Filippo Battaglia

Nanodesktop

User guide

Version 0.3.4 (PSP/PSPE)

What is Nanodesktop

Nanodesktop is an sdk with many libraries for the development of applications on embedded platforms.

Actually, Nanodesktop supports the PSPE Emulator and Sony PSP (Playstation Portable). At a later date we'll add support for other platforms, such as the Fox Board or Linux Embedded Platform (LEP).

Nanodesktop features

Nanodesktop is a simple windows manager. Under the PSP/PSPE platform it makes it easier to develop both simple and advanced applications. These are the features of the included libraries:

- Support for base screen drawing routines;
- Support for drawing routines in windows;
- Support for minimize, maximize, move, resize windows;
- Support for winmenu;
- Support for buttons, toolbars, progress-bars, trackbars;
- Support for fonts and text writing routines;
- Support for textboxes, listboxes, textareas, controlboxes;
- Support for mouse emulation;
- Support for file manager routines;
- Support for Virtual Keyboard;
- Support for loading/showing/saving images;
- Support for transparency and wallpapers;
- Support for True Type fonts
- Support for disk cache manager;
- Support for stdout/stderr terminal;
- Supports 12 different graphical formats;
- Support for USB, IRDA, Raw-IR;
- Support for webcam (audio and video recording);
- Support for sound;
- Support for MP3 and WAV decoding;
- Support for image recognition (using OpenCV by Intel);
- Support for network connections;
- Support for files transfers;
- Support for voice synthesis (using Flite Voice Synthesizer).

Nanodesktop also provides **NanoC and NanoM**; two libraries that provide standard ANSI C functions. The Nanodesktop libraries can work with the PSP console and under the PSPE emulator.

Nanodesktop supports HAL (Hardware Abstraction Layer) - this component allows for developers to easily port libraries from other platforms. A complete port only requires a few days of work.

The author

The author of Nanodesktop is Filippo Battaglia, a student of Electronic Engineering at the University of Messina.

Nanodesktop has been developed as a research project in the Visilab Research Center at the University of Messina. I want to thank my teacher, prof. Giancarlo Iannizzotto, for support and guidance.

I want also to thank *Nicholas Sinkinson* and *Sam Gluck*, that have revised this guide before the release of Nanodesktop.

Chapter 1

The contents of the package

This manual is downloadable from the Nanodesktop website or with the Nanodesktop distribution.

These are the folders that are included in the Nanodesktop distribution: all folders are found under the PSP main folder because you have downloaded the PSP version of Nanodesktop.

PSP\Emulators

This folder is dedicated to the PSPE emulator. It is normally empty. We cannot redistribute the PSPE emulator for legal reasons. The user must download and copy the emulator to this folder.

PSP\ndDevIL

This folder contains our porting of the DevIL library for loading/saving images in formats different from bitmap and the libraries that provide support for the specified graphical formats. These components are released under different licenses: view the files called NOTES.VISILAB.DOC for information related to the licenses of the different packages.

PSP\ndDevIL\DevILCore

This folder contains the core of the DevIL library. The modified version is called ndDevIL. It has been released under the LGPL license.

PSP\ndDevIL\JpegLib

A library for loading/saving JPEG files.

PSP\ndDevIL\lpng1210

A library for loading/saving PNG files

PSP\ndDevIL\tiff

A library for loading/saving TIFF files.

PSP\ndDevIL\zlib

Zlib library.

PSP\ndHighGUI

This folder contains the OpenCV ndHighGUI component. This is not a simple porting of the original Intel HighGUI, but is a component that has been totally rewritten by Filippo Battaglia in a way that allows the integration of OpenCV with the Nanodesktop graphical system. It maintains compatibility with the original Intel HighGUI prototypes, and also adds new functions.

All functionalities are implemented, except for the mouse callback system.

PSP\ndOpenCV

This folder contains different components related to the OpenCV system. These are the subfolders:

PSP\ndOpenCV\cv, PSP\ndOpenCV\cxcore, PSP\ndOpenCV\cvaux

They are ported versions of the original Intel libraries.

PSP\ndOpenCV\lib

Contains the binary versions of cv, cvaux and cxcore libraries for PSP.

PSP\ndOpenCV\samples

Contains OpenCV example programs (source code). In subfolder *Apps,* you will find the source codes of OpenCV Application 1 and OpenCV Application 2.

In the subfolder *Intel based,* there are examples that are equivalent to those included in the original OpenCV library for the x86 platform, and that have been developed by Intel engineers.

PSP\SDK

This folder contains a modified version of the free SDK for the Playstation Portable, with psp-gcc and psp-g++ compilers, and relative headers. It also contains the core of the Nanodesktop graphical system (binaries and codes).

PSP\SDK\Docs FW

Contains the stubs for PSP kernel systemcalls (FAT version – firmware version 1.50).

PSP\SDK\Docs PSPSDK

A *doxygen* document that illustrates the prototypes of the PSPSDK system libraries.

PSP\SDK\NanoCore

A software component that is at a lower level in the Nanodesktop System Architecture. It provides to the user different functions: it implements a new libc that is compatible with standard C, and a new mathematical library that is compatible with the PSP and PSPE.

PSP\SDK\NanoCore\NanoC

It contains NanoC library (source and library).

PSP\SDK\NanoCore\NanoM

It contains NanoM library (source and library).

PSP\SDK\NanoCore\NanoCPP

It contains the standard C++ library (Apache C++ standard library)

PSP\SDK\Nanodesktop

This is the core of the distribution. In */src* you'll find the sources of the library, in */lib* there are different versions of the library in binary format.

In *demo*, you'll find a simple demo that displays a few of Nanodesktop's capabilities.

PSP\SDK\PspDev

Contains a modified version of the PSPSDK. In */bin* you can find compilers like psp-gcc or psp-g++ and other support utilities. The binary versions of the libraries are in the folders */lib, /psp/lib, /psp/sdk/lib*.

PSP\ServicesFiles

Contains special scripts for the integration of Nanodesktop with the Dev-CPP IDE. These scripts are Makefile_PSPE.mak and Makefile_PSP.mak. It also contains the png files that are included automatically in the EBOOT.PBP during the compilation stage.

PSP\Tools
Contains tools such as M1pspc for image conversion and the NTF creator for Nanodesktop Text Format fonts.

PSP\ndLibCurl

This folder contains two components. The former is the *curl library,* a ported version of a standard library that is able to connect to HTTP/FTP servers and to download files. The latter is the *gcurl library,* a component that provides a graphical layer over the curl functionalities and that integrates itself into the nd graphical system.

PSP\ndFLite

This folder contains a ported version of the Flite voice synthesizer. Flite is a technology created by Carnegie Mellon University that is able to synthesize speech. NdFLite is a version of Flite, specifically designed to integrate itself into the nd environment (it uses Envelope Audio System, and EMI to accelerate computations).

PSP\ndSQLLite

This folder contains a version of SQLLite library for Nanodesktop environment. SQLLite is a library that is able to manage a database in a simple file.

PSP\ndSIFT

This folder contains a version of OpenCV SIFT library written by Robert Hess. This tecnology allows to the user the execution of *Scalar Invariant Features Transform* algorithm using a simple PSP. The library is available either in PSPE Version or in PSP version.

PSP\ndGOCR

A version of GNU OCR that is designed to work under Nanodesktop environment.

PSP\ndExOSIP2, PSP\ndORTP, PSP\ndOSIP2

These libraries support VOIP phone calls. They will be used in the next *ndFurikup* application.

PSP\ndFreeType

This library adds support for TrueType fonts

PSP\ndISpell

An orthographical corrector by GNU.

PSP\ndOCRAD

A version of OCR, written in C++ and ported under Nanodesktop environment.

PSP\ndZip

A set of programs (MiniZip and Unzip) that provide support for zip compression.

Chapter 2

Installing Nanodesktop

Now we'll begin the installation of Nanodesktop. Nanodesktop has been tested only under Windows OS, but it can be adapted to work under different operating systems, like Linux or *BSD. Remember that we cannot support installations that are not under Windows.

