OxJapi: an Ox version of Merten Joost's Java Application Programming Interface

Christine Choirat[*] Raffaello Seri[†]

November 2002 *(Japi version 1.0.5 r 2)*

We have ported M. Joost's Japi package (version `1.0.5 r 2`, see [4]) to Ox. It can be found on the site:

`http://site.voila.fr/choirat/software/oxjapi/oxjapi.html`

We thank Jurgen A. Doornik for his constant support and for his invaluable help with the Windows version.

1 Introduction and motivations

Ox is a well-known object-oriented matrix programming language written by J. A. Doornik (see [1]). The easiest and probably most powerful way to use it is done through GiveWin (see `http://www.oxmetrics.net`). In particular, GiveWin allows for interactive manipulation of graphics and for writing in a very simple way a graphical user interface (as submenus of GiveWin). This is done with the 'OxPack' tool, as described in Chapter 11 of [3] and in Chapter A3 of [2]. But in some cases, this approach is not possible, for example when:

- Your program cannot be written through the 'Modelbase' class.
- GiveWin is not installed on your system.
- Your code has to work on various platforms (Windows, Linux, ...).

1.1 Adding an interface to Ox

As Doornik explains in the Ox Appendix (Ox version 3.1, p. 11, see [2]):

> Ox is limited in terms of user interaction, only providing console style input using the `scan` function. [...] there are no plans to make

[*]Centre de Recherche Viabilité, Jeux, Contrôle, Université Paris Dauphine, 75775 PARIS CEDEX 16, FRANCE, email: `christine.choirat@voila.fr`

[†]CREST-LFA, Timbre J320, 15 bd Gabriel Péri, 92245 MALAKOFF CEDEX, FRANCE, email: `seri@ensae.fr`

> interface components an intrinsic part of Ox: this would always lag behind the latest developments. [...] Various approaches could be considered to add a user interface:
>
> (1) Write a separate program which creates an input file.
>
> (2) Write a separate program which generates an Ox Source file.
>
> (3) Write a DLL which exports dialogs to be used in the Ox source code.
>
> (4) Call Ox source code from an interactive program.
>
> The first two approaches are the most simple , and can be used if the code is 'unidirectional' (i.e. input is collected and then the program is run).

Then, Doornik shows, through a worked-out example called 'RanApp', how to implement method (4) with Microsoft Visual C++ and Microsoft Visual Basic. This example, though simple, requires a decent practice of the language used to build the user-friendly interface (Visual C++ or Visual Basic, but it could be Java or a set of C++ libraries such as Qt or wxwindows), but it also requires a C++ compiler and a good kwowledge of the Ox DLL mechanisms.

What we propose here is to explain how to use a simplified version of method (3) with a set of DLL's we have ported to Ox. That method avoids all the technicalities: the user can create an user-friendly interface directly from Ox . But of course, the price to pay for the extreme simplicity is that the user interface is less powerful than the one created with a specific DLL or with method (4) (since other people need a correctly installed version of Ox and of the `oxjapi` package to run your programs).

1.2 Choice of the interface language

Several matrix programming langages come with a package that provides a user interface with Tk. As Tk is a scripted language, interfacing it with Ox for example would require a permanent shift mechanism between the Tk interpreter and Ox. This solution would be rather easy to implement, but somehow not very pleasant to use.

So we thought that the most interesting thing would be porting to Ox a set of C functions. One key requirement we had was platform-independence of the Ox code.

2 The Japi package

The Java Application Programming Interface, or Japi for short, has been developed by Merten Joost (see [4]) . The aim of the package is to allow for building user interfaces with languages that are not object-oriented (e.g. C, Basic, Fortran 77). To build applications with Japi, the user only needs a Java

Runtime Environment (JRE) and the japi DLL associated with his system and his programming language.

We have ported to Ox the C version of Japi. The package is called `oxjapi`. We prefixed it with 'ox' to avoid possible confusions between header files.

