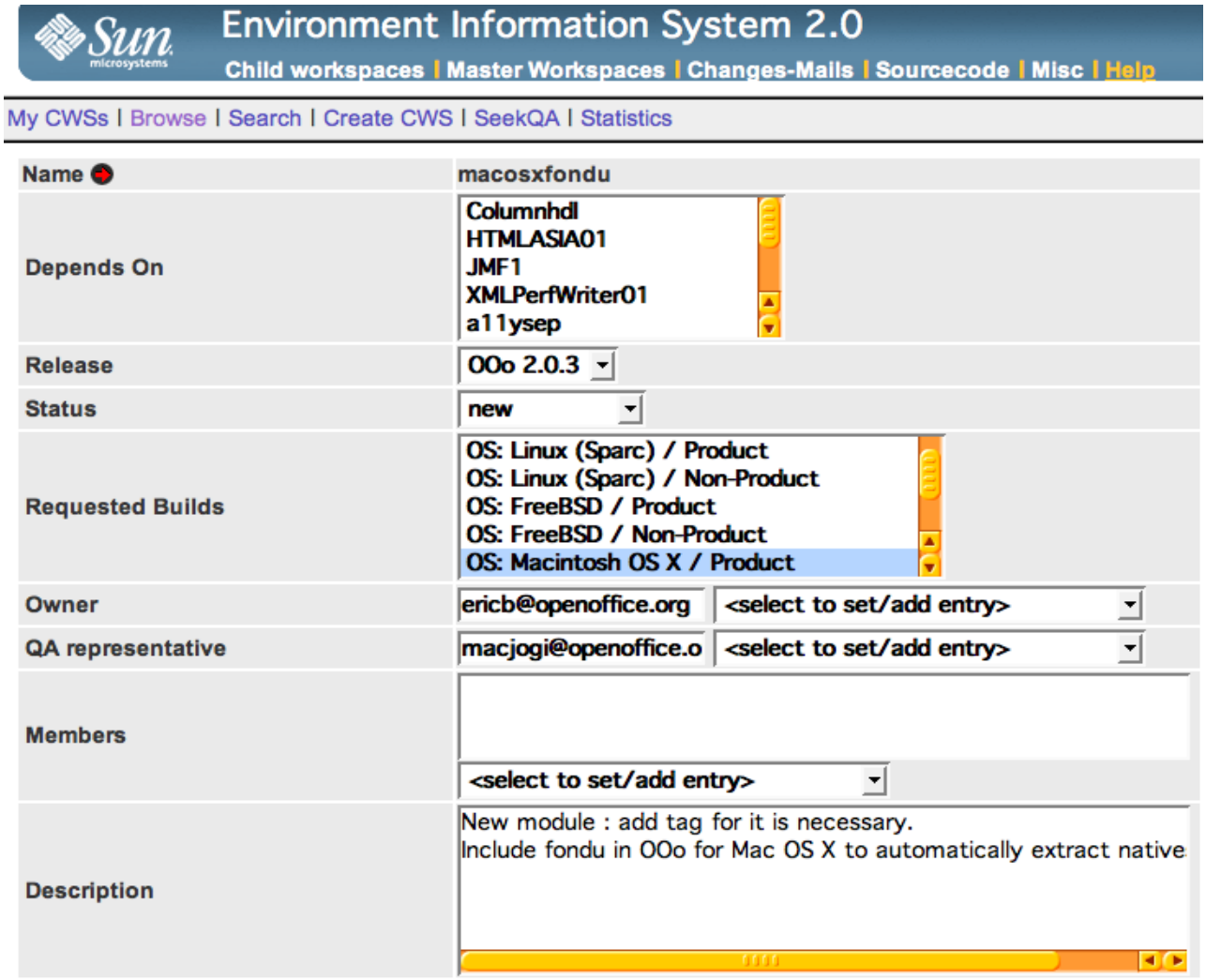
7. `canceled`: A `canceled` CWS has been abandoned, no more work will be done on it.



List of CWS belong to a developer called «user»



Description of a CWS



Creation of a new CWS

3) How do we use CVS ?

In this part, we are going to explain what are the necessary steps in order to create a new CWS.

Creation of a childworkspace :

Before a a child workspace can be created it's necessary to checkout OpenOffice sourcecode – preferably the milestone from which you want to create your CWS – and run configure :

```
$ cvs -d <cvsroot> co -rSRC680_m45 OpenOffice
$ cd config_office
$ ./configure
$ source <the configured script>
```

Now it's possible to create the CWS. Execute the `cwscreate` command.

```
$ cwscreate SRC680_m45 foo1
```

This command first checks if a connection to the EIS web service is available. Then, it updates the OpenOffice modules to the requested milestone and finally registers the freshly created CWS with EIS.

Working with child workspaces :

The environment variable `CWS_WORK_STAMP` is important for the CWS tools; it must be set to the name of your CWS. All CWS tools will refer to this environment variable to find out on which CWS we are working :

```
$ export CWS_WORK_STAMP=foo1
```

Now let's assume we want to implement a feature with the task ID '#i4711#' and fix the related bugs '#i42#' and '#i666#' together in this CWS. The changes are distributed over three modules, let's say 'sfx2', 'framework' and 'desktop'.

Adding tasks IDs to a CWS is done with the `cwsaddtask` tool:

```
$ cwsaddtask i4711 i42 i666
```

We have to not forget to assign a QA engineer to be responsible for that particular CWS. He will test the changes according to the specifications, test plans and bug descriptions of the task IDs which are registered in the EIS.

Now, it's time to do some coding. Add the needed modules with the `cwsadd` command to the CWS:

```
$ cwsadd sfx2 framework desktop
```

The `cwsadd` command creates the CWS branch in these modules, updates the modules to

the branch label, tags them and finally registers the modules with the EIS. All files in the CWS instance of the modules should now carry the CWS branch label as a sticky tag. The CWS branch label is of the form `cws_<mws-name>_<cws-name>`, in our example it is `cws_SRC680_foo1`.

Committing the changes is easy. Just use the `cvs commit` command. The sticky tag ensures that the changes go the CWS branch.

After we have finished our work we'll need to create an installation set. Change into the 'instsetoo' module and build it. The installation set is then created in the output tree.

Now we want to know which files have changed in our CWS and if they can be integrated without conflicts into the MWS. The `cwsanalyze` tool determines all changed files in the CWS and does a trial merge with the MWS for a conflict analysis.

```
$ cd /tmp
$ cwsanalyze all
```

The result is a list of all changed files and a notice as to whether they can be merged back into the MWS without conflict.

Since we have created our CWS, a lot of other CWSs have been integrated into the MWS. To be able to judge if our changes still play well with the current state of affairs we should resynchronize our changes with the master.

Let's say we created our CWS based on the milestone 'm45', but the latest milestone is 'm50'. The `cwsresync` command can resynchronize single files directly in your CWS copy. But, if we plan to resynchronize a whole child workspace, it's far safer to do the CVS operations in a scratch directory. Here the scratch directory is `/tmp`.

```
$ cd /tmp
$ cwsresync -m m50 all
 ... solve conflicts ...
$ cwsresync -c all
$ cd <workspace>
 ... update files in your CWS copy ...
$ cwsresync -l m50
$ cwsresync -r (optional)
$ cd <workspace>/config_office
$ ./configure
$ source <configured script>
```

The first command merges the changes from minor 'm45' to minor 'm50' on the MWS into the CWS. This might lead to conflicts, we need to solve them before we commit the merges onto the CWS branch. If every conflict is solved then we commit all changes with the `cwsresync -c all` command.

Next we update our modules and the current milestone information with the `cwsresync -l` command. Finally we can remove the module output trees with `cwsresync -r`.

The `cwsresync` command is smart enough to avoid the dreaded 'repeated-merge-syndrome'. This is done by keeping an administrative tag called 'anchor tag'. The anchor tag for our

example is CWS_SRC680_FOO1_ANCHOR.

Now we can rebuild our stuff and hand it over to the responsible QA engineer. If everything is okay he/she will approve the CWS and nominate it for integration. Release engineering then takes the CWS and integrates it with the cwsintegrate all command into the MWS.

All revision information including task IDs and authors are finally transferred to EIS. A CWS has reached the end of its life when it is successfully integrated.

4. Organisation of OO modules

What is a module ?

A module is a set of files sorted like this:

Directory	Description
module-name	The root directory of the module.
module-name/inc	Contains the header files and interface descriptions for the module.
module-name/prj	Contains the file d.lst. This file lists all the deliverables of the module. It details where the deliverables come from and where they go to.
module-name/source	Contains source files and a makefile to compile the source.
module-name/util	Linking to binaries occurs here. This directory contains a makefile that specifies how to build the module libraries or binaries.
module-name/$INPATH	The name of this directory comes from the INPATH variable. The INPATH variable derives from the OUTPATH and PROEXT variables. For example, a directory called *module-name*/unxlngi3.pro may exist or will be created when starting to build this platform. All compiled objects, libraries, and binaries are built into this directory. From there they are delivered to solver.
module-name/common.pro	Contains platform-independent output, such as resource files, .jar files, and .zip files.
module-name/res	Contains typical resource files such as bitmaps, icons, and cursor files.
module-name/sdi	Contains View Definition Interface files.
module-name/unoidl	Contains the UNO IDL compiler for .idl files, supplied with backends for C++, Java, documentation, and so on.
module-name/workben	Contains test applications.
module-name/mac|aqua	Contains implementation files specific to Macintosh (with X11 | without X11
module-name/unx	Contains implementation files specific to X Windows System.
module-	Contains implementation files specific to Win32.

name/win

The following table lists the subdirectories of a typical output directory, and describes the contents of those subdirectories.

Directory	Description
$INPATH	Root directory of the output structure.
bin	Contains binary and files.
class	Contains Java-compiled class and/or jar files.
dbo	In the past, this directory contained debug information from the Writer project only. It is obsolete now.
dib	
dlb	In the past, this directory contained debug information from the Writer project only. It is obsolete now.
doc	Contains generated HTML.
dso	In the past, this directory contained debug information from the Writer project only. It is obsolete now.
idl	Contains Interface definition Language (IDL) files.
inc	Contains project interface header files.
lib	Can contain the following files: • .a - Contains static UNIX libraries. • .so - Contains shared UNIX libraries. • .lib - On UNIX systems, contains a list of object files. On Win32 systems, contains a collection of object files. • .dump - Contains the symbols within a library.
misc	Contains a record of some of the commands run by the make process. This also contains the generated dependency description for this module. Typically, tools such as makedep, javadep, or rscdep generate this description. Also contains generated Java files, in a java subdirectory.
obj	Contains object files.
res	Contains resource files. These are organized in subdirectories named according to language codes. There are bitmaps in these subdirectories.
slb	Contains .lib files. These list the objects to be compiled into a shared library. On Win32 systems, the .lib files are a collection of objects.
slo	The shared library object (slo) directory contains object files that appear in shared libraries. Objects that appear in shared libraries appear in both the obj and slo directories.
srs	Contains string resource files.
www	Contains files published on the internet

Now, we undestand, how the OpenOffice.org project is organised and how it is built, we can now take a deeper look into a specific module : the Visual Class Library

II. VCL

1. General description of Vcl

Visual Class Library (VCL) is the window management and basic control library of OpenOffice.org. It includes the system abstraction layer for the user interface components such as:

- Windows
- Printing
- Fonts

The vcl project is divided in several parts:

`vcl/source/app` Contains the base application functionality such as:

- Application Class
- Main
- Timer
- Config
- Sound

`vcl/source/gdi` Contains all independent output functionality such as:

- Bitmap
- Region
- Polygon
- Gradient
- Font
- Graphics output

`vcl/source/window` Contain base window handling and some generic Windows classes.