The installation of Nanodesktop will be taken in steps:

Stage 1: Download Nanodesktop Package

Download the Nanodesktop package from the Nanodesktop website. Create a folder (for example C:\NDENV) and unzip the contents of the Nanodesktop Package into it.

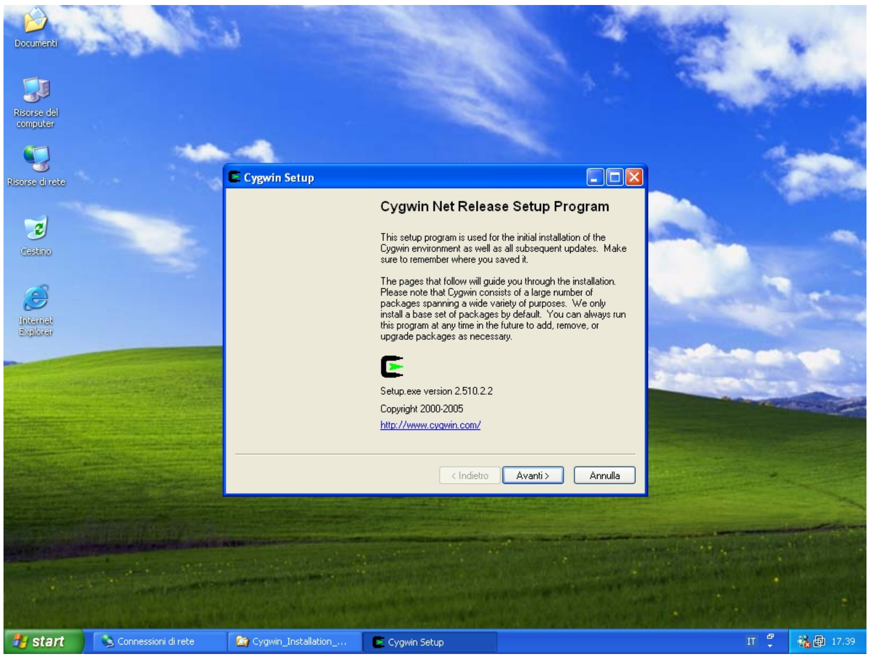
Stage 2: Prepare Cygwin

Nanodesktop needs Cygwin™ to work. To install Nanodesktop, you need to prepare Cygwin onn your operating system.

If you haven’t installed cygwin yet, you will have **to install a new cygwin complete installation from the cygwin website.**

Cygwin complete installation

Download *setup.exe* from cygwin.com.



Cygwin Net Release Setup Program

This setup program is used for the initial installation of the Cygwin environment as well as all subsequent updates. Make sure to remember where you saved it.

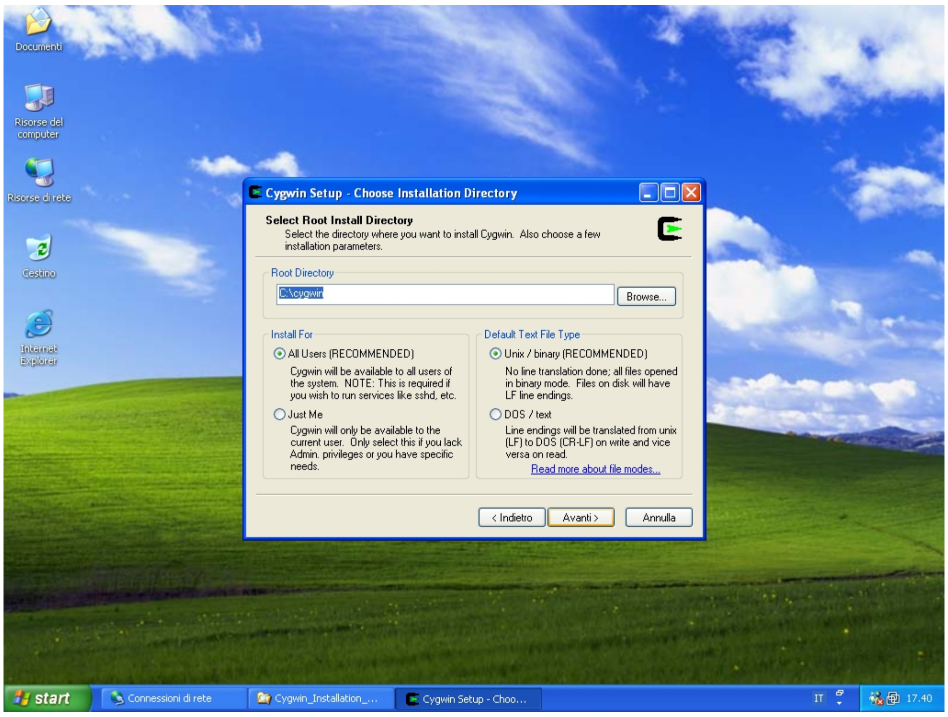
The pages that follow will guide you through the installation. Please note that Cygwin consists of a large number of packages spanning a wide variety of purposes. We only install a base set of packages by default. You can always run this program at any time in the future to add, remove, or upgrade packages as necessary.

Setup.exe version 2.510.2.2

Copyright 2000-2005

http://www.cygwin.com/

Now continue to follow these steps:


Cygwin Setup - Choose Installation Directory

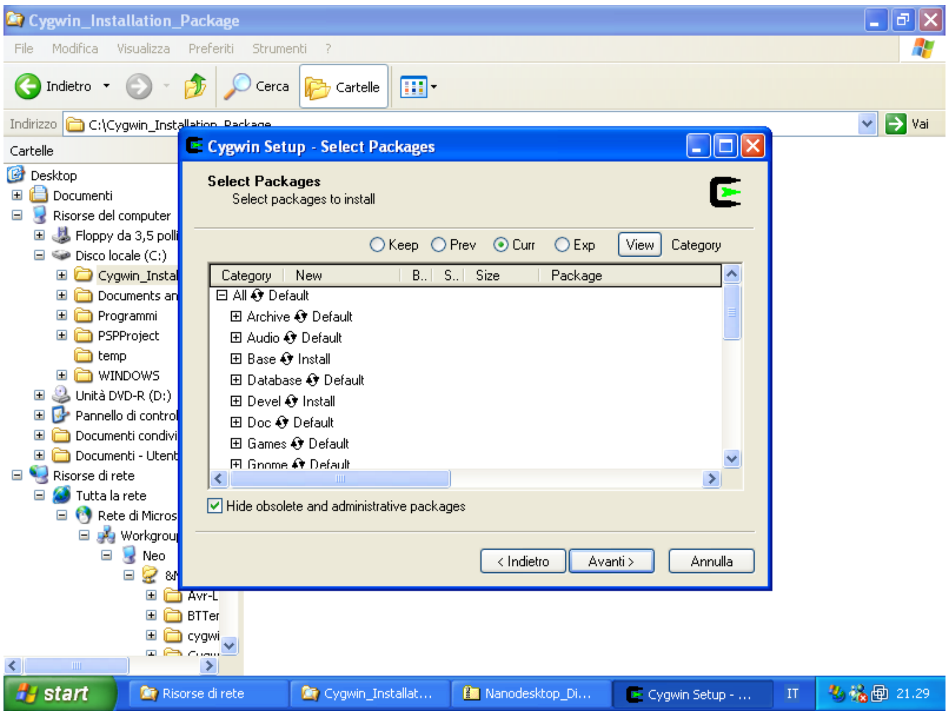
Select Root Install Directory
Select the directory where you want to install Cygwin. Also choose a few installation parameters.

Root Directory: C:\cygwin

Install For:
[✓ All Users (RECOMMENDED)] Cygwin will be available to all users of the system. NOTE: This is required if you wish to run services like sshd, etc.
Just Me — Cygwin will only be available to the current user. Only select this if you lack Admin. privileges or you have specific needs.