3 Installation

On all platforms, we need to have a Java Runtime Environment (JRE). If you don't have one, you can get it from
`http://java.sun.com/downloads`
or from
`http://site.voila.fr/choirat/software/oxjapi.oxjapi.html`
(for an older but smaller Windows version).

3.1 Windows

Unzip the file 'oxjapi.zip' into the directory 'packages' of your Ox installation.

3.2 Linux

We have compiled `oxjapi` under Mandrake 8.2. It should work on any recent Red Hat or Mandrake configuration. As 'root', just unpack the archive oxjapi.tar.gz in the directory 'packages' of your Ox installation through the command:

```
tar -xvvzf oxjapi.tar.gz
```

You may have to modify the `OX3PATH` variable.

3.3 Unix

1. Do the same as the Linux installation.
2. Go into the directory '[...]/packages/oxjapi/src' and type (we assume that your Ox path is settled as described in the Ox Documentation or in Chapter 1 of [1]):

   ```
   make; gcc -L. OxJapiDllSrc.c -shared -o ../oxjapi.so -ljapi
   ```

3.4 'Hello World!'

Just to fix ideas, we present here the `oxjapi` code of the traditional 'Hello World!' example. The source code can be found in 'packages/oxjapi/samples/HelloWorld.ox'.

```
#include <oxstd.h>
#include <packages/oxjapi/oxjapi.h>
```

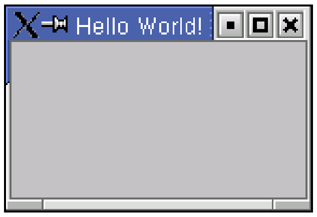

Figure 1: 'Hello World!' with `oxjapi` on Linux KDE

```
main()
{
    j_start();

    decl jFrame = j_frame("Hello World!");
    j_show(jFrame);

    while (j_nextaction() != jFrame) {}

    j_quit();
}
```

The result is shown in Figure 1.

4 The 'RanApp' example revisited

We propose here to go through the 'RanApp' example developed in the Ox Appendices (see Chapter 1 of [2]). The commented code can be found in 'packages/oxjapi/samples/RanApp.ox'. We have decided to recreate in the simplest way an application that has the same features as Doornik's example with only small visual differences. This section requires a basic kwowledge of Ox (as presented in the first chapters of [3]).

4.1 Hungarian notation

All the variables manipulated by `oxjapi` are integers. To keep in mind their specificity, we propose here to prefix them with '`j`'. For example, an `oxjapi` variable 'x' will be denoted '`jX`' and not '`iX`' as usually.

4.2 Creating the 'RanApp' window

The first step is to create the 'RanApp' window. The code can be found in 'packages/oxjapi/samples/RanAppTemp1.ox'.

```
#include <oxstd.h>
#include <packages/oxjapi/oxjapi.h>
```

```
main()
{
    j_start();

    decl jFrame  =  j_frame("RanApp");
    j_setborderlayout(jFrame);

    decl jPanel  =  j_panel(jFrame);
    j_setgridlayout(jPanel, 5, 1);
    j_setinsets(jPanel, 10, 10, 10, 10);
    j_setvgap(jPanel, 10);

    decl jButtonDimension   =  j_button(jPanel, "Dimension...");
    decl jButtonGenerate    =  j_button(jPanel, "Generate");
    decl jButtonVariance    =  j_button(jPanel, "Variance");
    decl jButtonDraw        =  j_button(jPanel, "Draw");
    decl jButtonClose       =  j_button(jPanel, "Close");

    j_setsize(jFrame, 150, 300);
    j_setpos(jFrame, 20, 10);
    j_setresizable(jFrame, J_FALSE);
    j_show(jFrame);
    j_seticon(jFrame, j_loadimage("images/RanApp.gif"));

    decl jObj;

    do
    {
        jObj = j_nextaction();

    } while ((jObj != jButtonClose) && (jObj != jFrame));

    j_quit();
    println("\ndone!");
}
```

All the functions of the `oxjapi` package are prefixed with '`j_`'.