`vcl/source/control` Contains basic controls such as:

- Edit
- FixedText
- PushButton
- CheckBox
- RadioButton

`vcl/[aqua|unx|win]` Contains the Graphics System Layer (GSL). This is the connection from the independent classes to the system APIs. For instance, on Mac OS X system, this is the

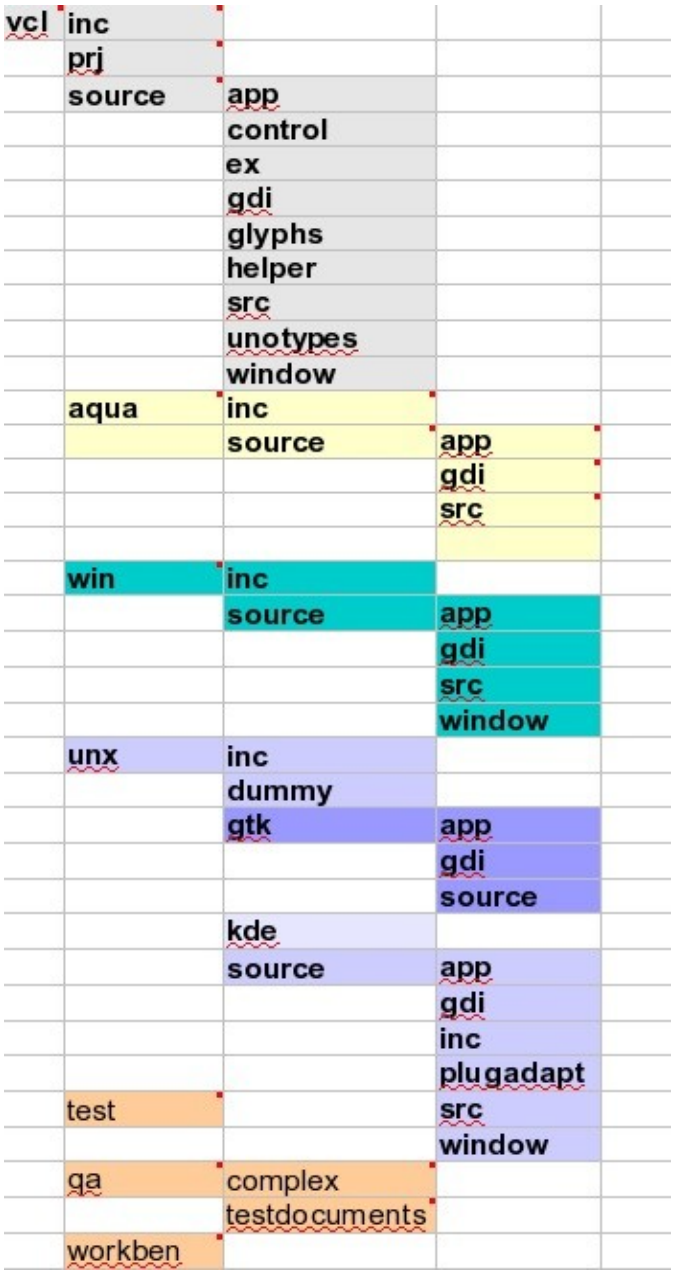
Carbon API which is used and on Unix family system, this is the classical X11 API which is used.

Common part :

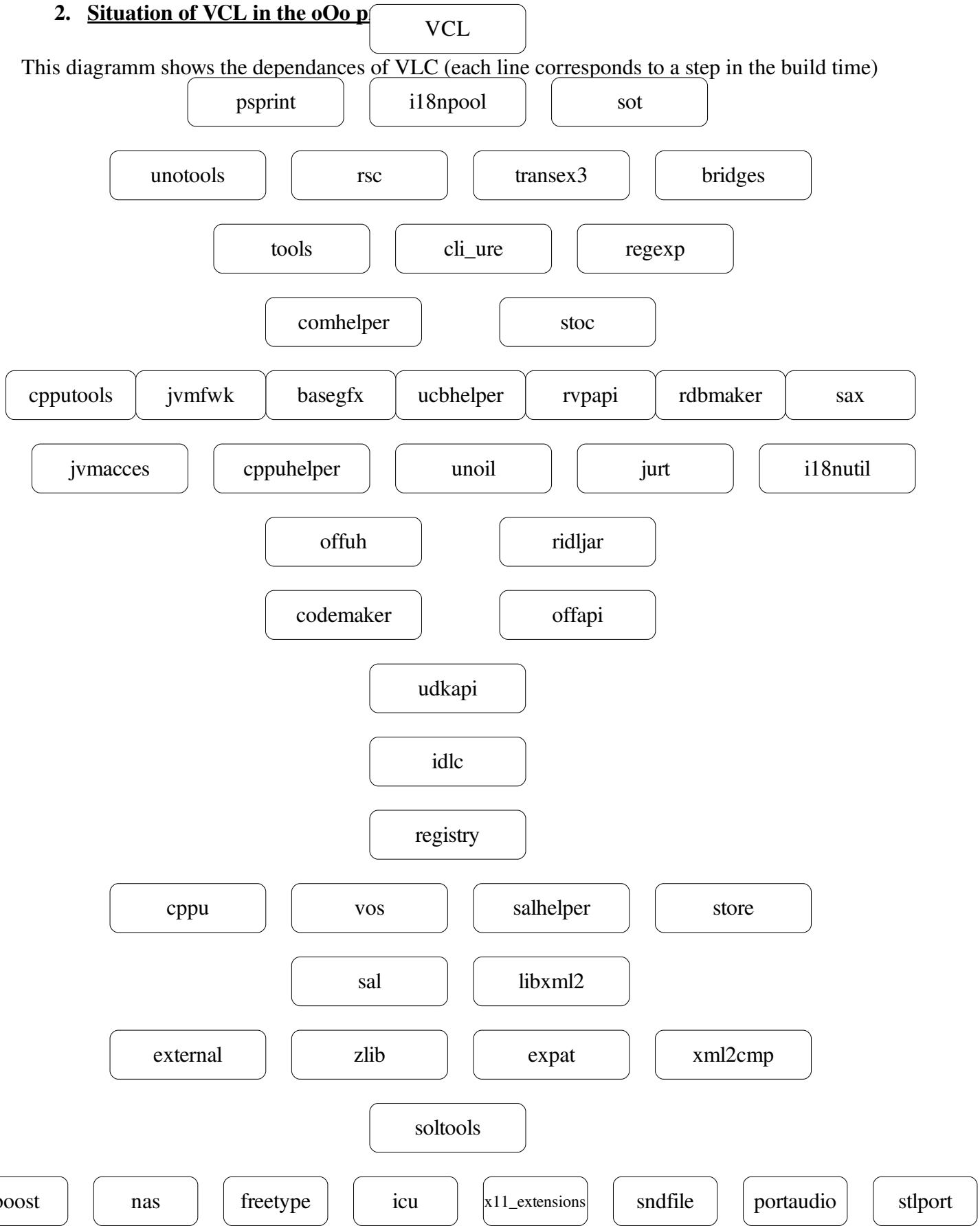
- in grey on right. The result will be a non architecture dependant library, built in **all cases** : for instance, libvcl680mxi.dylib on Mac Intel.

Specific part :

- Light yellow : aqua part will only concern Mac OS X (non X11)
- Green : Windows part
- Light Blue : Unix (Linux, Solaris or current Mac OS X X11)



2. Situation of VCL in the oOo p



This diagram has been built from our dependencies tree program. (see on the CD-ROM)

3. Aqua implementation

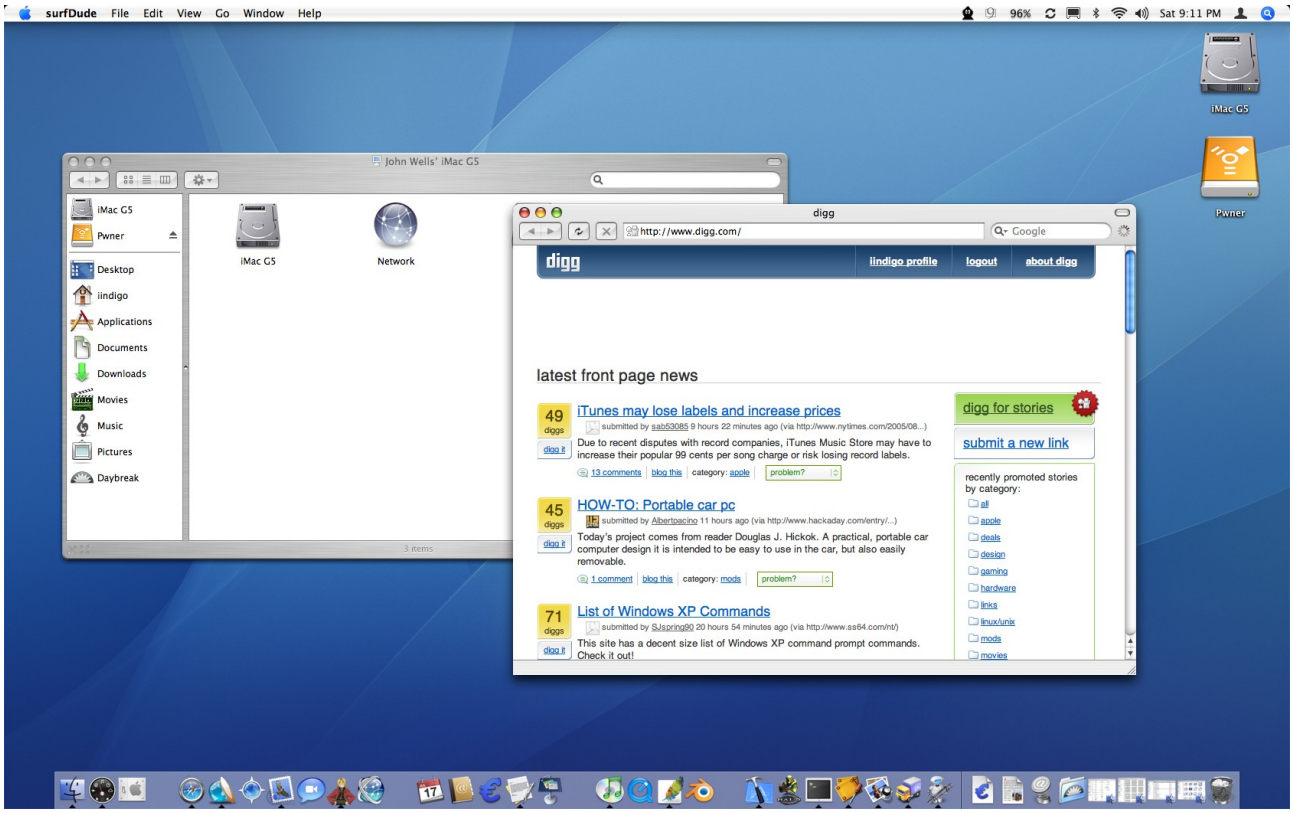
1. Carbon API

For porting OpenOffice.org on Aqua, it has been decided to use Carbon instead of Cocoa. In this part we are going to first see, what is Carbon. And in a second time, we will see how works the Carbon API, by looking at an application build with this API.

What's Carbon ?

Let's go back in 1998. This year at the World Wide Developers Conference, Apple the introduction of a new operating system, to be known as Mac OS X. Mac OS X, the first version of which was released on 24 March 2001 and it's not just another Mac OS update; it is a completely new operating system complete with "modern" operating system features such as pre-emptive multitasking and protected memory. It features a completely new user interface, called Aqua, whose appearance and behavior differs significantly from that of the original Mac OS (represented in its latest, and no doubt last, incarnation by Mac OS 9).