Default Text File Type:
[✓ Unix / binary (RECOMMENDED)] No line translation done; all files opened in binary mode. Files on disk will have LF line endings.
DOS / text — Line endings will be translated from unix (LF) to DOS (CR-LF) on write and vice versa on read.
Read more about file modes...

Follow the installation instructions until you reach the *package installation page:*


Cygwin Setup - Select Packages

Select Packages
Select packages to install

Keep Prev [✓ Curr] Exp — View: Category

Category	New	B..	S..	Size	Package
⊟ All ⟳ Default					
⊞ Archive ⟳ Default					
⊞ Audio ⟳ Default					
⊞ Base ⟳ Install					
⊞ Database ⟳ Default					
⊞ Devel ⟳ Install					
⊞ Doc ⟳ Default					
⊞ Games ⟳ Default					
⊞ Gnome ⟳ Default					

[✓] Hide obsolete and administrative packages

Change the following packages: **Devel, Base** and **Libs,** from default to **Install.**

Now install Cygwin and reboot the your system. You can now proceed to stage 3 of the installation.

Stage 3: Set NDENV_PATH,NDCYGWIN_PATH Variables Use MOVEFILES.BAT

Nanodesktop requires some files to be present in the folder:

<ndenv path>\PSP\SDK\PSPDEV\bin

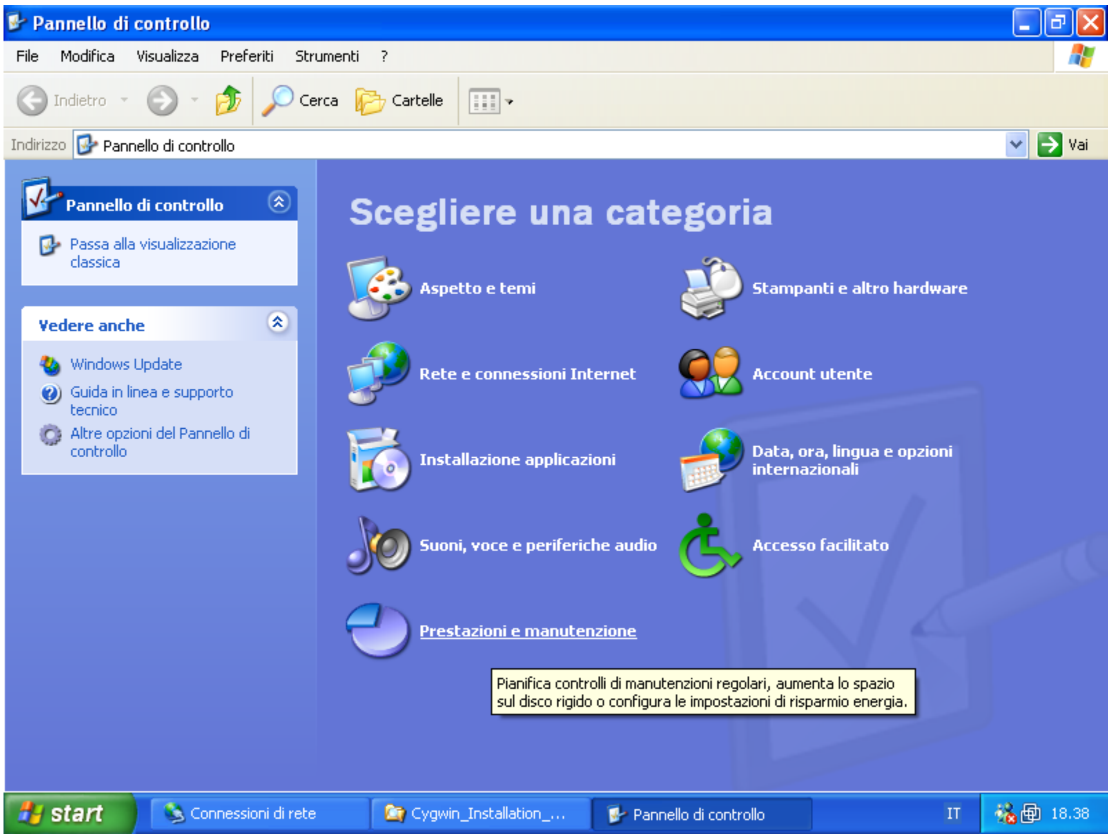
If these files are not present in the bin folder of the Nanodesktop environment, Nanodesktop will not work correctly.

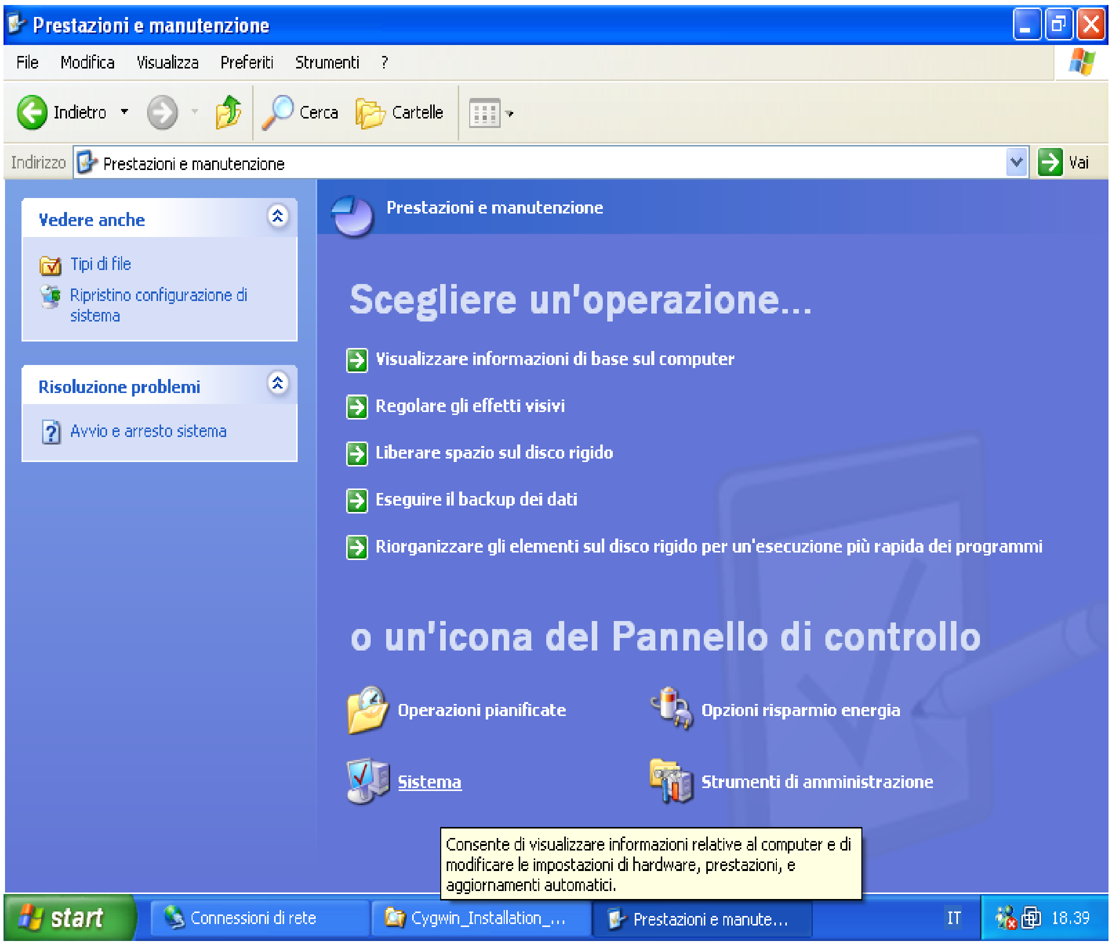
These files are distributed with Cygwin packages. We cannot redistribute these files in our package for licensing reasons (Cygwin packages are redistributable only with the sources, because they are released under the GPL license).

So, you must copy the needed files from the cygwin folder to the ndenv folder.