- The first thing to do is to connect to the Japi kernel. It is done through the function `j_start()`. But, a more secure way of calling the kernel is:

```
    if (!j_start()) print("can't connect to server\n"), exit(0);
```

- The fundamental element of the `oxjapi` package is the *frame*, which is created through the function `j_frame("RanApp")` where `RanApp` is of course the name of the frame. If you write `j_frame("")`, then the frame will automatically be entitled `JAPI (Java Application Programming Interface)`.

- We need to use some geometric properties of the frame to place the buttons afterwards, so we use the function: `j_setborderlayout().`
- But a frame cannot be easily manipulated directly. So we need to work with a more versatile element: the *panel*. The panel is, to make a parallel with object-oriented programming, the descendent of the frame and is created with: `j_panel()`.
- We want to place the buttons on a 5×1 grid, which is achieved through `j_setgridlayout()`.
- `j_setinsets()` makes a space (resp. top, bottom, left, right) of the specified value between the panel and the frame.
- `j_setvgap()` creates a vertical space between the forthcoming buttons.
- The 5 buttons are created with the `j_button(panel, "Label")` function.
- The `j_setsize()` and `j_setpos()` functions are used to set the initial size and initial position of the frame to the specified values.
- Frames are by default resizable. The `j_setresizable(frame, J_FALSE)` function changes this attribute. (Note that `J_FALSE` and the usual Ox `FALSE` are the same, that is 0).
- Finally the function `j_show()` displays the frame. The function `j_seticon()` is used to change the icon of the application (a simple example can be found in 'packages/oxjapi/samples/HelloWorldWithIcon.ox').

So, running Ox should give a window similar to Figure 2.

At that point, all the graphical aspects of the RanApp window have been created. But, we need to handle the actions of the user, that is for the moment only closing the window.

- The variable `obj` will receive the user's action known through the function `j_nextaction()`. Remark that *this function does not require all the system's resources*. This fact can easily be seen with a CPU monitoring tool. In fact, the application is still connected to the JAPI kernel but is, to put it simply, *frozen*.[1]
- the `do while` loop insures that the window will be displayed until the user either clicks on the button labelled 'Close' or closes the window directly.
- the `j_quit()` function removes the connection to the Japi kernel.

[1]There is another JAPI function called `j_sleep()` which, despite its name, stops the application for a given period of time.

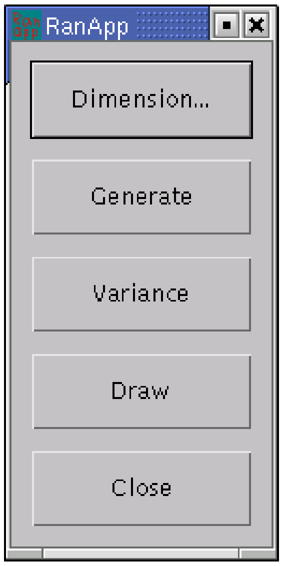

Figure 2: ‘RanApp’ main window on Linux KDE

4.3 Linking buttons and user actions

Now, we pass to the handling of the ‘Dimension’, ‘Generate’, ‘Variance’ and ‘Draw’ buttons. To make the code simpler to understand, we will not use static global variables as Doornik did in his example. We add the `OnDimension`, `OnGenerate`, `OnVariance` and `OnDraw` functions. Remark that the first parameter in `OnGenerate` is assumed to be the address of a variable. The code can be found in ‘packages/oxjapi/samples/RanAppTemp2.ox’.

```
OnDimension(const iT, const iN, const iAcf)
{
    println("T = ", iT, ", n = ", iN,", lag length = ", iAcf);
}

OnGenerate(const amX, const iT, const iN)
{
    amX[0] = rann(iT, iN);
}

OnVariance(const mX)
{
    println( variance(mX) );
}

OnDraw(const mX, const iAcf)
{
    DrawCorrelogram(0, mX[][0]', "ran1", iAcf);
    DrawSpectrum(1, mX[][0]', "ran1", iAcf);
    ShowDrawWindow();
}
```