Mac OS 9

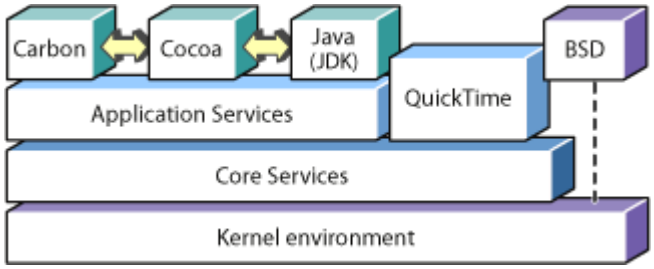


Mac OS X

Mac OS X runs on G3 and G4 PowerPC machines only, meaning that machines based on PowerPC 604 and 603 microprocessors must necessarily remain with Mac OS 9 and earlier. A large installed base of these latter machines will no doubt remain for many years to come. In addition, it is likely that many owners of machines capable of running Mac OS X will nonetheless remain with Mac OS 9 and earlier. In these circumstances, it was perceived as all but essential that programs written to take advantage of Mac OS X's advanced features also be capable of running on Mac OS 8 and 9 without modification. In this scope, Apple has devised the means whereby this can be achieved, namely, the Carbon API.

Carbon is a set of C APIs offering developers an user interface tool kit, event handling, the Quartz 2D graphics library (we will see a little bit later what is it), and multiprocessing support. Developers have access to other C and C++ APIs, such as the OpenGL drawing system too. This API derived from earlier Mac OS APIs which have been modified or extended to take advantage of new Mac OS X features. Originally designed to provide a gentle migration path for developers transitioning from Mac OS 9 to Mac OS X.

Carbon is one of several application environments available on Mac OS X as we can see on the following diagram:



These other environments include:

- Cocoa : the object-oriented interface for writing only Mac OS X applications in Objective-C.
- Java : a JDK-compliant virtual machine for running Java applications.

These environments depend on the same application and core services for their operation, and the underlying services rely on Darwin (Apple's open-source core operating system) and the Mach kernel.

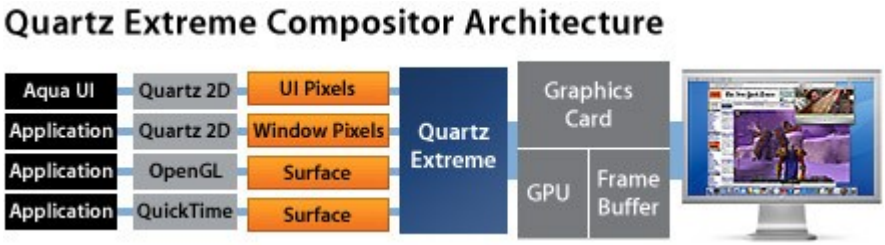
Carbon contains thousands of functions, data structures, and constants. Related functions and data structures are organized into functional groups, usually referred to as managers or services. For example, the Window Manager contains functions and data structures that let you create, remove, and otherwise manipulate application windows. The Event Manager contains functions that let you create, remove and manipulate events, such as mouse events, keyboard events, in your application.

A further advantage of Carbon is that existing applications can be "carbonized" with much less effort that would be required to completely rewrite them for Mac OS X using the Cocoa API.

What's is Quartz ?

A little bit before, we talked about the Quartz Engine 2D. But what is it exactly ?

In fact, every time you move, resize and scroll a window, you're using Quartz Extreme window compositor. This engine uses OpenGL technology to convert each window into a textures, then sends it to the graphics card to render on screen. Quartz 2D, is one component of Quartz Extreme. It is the primary graphics library in Mac OS X and it succeed to QuickDraw, which was used in earlier versions of Mac OS. Quartz 2D is based on PostScript and PDF. It provides access to features such as transparency layers, offscreen rendering, PDF document creation, ... The Quartz 2D API is part of the Core Graphics Framework. We'll see that every functions using to draw something on a window begins by *CG*.



We are now going to take a deeper a look on how we use the Carbon API both to draw objects on a window which has been created and to handle events. For illustrate this part, we will insert some source code using the Carbon API. All source codes have been written with the Mac OS application Xcode. All we'll see just after is just a little introduction to Carbon. We are going only to scratch the surface. For more informations and more details the website http://developer.apple.com can be very helpful to understand the whole possibilty of Quartz.

How do we create a window ?

To create a window, the preferred method is to call the function `CreateNewWindow`.

```
OSStatus CreateNewWindow (
                WindowClass windowClass,
                WindowAttributes attributes,
                const Rect * contentBounds,
                WindowRef * outWindow);
```

Parameters "`windowClass`" and "`attributes`" define properties of the created window. A window can only have one class but can have several attributes.

Attributes are added like this:

```
WindowAttributes  windowAttrs = kWindowStandardDocumentAttributes |
kWindowStandardHandlerAttribute | kWindowInWindowMenuAttribute;
```

List of classes: see Annex 1

List of Attributes: see Annex 2

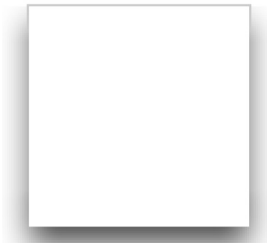
The `contentBounds` parameter is a structure describing the global coordinates of the content region.

```
SetRect (&contentBounds, LeftTop, Right, Bottom);
```

The last parameters "`WindowRef * outWindow`" will contains a reference to the new window after the execution of the function.

The created window (outWindow) is invisible and placed at the front of the window list. In order to display it, the function "`void ShowWindow( WindowRef outWindow)`" could be used.

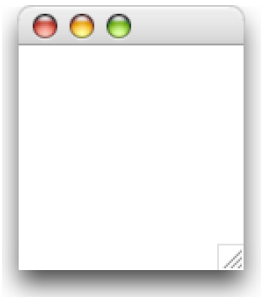
Window examples:



Class: kAlertWindowClass
Attributes: kWindowNoAttributes

Class: kDocumentWindowClass
Attributes: kWindowNoAttributes

Class: kDocumentWindowClass
Attributes: kWindowStandardFloatingAttributes



Class: kDocumentWindowClass
Attributes: kWindowNoAttributes
kWindowStandardDocumentAttributes
kWindowLiveResizeAttributes

In Connect 4 :

In our program, the function p4CreateNewWindow has been written to deal with all of this.

```
WindowRef         window=NULL;

// Class and Attributes for a floating and non-resizable window.
WindowClass windowClass = kDocumentWindowClass;

WindowAttributes  attributes =   kWindowStandardHandlerAttribute   ;
                  attributes |=  kWindowStandardFloatingAttributes ;

// Window size
Rect contentBounds;

// Set content rectangle  order : Left,Top,Right,Bottom
SetRect (&contentBounds, 100,100,100+WindowWidth,100+WindowHeight);
CreateNewWindow (windowClass,attributes,&contentBounds,&window);
// diplay the window
ShowWindow( window );
```

How do we create menus ?

In order to create a new menu, the function CreateNewMenu() has to be called :

```
OSStatus CreateNewMenu (
      MenuID inMenuID,
      MenuAttributes inMenuAttributes,
      MenuRef * Menu );
```

The `inMenuID` parameter is used to reference the Menu with a number.
The `inMenuAttributes` parameter gives attributes of the created menu. (see in Annexe 3)
The last one, `MenuRef`, contains the pointer of the created menu after the function's execution.

The next step is to give a name to the menu like this :

```
SetMenuTitleWithCFString(MenuRef Menu, CFSStringRef inString)
```

To obtain a string in CFSStringRef type, we use the function

```
CFSStringRef CFSTR(const char *cStr)
```

Now, the menu is created and it has a name, so we will add it in the menu bar :

```
void InsertMenu(
       MenuRef Menu,
       MenuID BeforeID )      // Menu ID of the previous menu
```

After that, we will add items to it. The function used is:

```
InsertMenuItemTextWithCFString(
       MenuRef Menu,
       CFStringRef ItemName,
       MenuItemIndex inAfterItem,     // Item ID of the next one
       MenuItemttributes inAttributes,
       MenuCommand inCommandID);      // ID of the current Item
```

In order to have many menus, we have to repeat all these operations.

Example:

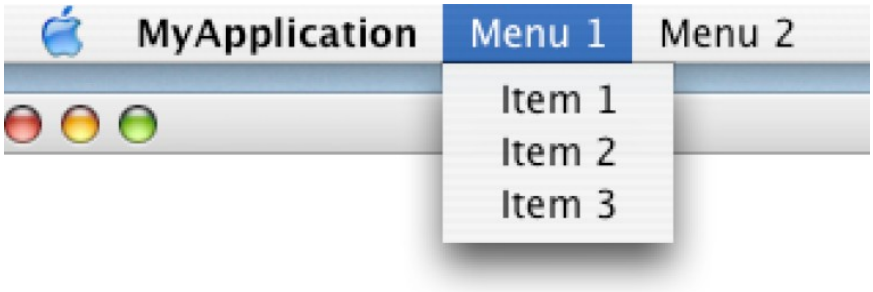
```
// Menu 1
MenuRef Menu1;

CreateNewMenu(1,0,&Menu1);  // 1: Menu ID  0: attribute(s), &Menu1: Created menu

SetMenuTitleWithCFString(Menu1,CFSTR("Menu 1")); // Set menu name
// Insert the menu Menu1 after the menu which has the ID 0

InsertMenu(Menu1,0);   //item
        // Item 1 is put after item 0 (which doesn't exist yet)
        InsertMenuItemTextWithCFString(Menu1,CFSTR("Item 1"),0,0,1);
        // Item 2 is put after item 1
        InsertMenuItemTextWithCFString(Menu1,CFSTR("Item 2"),1,0,2);
        // Item 3 is put after item 2
        InsertMenuItemTextWithCFString(Menu1,CFSTR("Item 3"),2,0,3);

// Menu 2
MenuRef Menu2;
CreateNewMenu(2,0,&Menu2);
SetMenuTitleWithCFString(Menu2,CFSTR("Menu 2"));
InsertMenu(Menu2,0);
        //item
        InsertMenuItemTextWithCFString (Menu2,CFSTR("Item 1"),0,0,1);
        InsertMenuItemTextWithCFString (Menu2,CFSTR("Item 2"),1,0,2);
```



How do we handle events with Carbon ?

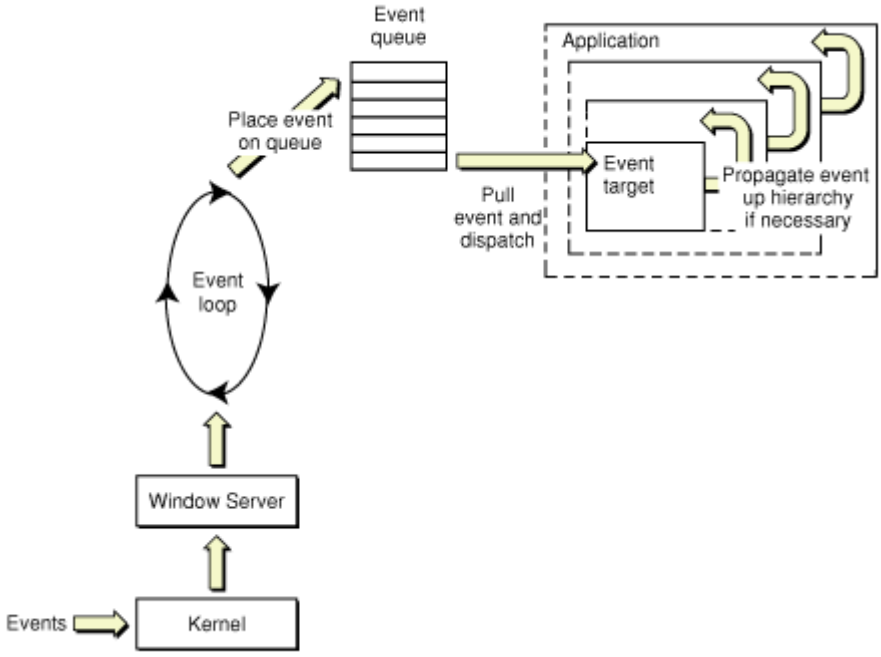
Events are the foundation of all Carbon programming. Each time the user clicks the mouse, types a character from the keyboard, or chooses a command from a menu, you're notified by means of an event. When one of your windows needs to be redrawn, moved, or resized, your application receives an event telling you to perform the operation. When your program becomes the active (foreground) application or moves to the background in favor of another, or when another application starts up or quits, you receive an event informing you of the fact. Just about everything a typical Carbon program does, whether interacting with the user or communicating with the system, takes place in response to an event.