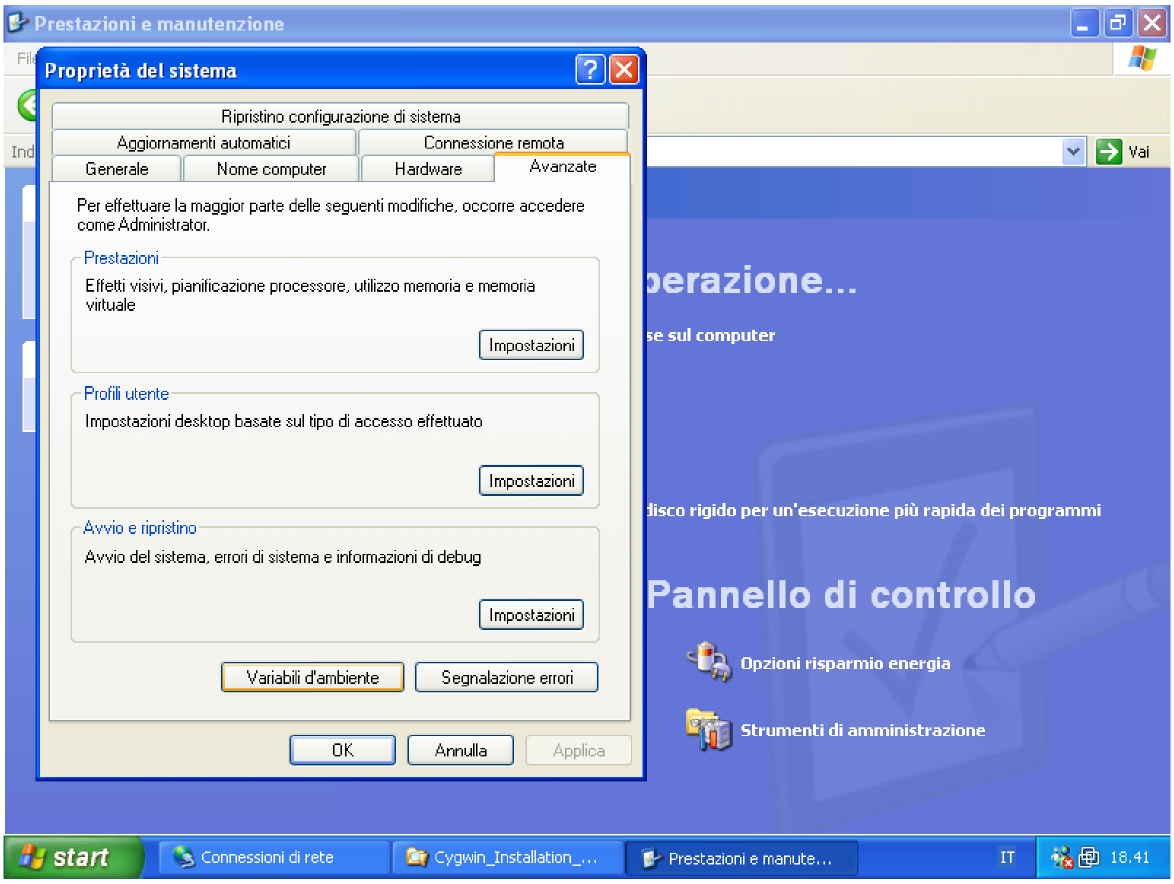
Proceed to follow these steps, firstly, set the **NDENV_PATH** variable:

Go to **Control Panel/System Settings:**


Pannello di controllo
File Modifica Visualizza Preferiti Strumenti ?
Indietro Cerca Cartelle
Indirizzo Pannello di controllo Vai
Pannello di controllo
Passa alla visualizzazione classica
Vedere anche
Windows Update
Guida in linea e supporto tecnico
Altre opzioni del Pannello di controllo
Scegliere una categoria
Aspetto e temi
Stampanti e altro hardware
Rete e connessioni Internet
Account utente
Installazione applicazioni
Data, ora, lingua e opzioni internazionali
Suoni, voce e periferiche audio
Accesso facilitato
Prestazioni e manutenzione
Pianifica controlli di manutenzioni regolari, aumenta lo spazio sul disco rigido o configura le impostazioni di risparmio energia.
start Connessioni di rete Cygwin_Installation_... Pannello di controllo IT 18.38


Prestazioni e manutenzione
File Modifica Visualizza Preferiti Strumenti ?
Indietro Cerca Cartelle
Indirizzo Prestazioni e manutenzione Vai
Vedere anche
Tipi di file
Ripristino configurazione di sistema
Risoluzione problemi
Avvio e arresto sistema
Prestazioni e manutenzione
Scegliere un'operazione...
Visualizzare informazioni di base sul computer
Regolare gli effetti visivi
Liberare spazio sul disco rigido
Eseguire il backup dei dati
Riorganizzare gli elementi sul disco rigido per un'esecuzione più rapida dei programmi
o un'icona del Pannello di controllo
Operazioni pianificate
Opzioni risparmio energia
Sistema
Strumenti di amministrazione
Consente di visualizzare informazioni relative al computer e di modificare le impostazioni di hardware, prestazioni, e aggiornamenti automatici.
start Connessioni di rete Cygwin_Installation_... Prestazioni e manute... IT 18.39

Go to **Advanced/Enviroment variables:**

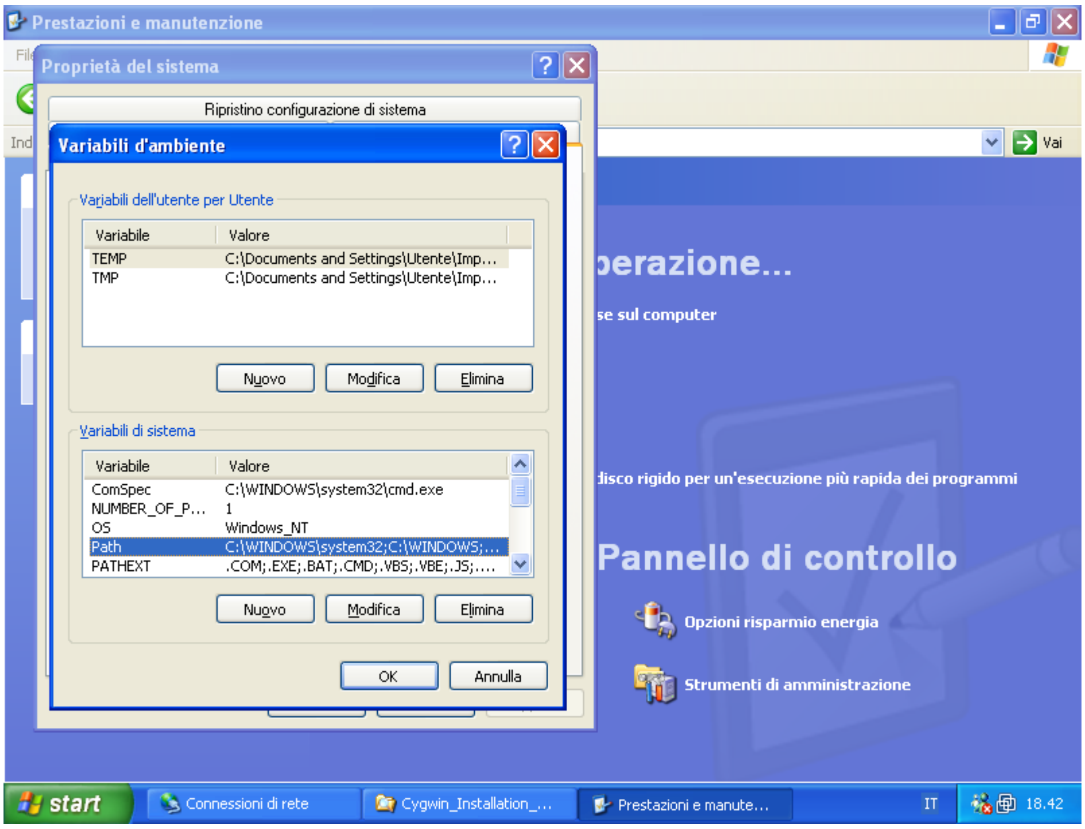


Per effettuare la maggior parte delle seguenti modifiche, occorre accedere come Administrator.