And the `do while` loop in the main function now becomes (remark that `j_dialogDimension` is the function that opens the dialog window, it will be written afterwards):

```
    decl mX = <0>, iT = 100, iN = 2, iAcf = 20;

    do
    {
        jObj = j_nextaction();

        if (jObj == jButtonDimension)
        {
            OnDimension(iT, iN, iAcf);
            //j_dialogDimension(jFrame, &iT, &iN, &iAcf);
        }

        else if (jObj == jButtonGenerate)
            OnGenerate(&mX, iT, iN);

        else if (jObj == jButtonVariance)
            OnVariance(mX);

        else if (jObj == jButtonDraw)
            OnDraw(mX, iAcf);

    } while ((jObj != jButtonClose) && (jObj != jFrame));
```

4.4 Creating the 'Dimensions' Dialog

The last part consists in writing the code of the dialog that appears after clicking on the 'Dimension' Button. The function that performs that task is called `j_dialogDimension`. The geometry of the 'Dimensions' Dialog is a little more complex: the panel we are working with is contained in the dialog window, which is a 'descendent' of the frame we created in the main function. Figure 3 shows the structure of the dialog.

The code can be found in 'packages/oxjapi/samples/RanAppTemp3.ox'.

```
j_dialogDimension(const jFrame, const aiT, const aiN, const aiAcf)
{
    decl jObj = 0, sVal = "", bRetval = J_FALSE;

   /* The first thing to do is to disable the 'Ranapp' window.*/

    j_disable(jFrame);
```

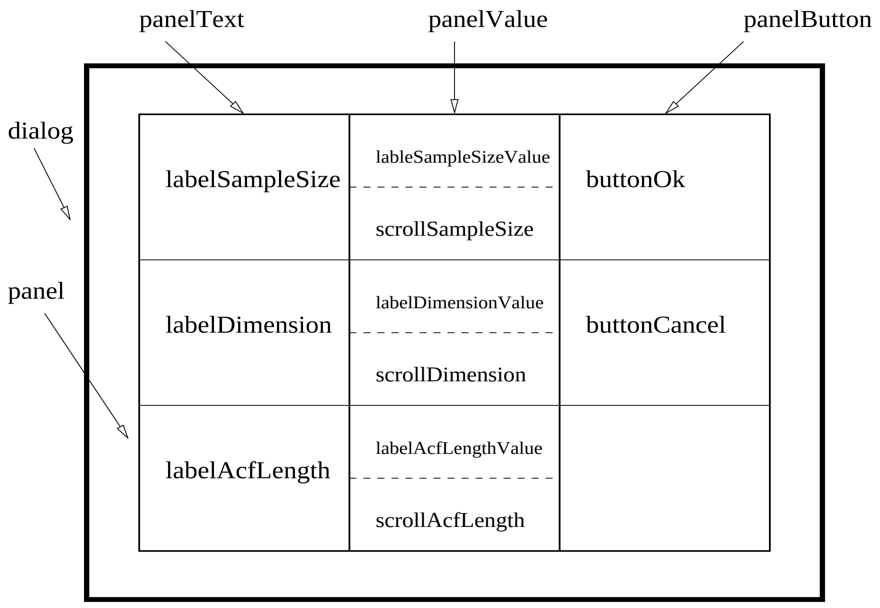

Figure 3: Structure of the dialog window

/* Then, we create a dialog window and we add all the required components.*/

```
decl jDialog    =   j_dialog(jFrame, "Dimensions");
j_setborderlayout(jDialog);
```

/* We create the main panel. Its geometry is a 3×1 grid.*/

```
decl jPanel     =   j_panel(jDialog);
j_setgridlayout(jPanel, 1, 3);
```