The Carbon Event Manager is the preferred interface for handling events in Carbon applications. You can use this interface to handle events generated in response to user input as well as to create your own custom events.

Some of the types of events that the Carbon Event Manager can handle include the following:

- Window events: resizing, closing, activation, moving, window updates, and so on.
- Menu events: menu tracking and selection, keyboard shortcuts, and so on.
- Control events: activation, selection, dragging, changes in user focus, and so on.
- Mouse events: mouse-up, mouse-down, mouse movement, multiple clicks, multiple buttons, dragging, chording, rollover states, scroll wheel operation, and so on.
- Text and keyboard events: Unicode or Macintosh-encoded text input and raw keyboard presses.
- Application events: application activation, deactivation, requests to quit, and so on.
- Apple events
- Volume events: insertion or ejection of CDs and disks.
- Tablet events: tablet proximity and movement.

Events are transmitted to an Event Loop by the API (Carbon) which replaces them in the Event Queue. It is like a stack and our Application will treat them one after the others in the chronologic order. The application can propagate an event up in hierarchy or call the next event in the stack.



Each Carbon event is defined by an event class (for example, mouse or window events) as well as an event kind (for example, a mouse-down event).

All of the available event classes and kinds are designated by constants defined in the Universal Interfaces header file `CarbonEvents.h`. (See in Annexe 4 for a non-exhaustive list)

The API transmits Event to the event loop in condition that we have install Event handler before launching the loop. In practical, we need to say to the API which specific event we want to transmit to our program. If nothing is specified, the event queue will always be empty.

The standard function for installing event handler is :

```
OSStatus InstallEventHandler (EventTargetRef         target,
                              EventHandlerUPP        handlerProc,
                              UInt32                 numTypes,
                              const EventTypeSpec*   typeList,
                              void*                  userData,
                              EventHandlerRef*       handlerRef);
```

The parameters of the functions are the following:

- `Target`: The event target to register your handler with.
- `HandlerProc`: A pointer to your event handler function. The function `NewEventHandlerUPP(FunctionName)` is used to get the function's pointer
- `numTypes`: The number of events you are registering for.
- `typeList`:A pointer to an array of EventTypeSpec entries representing the event you are interested in.

Example for 2 types:

```
EventTypeSpec    eventTypes[2];

eventTypes[0].eventClass = kEventClassKeyboard;
eventTypes[0].eventKind  = kEventRawKeyDown;

eventTypes[1].eventClass = kEventClassKeyboard;
eventTypes[1].eventKind  = kEventRawKeyRepeat;

InstallApplicationEventHandler (handlerUPP, 2,
eventTypes, NULL, NULL);
```

- `userData`: The value you pass in this parameter is passed to your event handler function when it is called.
- `handlerRef`: Pointer which will contain the event handler reference. It will be used later if we want to remove the handler.

In Connect 4:

In our program, the function `InstallMouseEvent` has been written to deal with all of this. We need to know when a player clicks with the mouse so we have installed an event handler on the mouse: `kEventClassMouse`, with event kind `kEventMouseDown`. The standard function for installing event handler is `InstallEventHandler` but for convenience, we prefer use a specific function: `InstallWindowEventHandler`

```
InstallWindowEventHandler (
               WindowRef              theWindow,
               EventHandlerUPP        handlerUPP,
               UInt32                 numTypes,
               const EventTypeSpec*   typeList,
               void*                  userData,
               EventHandlerRef*       &handlerRef );
```

Installation of the mouse event handler:

```
void InstallMouseEvent(p4_t* p4)
{
        EventTypeSpec     eventType;

                // Set event class
                eventType.eventClass = kEventClassMouse;

                // Set event kind
                eventType.eventKind  = kEventMouseDown;

        InstallWindowEventHandler(
                        p4->window,
                        NewEventHandlerUPP(mouse_event),
                        1,
                        &eventType,
                        p4,
                        &p4->mouse_event);
}
```

After having installed some event handler, the Event loop has to be launched in order to begin to record events. The function which runs the event loop is:

```
void RunApplicationEventLoop ()
```

The function which quits the event loop is:

```
void QuitApplicationEventLoop();
```

During the event loop, the application can install new handler or uninstall old one with this function:

```
OSStatus RemoveEventHandler(
            EventHandlerRef inHandlerRef );
```

There are other Carbon functions which are very useful, such as:

- `AddEventTypesToHandler(…)` and `RemoveEventTypesFromHandler(…)`: Change dynamically which events you want your handler to respond to.
- `CallNextEventHandler(…)`: Call the next event in the event stack.
- `GetCurrentEventQueue(…)` `GetMainEventQueue(…)`: Obtain the event queue for the current (main application) thread.
- `PostEventToQueue(…)`, `RemoveEventFromQueue(…)` : Add or Remove an event from the event queue.
- `IsEventInQueue(…)`: Determine whether an event is in a particular queue.
- `FlushEventsMatchingListFromQueue(…)`: Remove events from the event queue by kind and class.
- `FlushSpecificEventFromQueue(…)` : Remove specified events from the event queue.
- `FlushEventQueue(…)` : Remove all events from the event queue.
- `FindSpecifidEventInQueue(…)` : Find specific event in the event queue.
- `GetNumEventsInQueue(…)` : Return the number of events in the event queue.

In Connect 4:

File: main.c

In the main function, we have created a window and installed a mouse event handler on it. So, we can now launch the event loop to begin interaction with players.

```
RunApplicationEventLoop();
```

File: event/event.c

In this case, the player 2 wins the game. We don't need to keep on catching events from the mouse, so we remove the handler on the event "mouse_event".

Note: At this time, there is no event handler installed, so the player can only move the window, reduce it or close it.

```
if(win(p4->t,p4->player))
{
      MyDrawText (p4->window, "Player 2 wins" ,180,20,30);
      RemoveEventHandler(p4->mouse_event);
```

```
}
```

At the end of the mouse_event function, we remove all events in the stack. We do that to avoid bugs: if the player clicks on an empty box during the computer's turn, the program will first draw the choice of the computer, and just after go on to the next event in the stack: the “non-wanted” choice of the player.

```
FlushEventQueue(GetMainEventQueue ());
```

After that, we wait the next event:

```
CallNextEventHandler( handlerRef, event);
```

To create an event handler, we use the following function:

```
static OSStatus MonitorHandler(
               EventHandlerCallRef inCaller,
               EventRef inEvent,
               void* inRefcon )       {   /* Code */   }
```

- `InCaller`: Reference of the event handler called
- `InEvent`: Reference of the event in treatment
- `InRefcon`: The value you pass in this parameter is passed to your event handler function when it is called.

In our Connect 4, we use this for the function `mouse_event`.

Many events require more information than just the basic event to be truly useful. For example, knowing that the mouse was clicked is usually not very interesting unless you know where the click occurred. This additional information is embedded in the event reference structure, and you need to call the function `GetEventParameter` to obtain it. These additional parameters are identified by parameter name and type.

```
OSStatus GetEventParameter (
        EventRef inEvent,
        EventParamName inName,
        EventParamType inDesiredType,
        EventParamType * outActualType,
        UInt32 inBufferSize,
        UInt32 * outActualSize,
        void * outData );
```

A mouse-down event, for example, has four event parameters:

- `kEventParamMouseLocation`, a point (parameter type `typeQDPoint`) giving the screen coordinates at which the mouse button was pressed
- `kEventParamMouseButton`, an integer code (parameter type `typeMouseButton`) identifying which button was pressed (allowing support for a one-, two-, or three-button mouse)
- `kEventParamKeyModifiers`, a set of flag bits (parameter type `typeUInt32`) telling which modifier keys, if any, were being held down at the time the button was pressed
- `kEventParamClickCount`, an integer (parameter type `typeUInt32`) telling how many times the button was clicked in the same location (1 for a single click, 2 for a double click, and so on)

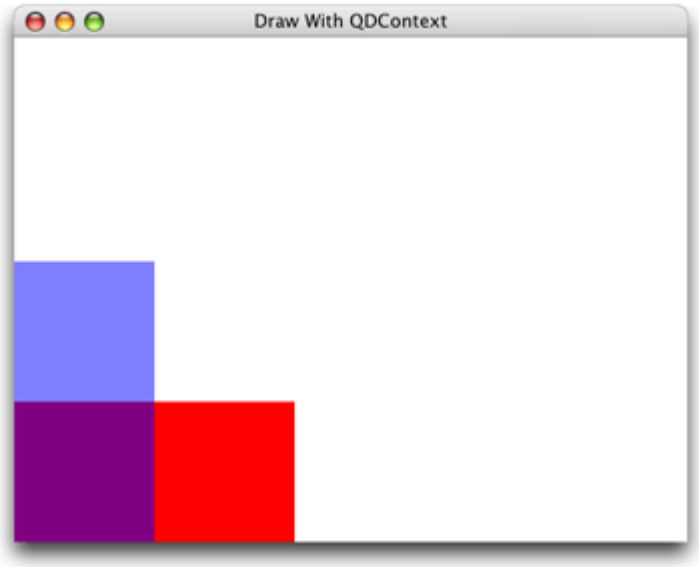
In Connect 4:

```
When the event “mouse_event” is detected, we need to know where the player has
clicked, so we get back the mouse position like this:

Point wheresMyMouse;
      // Give the Mouse position in the var wheresMyMouse from the top left
corner of the screen
GetEventParameter   (   event,   kEventParamMouseLocation,   typeQDPoint,   NULL,
sizeof(Point), NULL, &wheresMyMouse);
```

How do we draw objects on a window with Carbon ?

Let's begin by taking a look at a little example:



For create this window and theses two rectangles, first, we have to create an new carbon application in XCode. After that we have to writting the following piece of code :

```
void MyDrawInWindow (WindowRef window)
{
    CGContextRef myContext;
    SetPortWindowPort (window);
    QDBeginCGContext (GetWindowPort (window), &myContext);

     /* You put your drawing code here
        ...
    */

    CGContextFlush(myContext);
    QDEndCGContext (GetWindowPort(window), &myContext);
}
```

This function is designed to draw objects in the window. In the function main, the function *MyDrawInWindow* is called like that :

```
int main(int argc, char* argv[])
{
    WindowRef window;
```

```
    /* ... */

    MyDrawInWindow(window);
    RunApplicationEventLoop();

    /* ... */}
```

In this little example, there are some important functions to deal with. These function are the following:

- *CGContextRef* defines a Quartz 2D drawing environment.
- *SetPortWindowPort* sets the current graphics port to the window port.
- *QDBeginCGContext* obtains a graphics context for a window port and signals the beginning of Quartz 2D drawing calls. This function allow us to use Quartz function inside QuickDraw. It is a simpler way to use Quartz
- *CGContextFlush* forces all pending drawing operations in a window graphics context to be rendered immediately to the destination device. We must call this function when you obtain a graphics context using the function QDBeginCGContext.
- *QDEndCGContext* signals the end of Quartz 2D drawing calls and restores the window port

Now, we are going to see, some specific methods to draw objects in a window.