Effetti visivi, pianificazione processore, utilizzo memoria e memoria virtuale

Impostazioni desktop basate sul tipo di accesso effettuato

Avvio del sistema, errori di sistema e informazioni di debug



Variabili d'ambiente

Variabili dell'utente per Utente

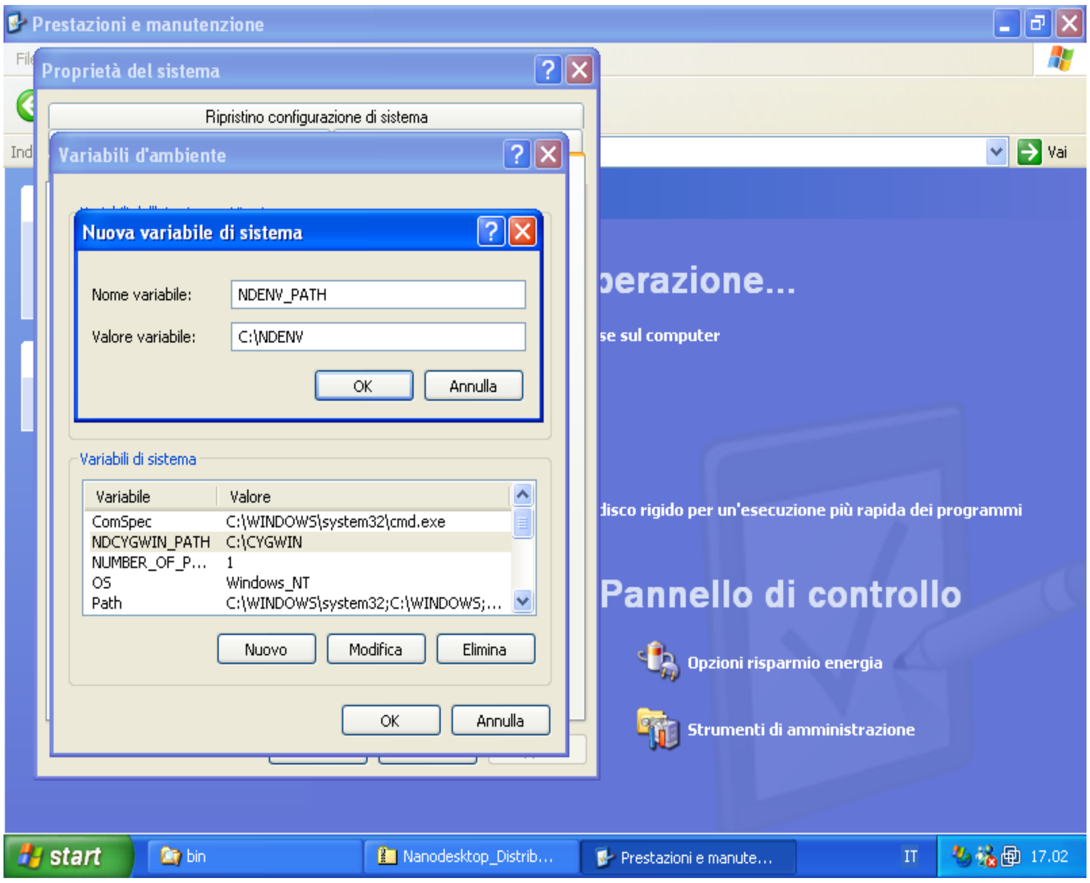
Variabile	Valore
TEMP	C:\Documents and Settings\Utente\Imp...
TMP	C:\Documents and Settings\Utente\Imp...

Variabili di sistema

Variabile	Valore
ComSpec	C:\WINDOWS\system32\cmd.exe
NUMBER_OF_P...	1
OS	Windows_NT
Path	C:\WINDOWS\system32;C:\WINDOWS;...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....

Now, you must add two new system environment variables.

The former is called **NDENV_PATH.** In this variable you must register the path in which you have unzipped the Nanodesktop code (C:\NDENV in my example):



Nuova variabile di sistema

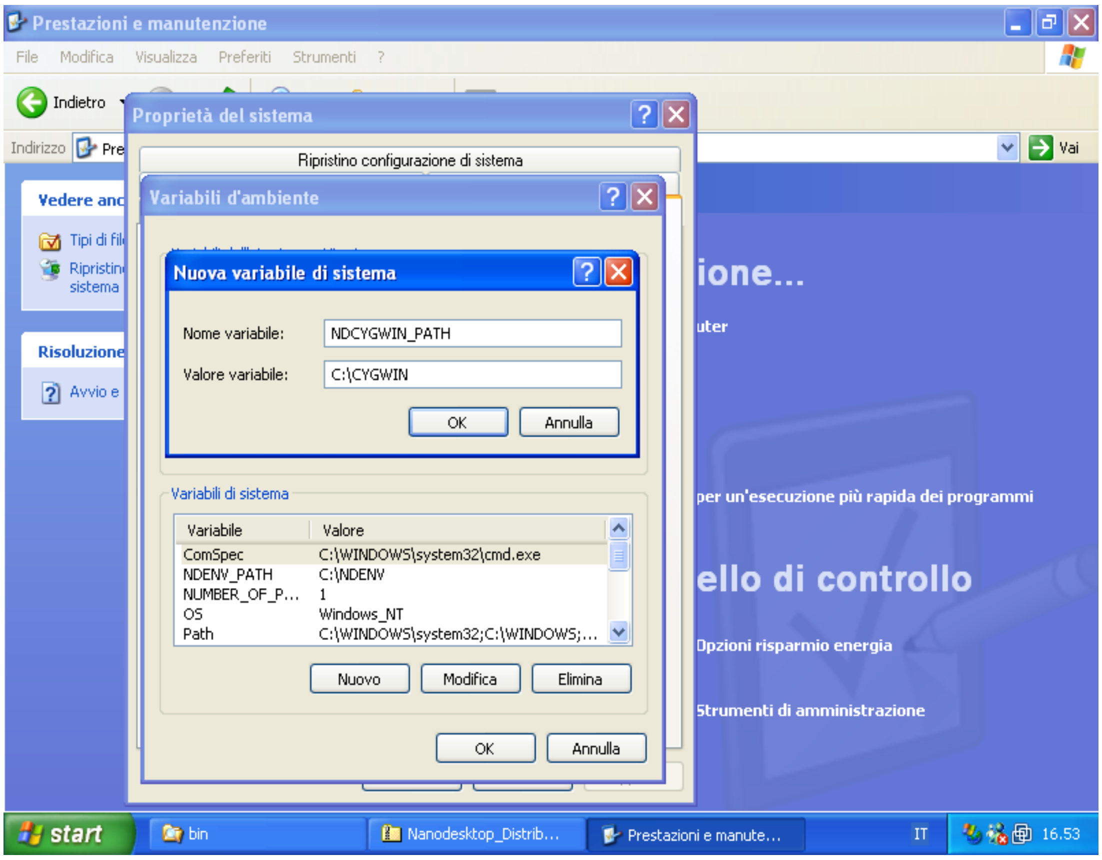
Nome variabile: NDENV_PATH

Valore variabile: C:\NDENV

Variabili di sistema

Variabile	Valore
ComSpec	C:\WINDOWS\system32\cmd.exe
NDCYGWIN_PATH	C:\CYGWIN
NUMBER_OF_P...	1
OS	Windows_NT
Path	C:\WINDOWS\system32;C:\WINDOWS;...

The second variable is **NDCYGWIN_PATH.** In this path you must register the path to which you have installed the Cygwin files (in my example C:\CYGWIN).