/* We create the panel ‘panelText’ and add the three labels.*/

```
decl jPanelText =   j_panel(jPanel);
j_setgridlayout(jPanelText, 3, 1);
j_setinsets(jPanelText, 20, 20, 20, 20);
j_setvgap(jPanelText, 20);
j_sethgap(jPanelText, 20);
decl jLabelSampleSize   =   j_label(jPanelText, "Sample size, T  ");
decl jLabelDimension    =   j_label(jPanelText, "Dimension, N  ");
decl jLabelAcfLength    =   j_label(jPanelText, "ACF length, s  ");
```

/* We create the panel ‘panelValue’. It has three subpanels, each containing a value and a scrollbar.*/

```
decl jPanelValue    =   j_panel(jPanel);
j_setgridlayout(jPanelValue, 3, 1);
```

```
    j_setinsets(jPanelValue, 20, 20, 20, 20);
    j_setvgap(jPanelValue, 20);
    j_sethgap(jPanelValue, 20);

    decl jPanelSampleSize       =   j_panel(jPanelValue);
    j_setnamedcolorbg(jPanelSampleSize, J_WHITE);
    j_setgridlayout(jPanelSampleSize, 2, 1);
    decl jLabelSampleSizeValue  =   j_label(jPanelSampleSize, "    ");
    decl jScrollSampleSize      =   j_hscrollbar(jPanelSampleSize);

    decl jPanelDimension        =   j_panel(jPanelValue);
    j_setnamedcolorbg(jPanelDimension, J_WHITE);
    j_setgridlayout(jPanelDimension, 2, 1);
    decl jLabelDimensionValue   =   j_label(jPanelDimension, "    ");
    decl jScrollDimension       =   j_hscrollbar(jPanelDimension);

    decl jPanelAcfLength        =   j_panel(jPanelValue);
    j_setnamedcolorbg(jPanelAcfLength, J_WHITE);
    j_setgridlayout(jPanelAcfLength, 2, 1);
    decl jLabelAcfLengthValue   =   j_label(jPanelAcfLength, "    ");
    decl jScrollAcfLength       =   j_hscrollbar(jPanelAcfLength);

   /* We have to set the maximal, minimal and initial values of the scrollbars.
The '10' we add corresponds to the thickness of the scrollbars*/

    j_setmax(jScrollSampleSize, 1000 + 10);
    j_setmax(jScrollDimension,  4 + 10);
    j_setmax(jScrollAcfLength,  1000 + 10);

    j_setmin(jScrollSampleSize, 10);
    j_setmin(jScrollDimension,  1);
    j_setmin(jScrollAcfLength,  10);

    j_setvalue(jScrollSampleSize, aiT[0]);
    j_setvalue(jScrollDimension,  aiN[0]);
    j_setvalue(jScrollAcfLength,  aiAcf[0]);

   /* We add the panel 'panelButton' and the buttons 'OK' and 'Cancel'.*/


    decl jPanelButton   =   j_panel(jPanel);
    j_setgridlayout(jPanelButton, 3, 1);
    j_setinsets(jPanelButton, 20, 20, 20, 20);
    j_setvgap(jPanelButton, 20);
    j_sethgap(jPanelButton, 20);

    decl jButtonOk      =   j_button(jPanelButton, "OK");
    decl jButtonCancel  =   j_button(jPanelButton, "Cancel");
    j_setinsets(jPanelButton, 20, 20, 20, 20);
    j_setvgap(jPanelButton, 20);
    j_sethgap(jPanelButton, 20);
   /* We set the size and position of the dialog window.*/

    j_setsize(jDialog, 450, 200);
    j_setpos(jDialog, 100, 100);
    j_show(jDialog);

   /* We link widgets and user actions.  First, we update the displayed scrollbar
values.  Then, if button 'OK' is clicked, the new sample size, dimension and
ACF length are returned.  If button 'Cancel' is clicked or if the dialog window
is closed the values are returned unchanged.*/

    while ((jObj != jButtonCancel) && (jObj != jDialog))
    {
        sVal = sprint(j_getvalue(jScrollSampleSize));
        j_settext(jLabelSampleSizeValue, sVal);
        sVal = sprint(j_getvalue(jScrollDimension));
        j_settext(jLabelDimensionValue, sVal);
        sVal = sprint(j_getvalue(jScrollAcfLength));
        j_settext(jLabelAcfLengthValue, sVal);

        jObj = j_nextaction();

        if (jObj == jButtonOk)
        {
            bRetval     =   J_TRUE;
            aiT[0]      =   j_getvalue(jScrollSampleSize);
            aiN[0]      =   j_getvalue(jScrollDimension);
            aiAcf[0]    =   j_getvalue(jScrollAcfLength);
            break;
        }
    }

   /* We set the focus back to the main window.*/

    j_dispose(jDialog);
    j_enable(jFrame);
    return bRetval;
}
```