How do we draw a text ? :

To draw a text you need to perform these following tasks:

- Set the font and font size.
- Set the text drawing mode.
- Set other items as needed as for instance, stroke color, fill color.
- Set up a text matrix if you want to translate, rotate, or scale the text space.
- Draw the text.

We want to create this window :

For this, we are going to create a function *MyDrawText* :

```
void MyDrawText (WindowRef window, CGRect contextRect);
```

In the function main, we insert the following piece of code :

```
int main(int argc, char* argv[])
{
    WindowRef window;
    CGRect contextRect;

    /* ... */

    MyDrawText(window, contextRect);
    RunApplicationEventLoop();

    /* ... */}
```

This is the code of the function itself (this function takes as parameters a graphics context and a rectangle to draw to) :

```
void MyDrawText (CGRect contextRect, CGContextRef myContext)
{
    float w, h;
    w = contextRect.size.width;
    h = contextRect.size.height;
    CGAffineTransform myTextTransform;
    CGContextSelectFont (myContext, "Times-Bold", h/10, kCGEncodingMacRoman);
    CGContextSetCharacterSpacing (myContext, 10);
    CGContextSetTextDrawingMode (myContext, kCGTextFillStroke);
    CGContextSetRGBFillColor (myContext, 0, 1, 0, .5);
    CGContextSetRGBStrokeColor (myContext, 0, 0, 1, 1);
    myTextTransform =  CGAffineTransformMakeRotation  (radians (45));
    CGContextSetTextMatrix (myContext, myTextTransform);
    CGContextShowTextAtPoint (myContext, 40, 0, "Quartz 2D", 9);
}
```

Let's take a look the Quartz functions include in this piece of code :

- *CGAffineTransform* stores informations for affine transforms.
- *CGContextSelectFont* sets the font to Times Bold and the font size to the height of the page rectangle divided by 10. In this example, the text is drawn into a resizeable window. When the user resizes the window, the text resizes as well. The encoding is set to kCGEncodingMacRoman (MacRoman is an ASCII variant originally created for use in the Mac OS, in which characters 127 and lower are ASCII, and characters 128 and higher are non-English characters and symbols).
- *CGContextSetCharacterSpacing* sets the character spacing to 10 text space units.
- *CGContextSetTextDrawingMode* sets the text drawing mode.
- *CGContextSetRGBStrokeColor* sets the fill color.
- *CGContextSetRGBStrokeColor* sets the stroke color.
- *CGAffineTransformMakeRotation* creates an affine transform that performs a 45 degree rotation.
- *CGContextSetTextMatrix* sets the text matrix to the transform created in the last step.
- *CGContextShowTextAtPoint* draws the text, passing the x- and y-coordinates in text space to start the drawing (40, 0), an array of characters to draw, and a value that specifies

the length of the text array. In this case, you pass a C-style string and the value 9 to specify the number of characters.

How do we construct and draw shapes ?

First, we have to see the notion of path. A path consists of straight lines, curves, or both. It can be open or closed. A path can be a line, circle, rectangle or a more complex shape. Path creation and path painting are separate tasks. First you create a path and when you want to render a path, you request Quartz to paint it.

Some examples of path

When you want to construct a path in a graphics context, you signal Quartz by calling the function `CGContextBeginPath`.

```
void CGContextBeginPath (CGContextRef context);
```

Next, you set the starting point for the first shape in the path by calling the function `CGContextMoveToPoint`.

```
void CGContextMoveToPoint (CGContextRef context, float x, float y);
```

After you establish the first point, you can add lines, arcs, curves, rectangles, or anything you want, to the path.

When you want to close a subpath within a path, call the function `CGContextClosePath` to connect the current point to the starting point.

```
void CGContextClosePath (CGContextRef context);
```

How do we paint a path ?

You can paint the current path by stroking or filling or both. Stroking paints a line that straddles the path. Filling paints the area contained within the path. Quartz has functions that let you stroke a path, fill a path, or both stroke and fill a path.

Functions that affect parameters stroking :

- `CGContextSetLineWidth` : Sets the line width for a graphics context
- `CGContextSetLineJoin` : Sets the style for the joins of connected lines in a graphics context
- `CGContextSetLineCap` : Sets the style for the endpoints of lines in a graphics context
- `CGContextSetMiterLimit` : Sets the miter limit for the joins of connected lines in a graphics context
- `CGContextSetLineDash` : Sets the pattern for dashed lines in a graphics context
- `CGContextSetStrokeColorSpace` : Sets the stroke color space in a graphics context
- `CGContextSetStrokeColor` : Sets the current stroke color
- `CGContextSetStrokePattern` : Sets the stroke pattern in the specified graphics context

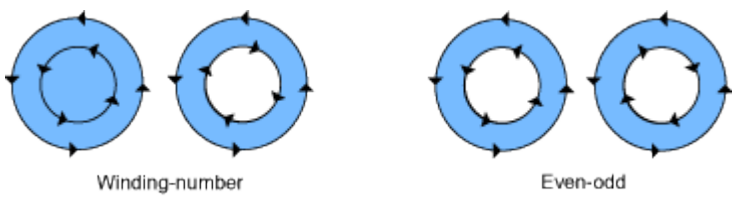
Functions for stroking a path :

- `CGContextStrokePath` : Strokes the current path
- `CGContextStrokeRect` : Strokes the specified rectangle
- `CGContextStrokeRectWithWidth` : Strokes the specified rectangle, using the specified line width
- `CGContextStrokeEllipseInRect` : Strokes an ellipse that fits inside the specified rectangle
- `CGContextStrokeLineSegments` : Strokes a sequence of lines

There are two ways Quartz can calculate the fill area. Simple paths such as ovals and rectangles have a well-defined area. But if your path is composed of overlapping segments, such as the concentric circles there are two rules you can use to determine the fill area:

- The default fill rule is called the nonzero winding number rule : To determine whether a specific point should be painted, start at the point and draw a line beyond the bounds of the drawing. Starting with a count of 0, add 1 to the count every time a path segment crosses the line from left to right, and subtract 1 every time a path segment crosses the line from right to left. If the result is 0, the point is not painted. Otherwise, the point is painted

- The even-odd rule : To determine whether a specific point should be painted, start at the point and draw a line beyond the bounds of the drawing. Count the number of path segments that the line crosses. If the result is odd, the point is painted. If the result is even, the point is not painted

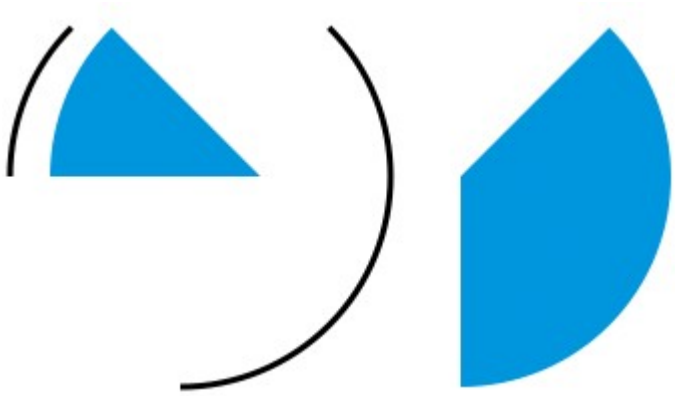


The following functions are used to fill a path :

- `GContextEOFillPath` : Fills the current path using the even-odd rule
- `CGContextFillPath` : Fills the current path using the non-zero winding number rule
- `CGContextFillRect` : Fills the area that fits inside the specified rectangle
- `CGContextFillRects` : Fills the areas that fits inside the specified rectangles
- `CGContextFillEllipseInRect` : Fills an ellipse that fits inside the specified rectangle.

How do we draw some specific shape ?

Arcs



A routine that constructs an arc path :

```
  void pathForArc (CGContextRef context, CGRect r, int startAngle, int arcAngle)
  {
      float start, end;
CGContextSaveGState(context);
      CGContextTranslateCTM(context, r.origin.x + r.size.width/2, r.origin.y +
r.size.height/2);
      CGContextScaleCTM(context, r.size.width/2, r.size.height/2);
      if (arcAngle > 0)
{
          start = (90 - startAngle - arcAngle) * M_PI / 180;
          end = (90 - startAngle) * M_PI / 180;
      }
      else
{
          start = (90 - startAngle) * M_PI / 180;
          end = (90 - startAngle - arcAngle) * M_PI / 180;
      }

      CGContextAddArc (context, 0, 0, 1, start, end, false);
      CGContextRestoreGState(context);
  }
```

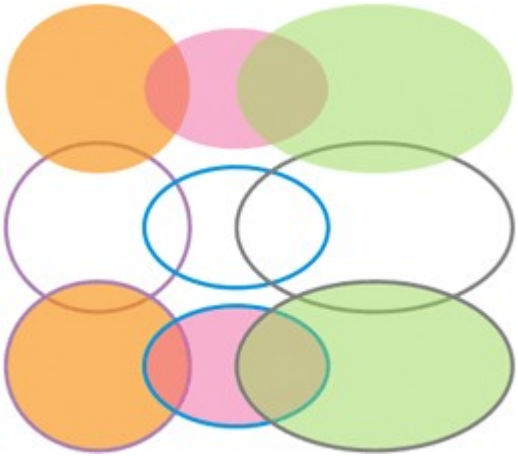
A routine that strokes an arc :

```
  void strokeArc(CGContextRef context, CGRect r, int startAngle, int arcAngle)
  {
      CGContextBeginPath (context);
      pathForArc (context,r,startAngle,arcAngle);
      CGContextStrokePath(context);
  }
```

A routine that fills an arc

```
  void fillArc (CGContextRef context, CGRect r, int startAngle, int arcAngle)
  {
      CGContextBeginPath (context);
      CGContextMoveToPoint (context, r.origin.x + r.size.width/2, r.origin.y +
r.size.height/2);
      pathForArc (context,r,startAngle,arcAngle);
      CGContextClosePath (context);
      CGContextFillPath (context);
  }
```

Ovales



A routine that constructs an oval path :

```
  void addOvalToPath(CGContextRef context, CGRect r)
  {
      CGContextSaveGState(context);
      CGContextTranslateCTM(context, r.origin.x + r.size.width/2,r.origin.y +
r.size.height/2);
      CGContextScaleCTM(context, r.size.width/2, r.size.height/2);
      CGContextBeginPath(context);
      CGContextAddArc(context, 0, 0, 1, 0, 2*pi, true);
      CGContextRestoreGState(context);
  }
```

A routine that fills an oval :

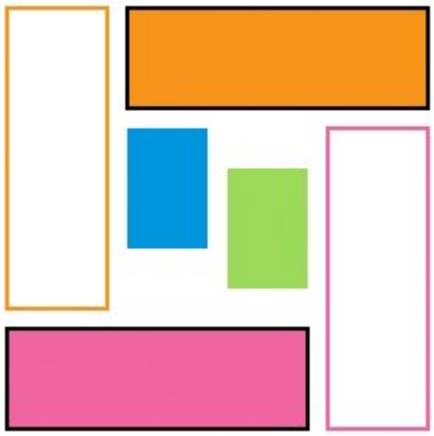
```
  void fillOval(CGContextRef context, CGRect r)
  {
      /* Define the color here */
      CGContextSetRGBFillColor (context, 1, 0, 0, 1);
      addOvalToPath (context,r);
      CGContextFillPath (context);
  }
```