Prestazioni e manutenzione
Proprietà del sistema
Ripristino configurazione di sistema
Variabili d'ambiente
Nuova variabile di sistema
OK
Annulla
Nuovo
Modifica
Elimina
start

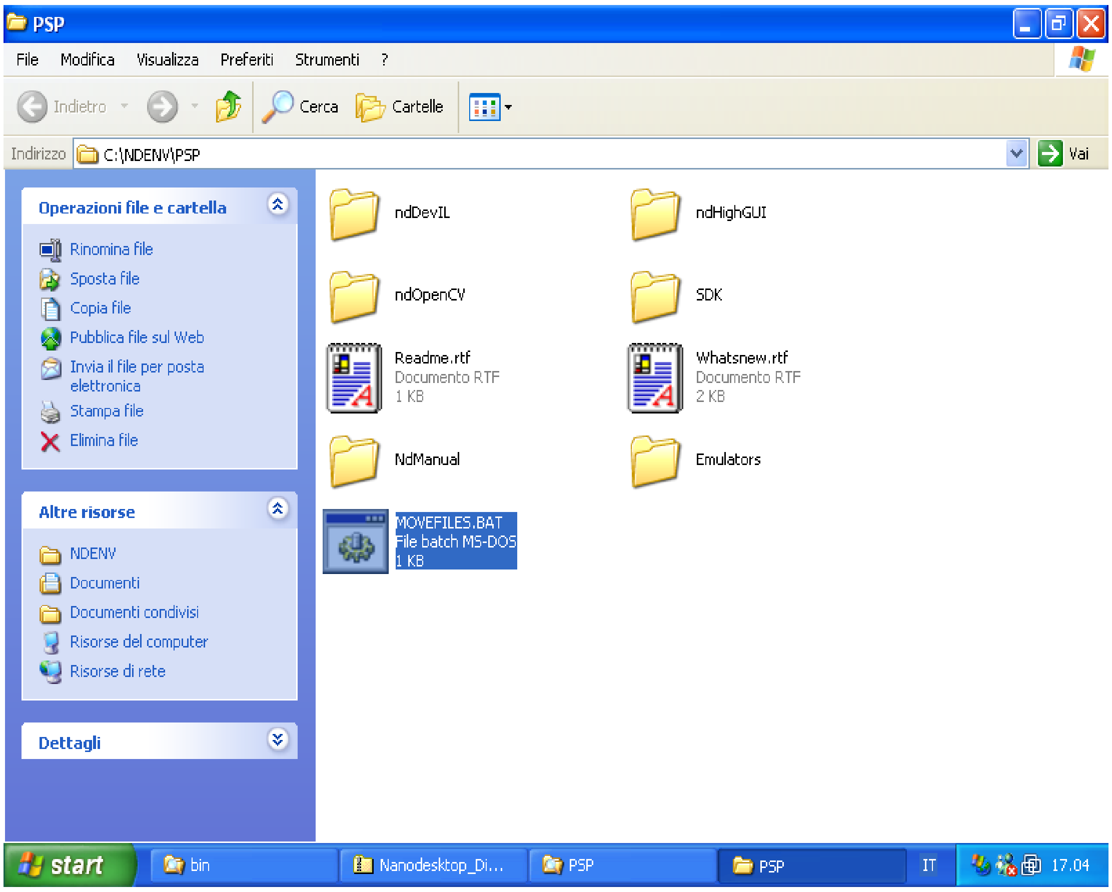
Nome variabile: NDCYGWIN_PATH

Valore variabile: C:\CYGWIN

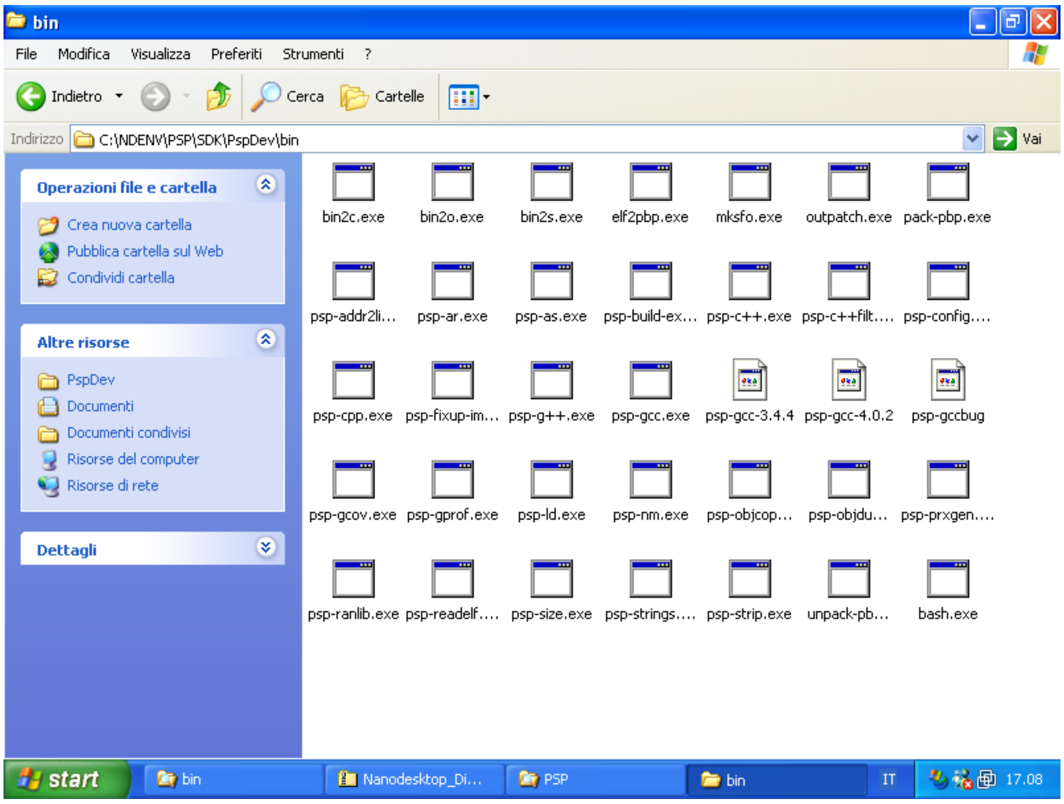
Variabili di sistema

Variabile	Valore
ComSpec	C:\WINDOWS\system32\cmd.exe
NDENV_PATH	C:\NDENV
NUMBER_OF_P...	1
OS	Windows_NT
Path	C:\WINDOWS\system32;C:\WINDOWS;...