The last thing to do before running the application is removing the comments in the main function before calling `j_dialogDimension`. The dialog window obtained is shown in Figure 4.

5 Function reference

The `oxjapi` function reference has still to be written. But, Merten Joost's Japi reference manual (for the C programming language) provides almost all the information you need (see [5]). Besides, we have also ported to Ox most of the Japi examples written by M. Joost (in the directory 'packages/oxjapi/samples/japi/):

- `alert.ox` displays several types of alert messages.

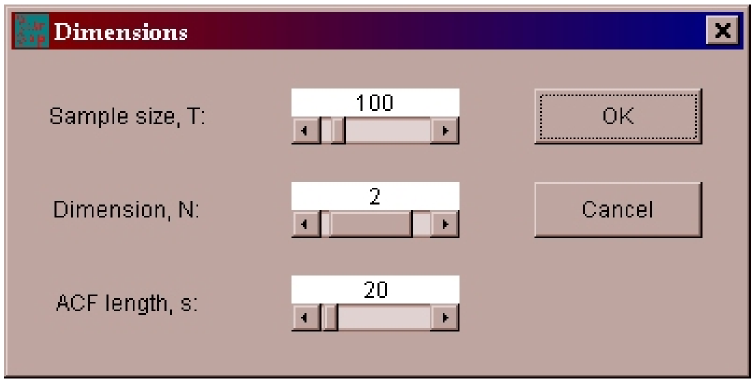

Figure 4: ‘RanApp’ dialog window on Windows

- `borderlayout.ox` illustrates the properties of the `j_setborderlayout` function.
- `borderpanel.ox` illustrates the properties of the `j_setborderpanel` function.
- `button.ox` displays an increasing and shrinking button.
- `canvas.ox` creates a simple canvas and progressively fills it with colored points.
- `checkbox.ox` shows how to deal with checkbox events.
- `choice.ox` lets the user pick a predefined color from a choice widget.
- `colorpicker.ox` is an application whose background color is interactively chosen.
- `colors.ox` displays a canvas of a shade of colors.
- `colors1.ox` also displays a canvas of a shade of colors, but considered as an image.
- `componentlistener.ox` displays a window that reports user actions.
- `cursor.ox` shows all the types of cursors defined.
- `daemon.ox` simply starts the Japi kernel.
- `dialog.ox` displays a dialog open from an application that has a menubar.
- `dialogmodal.ox` also displays a dialog open from an application that has a menubar.