A routine that strokes an oval :

```
  void strokeOval(CGContextRef context, CGRect r)
  {
      /* Define the color here */
      CGContextSetRGBStrokeColor(context, 1, 0, 0, 1);
      addOvalToPath(context,r);
```

```
    CGContextStrokePath(context);
}
```

Rectangles



You can simply use the Quartz functions `CGContextStrokeRect` and `CGContextFillRect`.

```
void CGContextStrokeRect (CGContextRef context, CGRect rect);

void CGContextFillRect (CGContextRef context, CGRect rect);
```

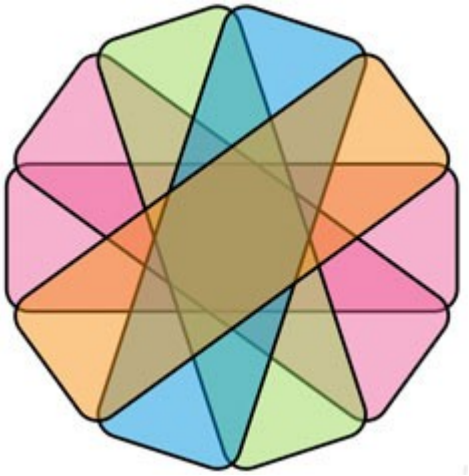
A routine that strokes a rectangle :

```
void strokeRectangle(CGContextRef context, CGRect r)
{
    /* Define the color here */
    CGContextSetRGBStrokeColor(context, 1, 0, 0, 1);
    CGContextStrokeRect (context, r);
}
```

A routine that fills a rectangle :

```
void fillRectangle(CGContextRef context, CGRect r)
{
    /* Define the color here */
    CGContextSetRGBStrokeColor(context, 1, 0, 0, 1);
    CGContextFillRect (context, r);
}
```

Rounded Rectangles



A routine that constructs a rounded rectangle path :

```
  void addRoundedRectToPath(CGContextRef context, CGRect rect, float
ovalWidth,float ovalHeight)
  {
      float fw, fh;
      if (ovalWidth == 0 || ovalHeight == 0)
      {
      CGContextAddRect(context, rect);
      return;
      }

      CGContextSaveGState(context);
      CGContextTranslateCTM (context, CGRectGetMinX(rect), CGRectGetMinY(rect));
      CGContextScaleCTM (context, ovalWidth, ovalHeight);
      fw = CGRectGetWidth (rect) / ovalWidth;
      fh = CGRectGetHeight (rect) / ovalHeight;
      CGContextMoveToPoint(context, fw, fh/2);
      CGContextAddArcToPoint(context, fw, fh, fw/2, fh, 1);
      CGContextAddArcToPoint(context, 0, fh, 0, fh/2, 1);
      CGContextAddArcToPoint(context, 0, 0, fw/2, 0, 1);
      CGContextAddArcToPoint(context, fw, 0, fw, fh/2, 1);
      CGContextClosePath(context);
      CGContextRestoreGState(context);
  }
```

A routine that strokes a rounded rectangle :

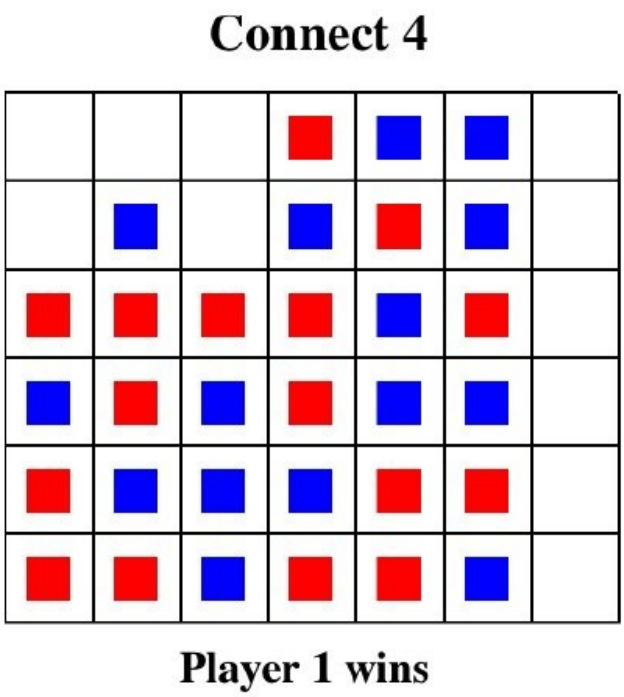
```
  void strokeRoundedRect(CGContextRef context, CGRect rect, float ovalWidth,
float ovalHeight)
```

```
  {
      CGContextBeginPath(context);
      addRoundedRectToPath(context, rect, ovalWidth, ovalHeight);
      CGContextStrokePath(context);
  }
```

A routine that fills a rounded rectangle :

```
  void fillRoundedRect (CGContextRef context, CGRect rect, float ovalWidth,
float ovalHeight)
  {
      CGContextBeginPath(context);
      addRoundedRectToPath(context, rect, ovalWidth, ovalHeight);
      CGContextFillPath(context);
  }
```

In Connect 4:



Let's take a look on the drawing functions of the game Connect 4. These functions are implemented in *draw.c* :

- `DrawARect` : function used to draw a rectangles
- `DrawGrid` : function used to draw the grid
- `DrawToken` : function used to draw each piece (blue or red) of the game
- `MyDrawText` : function used to a specific text in the window

The prototype of theses functions are the following :

```
void DrawARect(WindowRef window, CGRect rect,float color[4]);
void DrawGrid(WindowRef window);
void MyDrawText (WindowRef window, char * text,int x,int y,int size);
void DrawToken(WindowRef window, int player, int Column, int Line);
```

2. Carbon in VCL

All the functions used for creating or manipulating a window are implemented is /vcl/aqua/window/salframe.cxx. The class SalFrame is an empty box (technically an abstract class with pure virtual methods) which is used to create a specific class in accordance with the OS used. *salframe.hxx* is included in *salframe.h* existing for every single OS and which are included itself into s*alframe.cxx* existing for every single OS too.

How does vcl create a window ?

In order to create a window, the function `CreateNewSystemWindow()` has to be called.

`CreateNewSystemWindow()` :

The parameters are :

- pParent: Handler on an existing window.
- nSalFrameStyle: Give the window class and window attributes for the new window. Possible value: *(Possibility to combine: «attribute_1 | attribute_2»)*

SAL_FRAME_STYLE_DEFAULT
- Class: kDocumentWindowClass
- Attributes: kWindowStandardHandlerAttribute
  kWindowStandardDocumentAttributes

SAL_FRAME_STYLE_MOVEABLE
- Class: kDocumentWindowClass
  or kMovableModalWindowClass
- Attributes kWindowStandardHandlerAttribute
  kWindowCollapseBoxAttribute

SAL_FRAME_STYLE_SIZEABLE
- Class: kDocumentWindowClass
- Attributes: kWindowStandardHandlerAttribute
  kWindowResizableAttribute
  kWindowLiveResizeAttribute
  (kWindowFullZoomAttribute)

SAL_FRAME_STYLE_CLOSEABLE
- Class: kPlainWindowClass
- Attributes: kWindowStandardHandlerAttribute
  kWindowCloseBoxAttribute

SAL_FRAME_STYLE_NOSHADOW
- Class: kPlainWindowClass
- Attributes kWindowStandardHandlerAttribute
  kWindowNoShadowAttribute

SAL_FRAME_STYLE_TOOLTIP
- Class: kPlainWindowClass

Attributes: kWindowStandardHandlerAttribute

SAL_FRAME_STYLE_OWNERDRAWDECORATION
Class: kPlainWindowClass
Attributes: kWindowStandardHandlerAttribute

SAL_FRAME_STYLE_DIALOG
Class: kPlainWindowClass
Attributes: kWindowStandardHandlerAttribute

SAL_FRAME_STYLE_CHILD
Class: kPlainWindowClass
Attributes: kWindowStandardHandlerAttribute

SAL_FRAME_STYLE_FLOAT
Class: kPlainWindowClass
Attributes: kWindowStandardHandlerAttribute

SAL_FRAME_STYLE_TOOLWINDOW
Class: kFloatingWindowClass
Attributes: kWindowStandardHandlerAttribute

SAL_FRAME_STYLE_INTRO
Class: kPlainWindowClass
Attributes: kWindowStandardHandlerAttribute

- Create a window and put the window reference in the variable : `mhWnd`
- Define window areas:

fullWindowRect:
Defined in GetOptimalWindowSize() function
Height: 400px
Width: 400px
Position of the top left corner: (100,100)

contentRect:
Same as kWindowContentRgn in Carbon

titleBarRect
Same as kWindowTitleBarRgn in Carbon

maGeometry: SalFrameGeometry DataStructure
nX, nY:
Top left corner of contentRect
nLeftDecoration, nRightDecoration, nTopDecoration, nBottomDecoration:
Border Decorations
nWidth, nHeight:
Size of contentRect

- Event handlers :

All following event types are installed on each new window. They are installed and registered throw the funtion InstallAndRegisterEventHandler() and uninstall and unregister with DeinstallAndUnregisterAllEventHandler()

windowBoundsChangedEvent

Description: Indicates that the window has been moved or resized
Even Class: kEventClassWindow
Event Kind: kEventWindowBoundsChanged

windowCloseEvent

Description: Sent by the standard window handler after it has received kEventWindowClickCloseRgn and successfully called TrackBox. Applications might intercept this event to check if the document is dirty, and display a Save/Don'tSave/Cancel alert.
Even Class: kEventClassWindow
Event Kind: kEventWindowClose

windowActivatedEvent

Desciption: The window is active now. Sent to any window that is activated, regardless of whether the window has the standard window handler installed.
Even Class: kEventClassWindow
Event Kind: kEventWindowActivated

windowPaintEvent

Desciption: Sent when it is time to draw the entire window (such as when the window is first displayed). This is a convenience event that gives you a chance to draw all the window elements at once. (not use for the moment)
Even Class: kEventClassWindow
Event Kind: kEventWindowPaint

windowDrawContentEvent

Desciption: Higher-level update event sent only if you have the standard window handler installed.
Even Class: kEventClassWindow
Event Kind: kEventWindowDrawContent

mouseUpDownEvent[]

Desciption: A mouse button was pressed
Even Class: kEventClassMouse
Event Kind: kEventMouseDown

Desciption: A mouse button was released
Even Class: kEventClassMouse
Event Kind: kEventMouseUp

cWindowResizeStarted

Desciption: Indicates that the user has just started to resize a window. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value. The standard window handler ignores this event.
Even Class: kEvenClassWindow
Event Kind: kEventWindowResizeStarted

cWindowResizeCompleted

Desciption: Indicates that the user has just finished resizing a window. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value. The standard window handler ignores this event.
Even Class: kEvenClassWindow
Event Kind: kEventWindowResizeCompleted

How does VCL draw element in the window ?

All the functions used to draw in a window are implemented in the file *salgdi.*cxx. Remember that the class SalGraphics, defined in *salgdi.hxx* is an empty box (technically an abstract class with pure virtual methods) which is used to create a specific for each OS used. Therefore, *salgdi.hxx* is included in *salgdi.h* existing for every single OS. *salgdi.h is* included into *salgdi.cxx*.