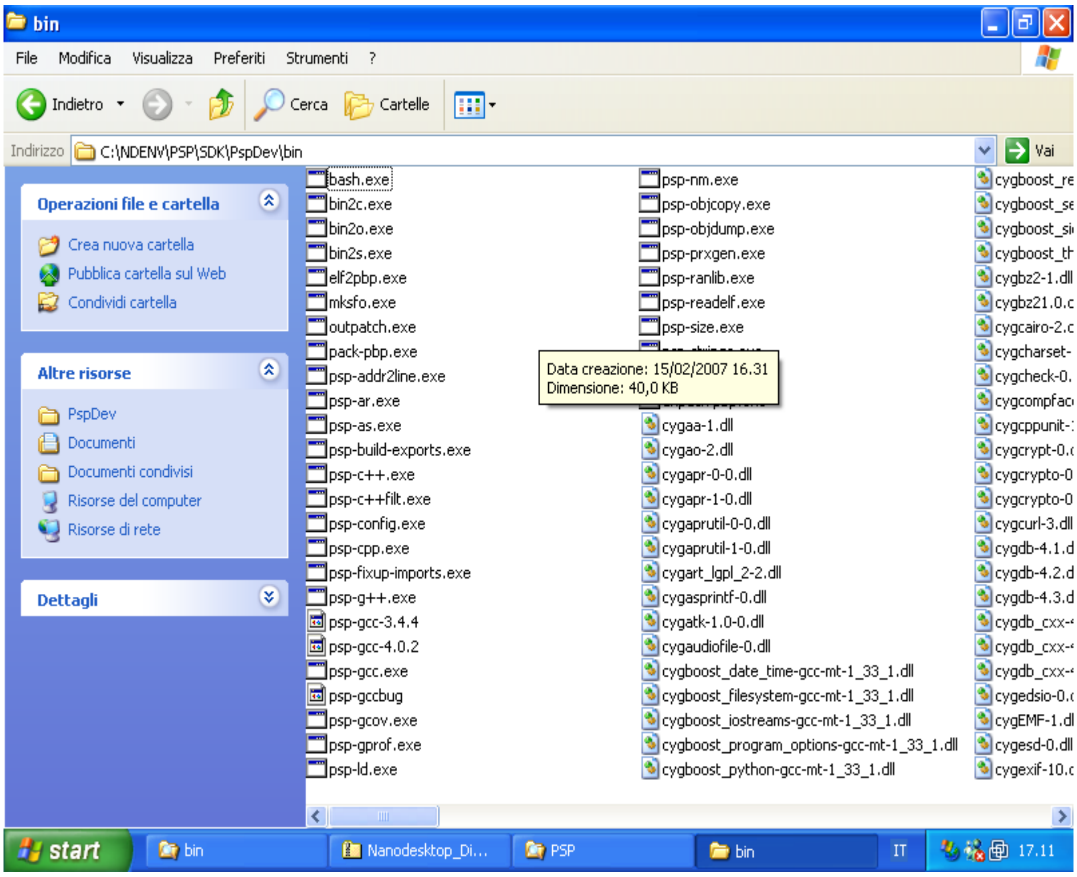
Now, you can use a script that we have prepared to copy the needed files. This script is called MOVEFILES.BAT and it is in <ndenv folder>/PSP


PSP
File Modifica Visualizza Preferiti Strumenti ?
Indietro Cerca Cartelle
Indirizzo C:\NDENV\PSP Vai
Operazioni file e cartella
Rinomina file
Sposta file
Copia file
Pubblica file sul Web
Invia il file per posta elettronica
Stampa file
Elimina file
Altre risorse
NDENV
Documenti
Documenti condivisi
Risorse del computer
Risorse di rete
Dettagli
ndDevIL
ndHighGUI
ndOpenCV
SDK
Readme.rtf Documento RTF 1 KB
Whatsnew.rtf Documento RTF 2 KB
NdManual
Emulators
MOVEFILES.BAT File batch MS-DOS 1 KB
start bin Nanodesktop_Di... PSP PSP IT 17.04

These are the files in the folder **<ndenv folder>\PSP\SDK\PSPDEV\bin,** before using MOVEFILES.BAT



Double click on MOVEFILES.BAT. The necessary files will be copied. This is what you should see at the end of the process



Indirizzo C:\NDENV\PSP\SDK\PspDev\bin

```
bash.exe                 psp-nm.exe                                     cygboost_re
bin2c.exe                psp-objcopy.exe                                cygboost_se
bin2o.exe                psp-objdump.exe                                cygboost_si
bin2s.exe                psp-prxgen.exe                                 cygboost_th
elf2pbp.exe              psp-ranlib.exe                                 cygbz2-1.dll
mksfo.exe                psp-readelf.exe                                cygbz21.0.c
outpatch.exe             psp-size.exe                                   cygcairo-2.c
pack-pbp.exe                                                            cygcharset-
psp-addr2line.exe                                                       cygcheck-0.
psp-ar.exe                                                              cygcompfac
psp-as.exe               cygaa-1.dll                                    cygcppunit-
psp-build-exports.exe    cygao-2.dll                                    cygcrypt-0.
psp-c++.exe              cygapr-0-0.dll                                 cygcrypto-0
psp-c++filt.exe          cygapr-1-0.dll                                 cygcrypto-0
psp-config.exe           cygaprutil-0-0.dll                             cygcurl-3.dll
psp-cpp.exe              cygaprutil-1-0.dll                             cygdb-4.1.d
psp-fixup-imports.exe    cygart_lgpl_2-2.dll                            cygdb-4.2.d
psp-g++.exe              cygasprintf-0.dll                              cygdb-4.3.d
psp-gcc-3.4.4            cygatk-1.0-0.dll                               cygdb_cxx-
psp-gcc-4.0.2            cygaudiofile-0.dll                             cygdb_cxx-
psp-gcc.exe              cygboost_date_time-gcc-mt-1_33_1.dll           cygdb_cxx-
psp-gccbug               cygboost_filesystem-gcc-mt-1_33_1.dll          cygedsio-0.
psp-gcov.exe             cygboost_iostreams-gcc-mt-1_33_1.dll           cygEMF-1.dl
psp-gprof.exe            cygboost_program_options-gcc-mt-1_33_1.dll     cygesd-0.dll
psp-ld.exe               cygboost_python-gcc-mt-1_33_1.dll              cygexif-10.c
```

Data creazione: 15/02/2007 16.31
Dimensione: 40,0 KB

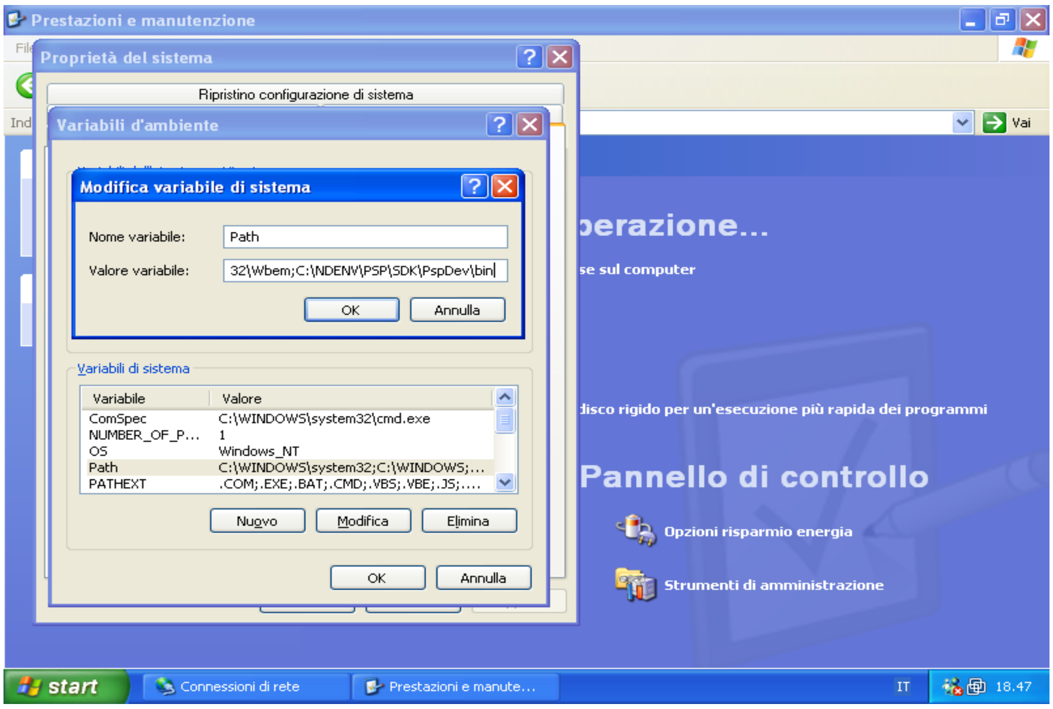
Stage 4: Set PATH environment variable

Now **you must add a new entry to the PATH variable:** the entry is

<ndenv>/PSP/SDK/PspDev/bin

where <ndenv> substitutes the folder where you have unzipped the Nanodesktop distribution. (in my example C:\NDENV).

For example:



Nome variabile: Path

Valore variabile: 32\Wbem;C:\NDENV\PSP\SDK\PspDev\bin

Variabile	Valore
ComSpec	C:\WINDOWS\system32\cmd.exe
NUMBER_OF_P...	1
OS	Windows_NT
Path	C:\WINDOWS\system32;C:\WINDOWS;...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....