- `drawables.ox` shows several drawings on a canvas, that can be printed or saved as a bitmap.
- `filedialog.ox` displays a filedialog.
- `flowlayout.ox` is an example of the flow layout.
- `flowsimple.ox` is an elementary example of the flow layout.
- `focuslistener.ox` shows how buttons react to user actions.
- `font.ox` is an application that shows how to manipulate fonts, font styles and font sizes.
- `frame.ox` explains how to replace the Java cup by an image of your choice.
- `graphic.ox` shows the different graphical primitives defined in Japi.
- `graphicbutton.ox` shows how to use images to make graphical buttons.
- `graphiclabel.ox` illustrates the use of graphical labels.
- `gridlayout.ox` illustrates the properties of the grid layout.
- `image.ox` shows how to invert the colors of an image.
- `insets.ox` shows how `j_setinsets` works.
- `keylistener.ox` shows how to handle keyboard events.
- `label.ox` displays a bouncing label.
- `lines.ox` shows the predefined types of lines.
- `list.ox` lets the user pick a predefined color from a list widget.
- `listmultiple.ox` lets the user pick several predefined colors from a list widget.
- `mandel.ox` computes and displays Mandelbrot's fractal.
- `mandel1.ox` computes and displays a resizable Mandelbrot's fracal.
- `mandel2.ox` computes and displays a resizable Mandelbrot's fracal as an image that can be zoomed, as shown in Figure 5.
- `menu.ox` is a menubar application.
- `mousebuttons.ox` shows how to deal with mouse clicks.
- `mouselistener.ox` shows how to handle mouse events.
- `panel.ox` shows a label bouncing in a panel.
- `popupmenu.ox` displays a simple popupmenu.

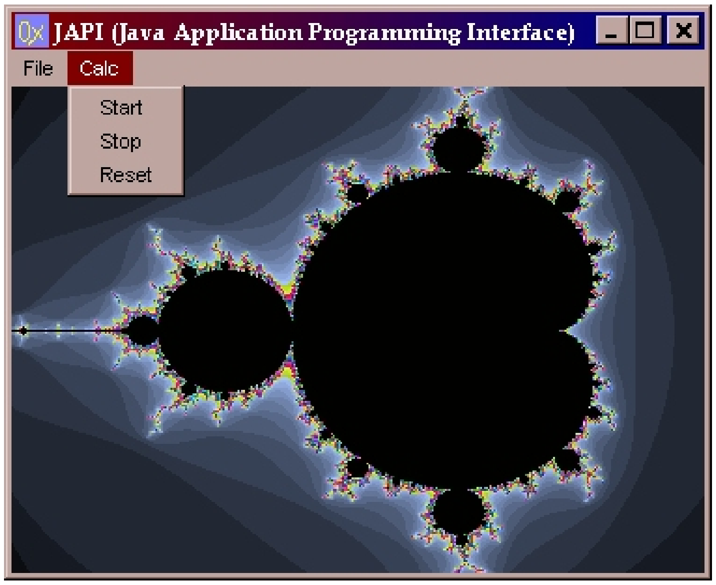

Figure 5: Mandelbrot's fractal on Windows

- `print.ox` shows how to print a file.
- `radiobutton.ox` shows some properties of radiobuttons.
- `scrollbar.ox` illustrates the use of a scrollbar.
- `scrollpane.ox` shows a scrollable panel.
- `simple.ox` is Japi's 'Hello World!'.
- `simplemenu.ox` is an elementary menubar application.
- `text.ox` is a small text editor.
- `textfield.ox` illustrates the use of the textfield widget through a 'login password' window.
- `video.ox` plays a short video movie.
- `viewer.ox` is an image viewer.
- `windowlistener.ox` displays a frame in which user actions get printed.

References

[1] Doornik, J. A. (2001). *Ox 3.0 - an object-oriented matrix programming language*, Timberlake Consultants Press.

[2] Doornik, J. A. (2002). *Ox Appendices, Ox version 3.1*, downloadable from: `http://www.nuff.ox.ac.uk/Users/Doornik`.

[3] Doornik, J. A., Draisma, G. and Ooms, M. (2001). *Introduction to Ox Version 3*, Timberlake Consultants Press, downloadable from: `http://www.nuff.ox.ac.uk/Users/Doornik`.

[4] Joost, M. *JAPI: Java Application Programming Interface*, downloadable from: `http://www.japi.de`.

[5] Joost, M. *JAPI Reference Manual for the C language*, downloadable from: `http://www.japi.de`.