This class is the an abstract data type which can not be instantiated. It is completed for each OS in following files:

- **Windows**

  Inherited class: WinSalGraphics
  /vcl/win/inc/salgdi.h
  /vcl/win/source/gdi/salgdi.cxx

- **X11**

  Inherited class: X11SalGraphics
  /vcl/unx/inc/salgdi.h
  /vcl/unx/source/gdi/salgdi.cxx

- **Aqua**

  Inherited class: SalGraphics
  /vcl/aqua/inc/salgdi.h
  /vcl/aqua/source/gdi/salgdi.cxx

Let's take a look to some functions of *salgdi.cxx* (note that as the aqua version is still in development, all the functions have not yet been implemented:

```
void AquaSalGraphics::drawLine( long nX1, long nY1, long nX2, long nY2 )
{
        if ( BeginGraphics() )
        {
        CGContextBeginPath( mrContext );
        CGContextMoveToPoint( mrContext, nX1, nY1 );
        CGContextAddLineToPoint( mrContext, nX2, nY2 );
        CGContextDrawPath( mrContext, kCGPathStroke );

                EndGraphics();
        }
}
```

We can see that this is the classical functions which are used to draw a line. We first create a path, as we have seen before and then we stroke this path.

Another example of drawing function. This function draw rectangle :

```
void AquaSalGraphics::drawRect( long nX, long nY, long nWidth, long nHeight )
{
        if ( BeginGraphics() )
        {
                if ( IsBrushTransparent() )
                        CGContextStrokeRect (mrContext, CGRectMake (nX, nY,
nWidth, nHeight));
                else
                        CGContextFillRect (mrContext, CGRectMake (nX, nY,
nWidth, nHeight));

                EndGraphics();
        }
}
```

This two function use two functions called BeginGraphics() and EndGraphics() which are implemented in *salgdiutils.cxx*

```
bool AquaSalGraphics::BeginGraphics ()
{
        if( mrWindow != NULL )
        {
                SetPortWindowPort (mrWindow);
                if( noErr == QDBeginCGContext (GetWindowPort (mrWindow),
&mrContext))
                {
                        // switch to HIView coordinate system, i.e. (0,0) is
top-left
                        Rect windowBounds;
                        GetWindowPortBounds ( mrWindow, &windowBounds);
            /*
                        fprintf(stderr, "windowPortBounds: left: %d top: %d
width: %d height: %d\n",
                                        windowBounds.left, windowBounds.top,
                    windowBounds.right - windowBounds.left,
                    windowBounds.bottom - windowBounds.top);
                        */
                        CGContextTranslateCTM (mrContext, 0, windowBounds.bottom
- windowBounds.top);
                        CGContextScaleCTM (mrContext, 1.0, -1.0);

            // set up clipping area
            if( mrClippingPath )
            {
                CGContextBeginPath( mrContext );                // discard any
existing path
                CGContextAddPath( mrContext, mrClippingPath );  // set the
current path to the clipping path
                CGContextClip( mrContext );                     // use it for
clipping
            }

            // set RGB colorspace and line and fill colors
            CGContextSetFillColorSpace( mrContext, mrRGBColorSpace );
            CGContextSetFillColor( mrContext, mpFillColor );
            CGContextSetStrokeColorSpace( mrContext, mrRGBColorSpace );
            CGContextSetStrokeColor( mrContext, mpLineColor );

                        return true;
                }
                else
                {
```

```
                        //fprintf(stderr, "QDBeginCGContext() error\n");
      }
        }
        else
        {
                //fprintf(stderr, "BeginGraphics: mhWindow == NULL !\n");
                return false;
        }
}
```

We can see that this function is responsible of the creation of a Quartz context inside QuickDraw. For this the function `QDBeginCGContext` is used. This function defines too, the colors used to fill and stroke path `CGContextSetFillColor()` and `CGContextSetStrokeColor()`

On the other hand, the funcition `EndGraphics()` forces all drawing operations in a window context to be rendered immediately to the destination device buy using the function `CGContextFlush()`. After that, the Quart context is closed.

```
bool AquaSalGraphics::EndGraphics ()
{
        if( mrContext != NULL && mrWindow != NULL )
        {
                CGContextFlush(mrContext);
                QDEndCGContext (GetWindowPort(mrWindow), &mrContext);
        }
        return true;
}
```

Conclusion

OpenOffice.org is becoming a very popular software and not only on the opensource community. Indeed, today, more and more people are interested in this new kind of software : both efficiency and free at the same time. But to attract more and more users, the community have to innovate again and again. Programmers have to create faster and simplier software to attract new users. Nevertheless, one of the easiest way to attract more people is certainly the creation of versions for each platform. Today, almost each platform has its version of OpenOffice.org, Windows, Linux, Unix. Therefore, porting OpenOffice.org on Mac OS X Aqua is one of these innovation which can bring a lot of new users. If we want to run OpenOffice.org on a new platform, we have to update VCL, the graphic engine of OpenOffice.org. It is very important to understand that VCL is the heart of this software. In this report, and with the help of our little application, we have tried to explain how the Carbon API works and how it has been implemented in VCL to allow OpenOffice.org to run on Mac OS X without X11. Nevertheless, there is always things to improve. If we look at the VCL structure, for each new platform added, the VCL increase in size. Maybe, a good way to improve VCL is to rethink him by using an API which exist on all the plateform, like for instance Gtk, or Qt. But this is another story... and another TX or maybe ... a summer of Code..., well who knows...

Bibliography

Websites :

- http://www.openoffice.org
- http://developer.apple.com
- ftp://eric.bachard.free.fr

In a nutshell

Keywords :

- OpenOffice.org
- Apple
- Carbon
- OpenSource
- API
- Programmation
- Linux
- CVS
- Project
- Compilation

Summary :

Nowaydays, the project OpenOffice.org is one of the biggest project of the opensource community. Available for the principle OS, like Linux, Mac OS X and Windows, it is a perfect example of what the opensource community is capable of. OpenOffice.org is becoming very popular and not only in the opensource community. But, as any other sofware there is always something to do in order to improve it. For instance, let's take the case of the Mac OS X version of OpenOffice.org. Today, OpenOffice.org won't running without X11, the graphic server of all the Unix family system. By this simple observation, some developers have decided to create a version of OpenOffice.org using instead the graphic server of Mac OS X called Quartz. If we want to run OpenOffice.org on a new platform, we have to update VCL. VCL, for Visual Class Library is the graphics engine of OpenOffice.org. Without it, you have nothing on your screen. It is very important to understand that VCL is the heart of this software. In this report, and with the help of a little application written in Carbon, we have tried to show how the Carbon API works and how it has been implemented in VCL to allow OpenOffice.org to run on Mac OS X without X11.

ANNEXE 1
WINDOW CLASSES

kAlertWindowClass	Identifies an alert box window.
kMovableAlertWindowClass	Identifies a movable alert box window.
kModalWindowClass	Identifies a modal dialog box window.
kMovableModalWindowClass	Identifies a movable modal dialog box window.
kFloatingWindowClass	Identifies a window that floats above all document windows. If your application assigns this constant to a window, the Window Manager ensures that the window has the proper floating behavior.
kDocumentWindowClass	Identifies a document window or modeless dialog box window.
kUtilityWindowClass	Identifies a utility window.
kHelpWindowClass	Identifies a window used by Apple Help.
kSheetWindowClass	Identifies a sheet.
kToolbarWindowClass	
kPlainWindowClass	
kOverlayWindowClass	
kSheetAlertWindowClass	Identifies an alert sheet.
kAltPlainWindowClass	
kDrawerWindowClass	Identifies a drawer
kAllWindowClasses	Specifier used to designate all window classes.

ANNEXE 2
WINDOW ATTRIBUTES

kWindowNoAttributes
: If no bits are set, the window has none of the following attributes.

kWindowCloseBoxAttribute
: If the bit specified by this mask is set, the window has a close box.

kWindowHorizontalZoomAttribute
: If the bit specified by this mask is set, the window has a horizontal zoom box.

kWindowVerticalZoomAttribute
: If the bit specified by this mask is set, the window has a vertical zoom box.

kWindowFullZoomAttribute
: If the bits specified by this mask are set, the window has a full—horizontal and vertical—zoom box.

kWindowCollapseBoxAttributeRunApplicationEventLoop();
: If the bit specified by this mask is set, the window has a collapse box.

kWindowResizableAttribute
: If the bit specified by this mask is set, the window has a resize tab/box and is resizable.

kWindowSideTitlebarAttribute
: If the bit specified by this mask is set, the window has a side title bar. This attribute may be applied only to floating windows, that is, those windows assigned the window class constant kFloatingWindowClass. See “Window Class Constants” for a description of this constant.

kWindowToolbarButtonAttribute
: If the bit specified by this mask is set, the window has a toolbar button. This oblong clear button shows and hides the toolbar.

kWindowMetalAttribute
: If the bit specified by this mask is set, the window has a brushed-metal appearance.

kWindowNoUpdatesAttribute
: If the bit specified by this mask is set, the window does not receive update events.

kWindowNoActivatesAttribute
: If the bit specified by this mask is set, the window does not receive activate events.

kWindowOpaqueForEventsAttribute
: If the bit specified by this mask is set, the window does not receive any events.

kWindowCompositingAttribute
: If the bit specified by this mask is set, the window uses HIView-based compositing.

kWindowNoShadowAttribute

kWindowHideOnSuspendAttribute

kWindowStandardHandlerAttribute
: If the bit specified by this mask is set, the window supports the standard window event handler. The standard event handler provides standard actions for common window events. See Inside Mac OS X: Handling Carbon Events for more details.

kWindowHideOnFullScreenAttributeRunApplicationEventLoop();

kWindowInWindowMenuAttribute
: If the bit specified by this mask is set, the window title appears in the system-generated Window menu.

kWindowLiveResizeAttribute
: If the bit specified by this mask is set, the window supports live resizing.

kWindowIgnoreClicksAttribute

kWindowNoConstrainAttribute

kWindowStandardDocumentAttributes
: If the bits specified by this mask are set, the window has the attributes of a standard document window—that is, a close box, full zoom box, collapse box, and size box.

kWindowStandardFloatingAttributes
: If the bits specified by this mask are set, the window has the attributes of a standard floating window—that is, a close box and collapse box.

ANNEXE 3
MENU ATTRIBUTES

kMenuItemAttrDisabled — This menu item is disabled.

kMenuItemAttrIconDisabled — This menu item's icon is disabled.

kMenuItemAttrSubmenuParentChoosable — The user can select the parent item of a submenu.

kMenuItemAttrDynamic

This menu item changes dynamically based on the state of the modifier keys. For example, holding down the command key might change the menu item from "Select widget" to "Select all widgets."

When a menu item has alternate dynamic states, you should group them together sequentially in the menu and assign them the same command key. A collection of menu item alternates is called a dynamic group.

kMenuItemAttrNotPreviousAlternate

This item is not part of the same dynamic group as the previous item. The Menu Manager determines which menu items belong to a dynamic group by examining the command keys of each item; if a menu item has the same command key as the previous item, the Menu Manager considers it to be part of the same dynamic group.

However, in some cases you may have sequential items with the same command key (or no command key at all) that should not be considered part of the same dynamic group. To distinguish the separation, you should set this flag for the first menu item in the new group.

kMenuItemAttrHidden

The menu item is not drawn when displaying the menu. The item is also not included in command-key matching unless the kMenuItemAttrDynamic or kMenuItemIncludeInCmdKeyMatching attribute is set.

kMenuItemAttrSeparator

The menu item is a separator; any text in the item is ignored.

kMenuItemAttrSectionHeader

The menu item is a menu section header; this item is disabled and not selectable.

kMenuItemAttrIgnoreMeta

Ignore the dash (-) metacharacter in this menu item. Dashes at the beginning of a menu item title traditionally signify that the menu item is a separator. However, in some cases you might want to display the dash in the title (for example, if you wanted the menu item to read "-40 degrees F.")

kMenuItemAttrAutoRepeat

The IsMenuKeyEvent event function recognizes this menu item when it receives an autorepeat keyboard event.

kMenuItemAttrUseVirtualKey

When MenuEvent and IsMenuKeyEvent compare this menu item's keyboard equivalent against a keyboard event, they use the item's virtual keycode equivalent rather than its character code equivalent.

kMenuItemAttrCustomDraw

This is a custom menu item. Setting this attribute causes custom menu item drawing Carbon events to be sent to your application.

kMenuItemAttrIncludeInCmdKeyMatching

If this attribute is set, functions such as MenuKey, MenuEvent and IsMenuKeyEvent examine this menu item during command key matching. Typically, visible items are examined and hidden items (unless they have the kMenuItemAttrDynamic attribute set) are ignored during command key matching. However, by setting this attribute, you can force hidden items to be included in the matching

kMenuItemAttrAutoDisable

Disables the menu item if it does not respond to the kEventCommandUpdateStatus event . That is, if no kEventCommandUpdateStatus handler is installed on this item, or if all the handlers installed for the update event return eventNotHandledErr, this item is automatically disabled. This attribute is useful if your application uses the kEventCommandUpdateStatus event to enable menu items; for example you no longer have to install an update status handler on the application target to disable menu items when there are no document windows open.

kMenuItemAttrUpdateSingleItem

Update only the menu item that matches when searching available command keys. Normally when the Menu Manager does command key matching, it sends a kEventMenuEnableItems event to the menu containing the matching item and then sends a kEventCommandUpdateStatus to each item in the menu. Doing so can be inefficient, since in most cases only the item that matches needs to be updated. By setting this attribute, only the matching item receives the update event and kEventMenuEnableItems is not sent to the menu. If your application enables menu items solely through kEventCommandUpdateStatus event handlers, you should set this attribute for your menu items.

ANNEXE 4
Event Classes

Event Class	Constant Descriptions
kEventClassMouse	Events related to the mouse (mouse down/up/moved).
kEventClassKeyboard	Events related to the keyboard.
kEventClassTextInput	Events related to text input (by keyboard or by input method).
kEventClassApplication	Application-level events (launch, quit, and so on.).
kEventClassAppleEvent	Apple Events.
kEventClassMenu	Menu-related events.
kEventClassWindow	Window-related events.
kEventClassControl	Control-related events.
kEventClassCommand	Command events (HICommands).
kEventClassTablet	Events related to tablet input.
kEventClassVolume	Events related to File Manager volumes.
kEventClassAppearance	Events related to the Appearance Manager.
kEventClassService	Events related to the Services Manager.
kEventClassToolbar	Events related to the toolbar (not the toolbar window class).
kEventClassToolbarItem	Events related to toolbar items.
kEventClassAccessibility	Events related to application accessibility features.

ANNEXE 5
Connect 4 : Sourcecode

main.c

```
#include "include.h"
#include "divers/draw.h"
#include "window/window.h"
#include "event/event.h"
#include "game.h"

int main(int argc, char* argv[])
{

	p4_t* p4=malloc(sizeof(p4_t));
		p4->player=1;
		p4->window=NULL;
		p4->game=INMENU;
		init_tab(p4->t);
		p4->Ptype=CP;


	// Create the main window
	p4->window = p4CreateNewWindow();

	InstallMouseEvent(p4);

	// Display the grid of the P4
	DrawGrid(p4->window);

	//Display the title
	MyDrawText(p4->window, "Connect 4", 200, 450,35);


	// Run the event loop
	RunApplicationEventLoop();

	return 1;
}
```

include.h

```
#ifndef __INCLUDE_H__
#define __INCLUDE_H__

#include <Carbon/Carbon.h>

// Game Constants
#define INGAME 0
#define INMENU 1
#define CP 1
#define HUM 0

// Window Constants
#define WindowWidth 600
#define WindowHeight 500

// Grid Constants
#define Grid_spacing 60
#define Grid_NbColumn 7
#define Grid_NbLine 6
```

```
#define Grid_BorderWidth 2
#define Grid_x 60
#define Grid_y 60

typedef struct {
            WindowRef window;
            short int game;
            int Ptype;
            int player;
            char t[Grid_NbLine][Grid_NbColumn];

            } p4_t;

#endif
```

window.c

```
#include "window.h"

WindowRef p4CreateNewWindow()

{
            WindowRef         window=NULL;

            WindowClass windowClass = kDocumentWindowClass;
                  WindowAttributes  attributes =
kWindowStandardHandlerAttribute;
                              attributes  |= kWindowStandardFloatingAttributes ;
                              // attributes  =
kWindowStandardDocumentAttributes;
                              // attributes |= kWindowLiveResizeAttribute;
            // Window size
            Rect contentBounds;
            // Set content rectangle  order : Left,Top,Right,Bottom
            SetRect (&contentBounds, 100,100,100+WindowWidth,100+WindowHeight);

            CreateNewWindow (windowClass,attributes,&contentBounds,&window);

            // Diplay the window
            ShowWindow( window );


            return window;
}
```

event.c

```
#include "event.h"

pascal OSStatus mouse_event(EventHandlerCallRef handlerRef, EventRef event, p4_t
*p4 )
{
      int column, line;
      Point wheresMyMouse;
      // Give the Mouse position in the var wheresMyMouse from the top left
corner of the screen
      GetEventParameter ( event, kEventParamMouseLocation, typeQDPoint, NULL,
sizeof(Point), NULL, &wheresMyMouse);

      Rect globalBounds;
      // Give the position of the window in the var globalBounds
      GetWindowBounds(p4->window,kWindowContentRgn,&globalBounds);
      // Calculate new-coordinate
      wheresMyMouse.h-=globalBounds.left;
```

```
	wheresMyMouse.v-=globalBounds.top;

	column=WhichColumn(wheresMyMouse.h,wheresMyMouse.v);
	line=WhichLine(wheresMyMouse.h,wheresMyMouse.v);

	if(column!=-1 && line!=-1 && p4->t[line][column]==' ' &&  ( line==0 || p4->t[line-1][column]!=' ' ) )
	{
		DrawToken(p4->window,p4->player, column, line);
		//PLAYER 1
		if (p4->player==1)
		{
			p4->t[line][column]='X';
			if(win(p4->t,p4->player))
			{
				MyDrawText (p4->window, "Player 1 wins" ,180,20,30);
				RemoveEventHandler((EventHandlerRef)handlerRef);
			}

		// COMPUTER
			else if( p4->Ptype==CP)
			{
				int i,k,Bot;
				// Search the best place to put the token
				do
				{
					Bot=bot(p4->t,2);
					k=0;
					i=0;
					while (i<=5 && k==0)
					{
						if (p4->t[i][Bot]=='X' || p4->t[i][Bot]=='O') {i++;}
						else {k=1;}
					}

				} while (k==0);

				p4->t[i][Bot]='O';
				DrawToken(p4->window,2, Bot, i);

				if(win(p4->t,2))
				{
					MyDrawText (p4->window, "Computer wins" ,180,20,30);
					RemoveEventHandler((EventHandlerRef)handlerRef);
				}
			}
			else
				p4->player=2;
		}

		//PLAYER 2
		else
		{
			p4->t[line][column]='O';

			if(win(p4->t,p4->player))
			{
				MyDrawText (p4->window, "Player 2 wins" ,180,20,30);
				RemoveEventHandler((EventHandlerRef)handlerRef);
			}

			p4->player=1;
		}
	}
```

```
      // Removes all events from the main event queue.
      FlushEventQueue(GetMainEventQueue ());

      // Now propagate the event to the next handler
      CallNextEventHandler( handlerRef, event);

      return noErr;
}

void InstallMouseEvent(p4_t* p4)
{
      EventTypeSpec     eventType;
            eventType.eventClass = kEventClassMouse;          // Set event class
            eventType.eventKind  = kEventMouseDown;          // Set event kind
      InstallWindowEventHandler(p4->window,
NewEventHandlerUPP((EventHandlerProcPtr)mouse_event), 1, &eventType, p4 , NULL);
}
```

draw.c

```
#include "draw.h"

void DrawARect(WindowRef window, CGRect rect,float color[4])
{
      // color[4] {red,green,blue,alpha}

      CGContextRef myContext;

    SetPortWindowPort (window);

      Rect globalBounds;
      // give the position of the window in the var globalBounds
      GetWindowBounds(window,kWindowContentRgn,&globalBounds);

    QDBeginCGContext (GetWindowPort (window), &myContext);

        CGContextSetRGBFillColor (myContext,
color[0],color[1],color[2],color[3]);
        CGContextFillRect (myContext, rect);

        CGContextFlush(myContext);

     QDEndCGContext (GetWindowPort(window), &myContext);
}

void DrawToken(WindowRef window, int player, int Column, int Line)
{

      int color[4]={0,0,0,1};//{red,green,blue,alpha}
      if (player==1) color[0]=1;  //red
      else color[2]=1; //blue

      CGRect rect = CGRectMake(Grid_x+Column*Grid_spacing +(Grid_spacing-
30)/2,Grid_y+Line*Grid_spacing+(Grid_spacing-30)/2,30,30);

      CGContextRef myContext;

    SetPortWindowPort (window);

      Rect globalBounds;
      // give the position of the window in the var globalBounds
      GetWindowBounds(window,kWindowContentRgn,&globalBounds);
```

```
    QDBeginCGContext (GetWindowPort (window), &myContext);

        CGContextSetRGBFillColor (myContext,
color[0],color[1],color[2],color[3]);
        CGContextFillRect (myContext, rect);

        CGContextFlush(myContext);

     QDEndCGContext (GetWindowPort(window), &myContext);


}


void DrawGrid(WindowRef window)
{
      float color[4]={0,0,0,1};
      int i=0;
      //Horizontal lines
      for(i=0;i<Grid_NbLine+1;i++)
            DrawARect(window, CGRectMake( Grid_x  ,  Grid_y + Grid_spacing*i  ,
Grid_NbColumn * Grid_spacing  ,  Grid_BorderWidth )  ,  color );

      //vertical line
      for(i=0;i<Grid_NbColumn+1;i++)
            DrawARect(window, CGRectMake(Grid_x + Grid_spacing*i  ,  Grid_y
,Grid_BorderWidth  ,  Grid_NbLine * Grid_spacing )  ,  color );

}

void MyDrawText (WindowRef window, char * text,int x,int y,int size)

{
      CGContextRef myContext;

      QDBeginCGContext (GetWindowPort (window), &myContext);
            CGContextSelectFont (myContext,"Times-Bold", size,
kCGEncodingMacRoman);
            CGContextShowTextAtPoint (myContext, x, y, text , strlen(text));

            CGContextFlush(myContext);

     QDEndCGContext (GetWindowPort(window), &myContext);


}
```

Public Documentation License Notice

The contents of this Documentation are subject to the Public Documentation License Version 1.0 (the "License"); you may only use this Documentation if you comply with the terms of this License. A copy of the License is available at *http://www.openoffice.org/licenses/PDL.html* .

The Original Documentation is Carbon_VCL.odt **.** The Initial Writers of the Original Documentation is Barb Yann and Mohr Alexandre Copyright © 2006 *[*. All Rights Reserved. (Initial Writer contact(s): amohr3010@gmail.com, yann.barb@wanadoo.fr).

Contributor(s): _____.

Portions created by _____ are Copyright © _____ *[Insert year(s)]* . All Rights Reserved. (Contributor contact(s):_____ *[Insert hyperlink/alias]*).

NOTE: The text of this Appendix may differ slightly from the text of the notices in the files of the Original Documentation. You should use the text of this Appendix rather than the text found in the Original Documentation for Your Modifications.