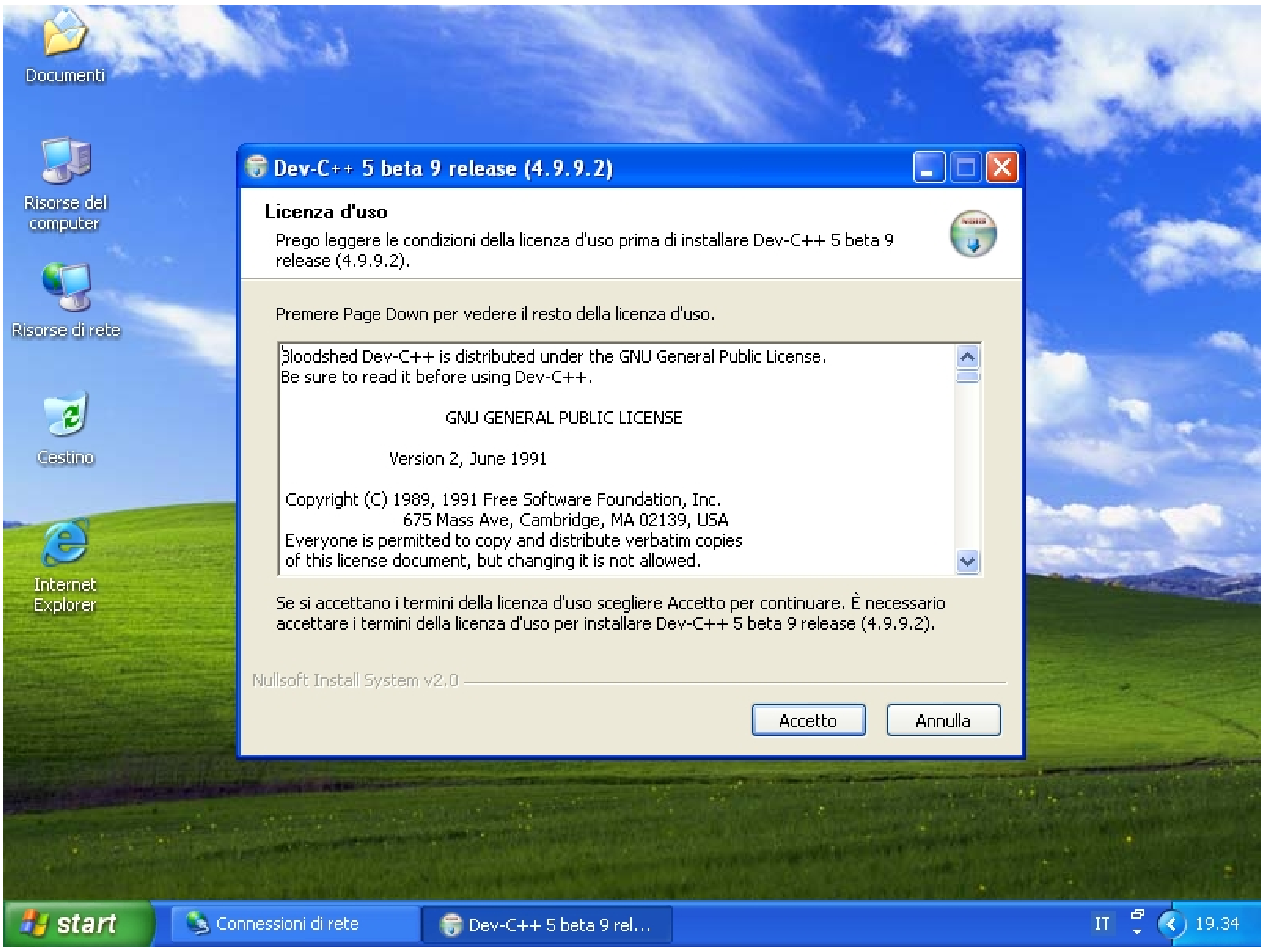
Stage 5: Install Dev-CPP

Nanodesktop has been developed to integrate itself into the Dev-CPP IDE. So you can develop your programs on your PC, test them using the PSPE emulator and then when they are complete, transfer them on to your PSP.

Download Dev-CPP from (www.bloodshed.net/devcpp.html).

The file is called devcpp-4.9.9.2_setup.exe and includes also Mingw env (DON'T USE executable only version).

Now run the exe and choose a path to install the IDE to, for example: **C:\PSPDEVCPP\**


Dev-C++ 5 beta 9 release (4.9.9.2)

Licenza d'uso

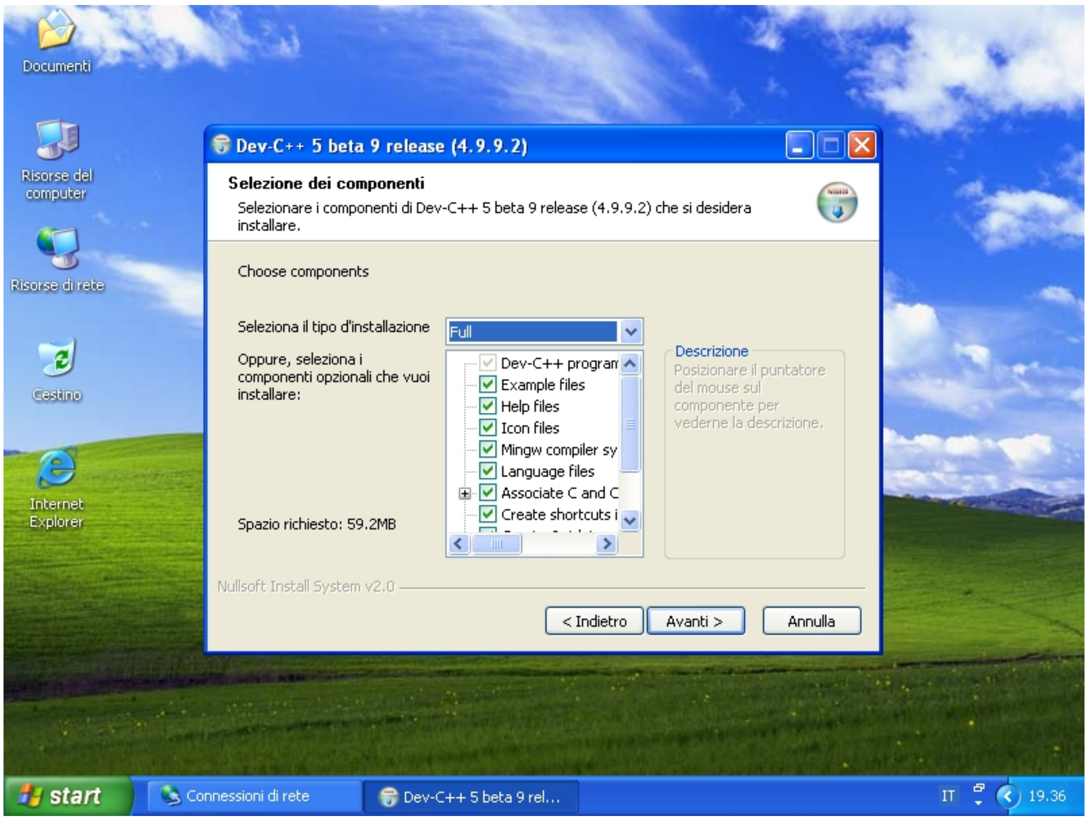
Prego leggere le condizioni della licenza d'uso prima di installare Dev-C++ 5 beta 9 release (4.9.9.2).

Premere Page Down per vedere il resto della licenza d'uso.

```
Bloodshed Dev-C++ is distributed under the GNU General Public License.
Be sure to read it before using Dev-C++.

                    GNU GENERAL PUBLIC LICENSE

              Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.
                  675 Mass Ave, Cambridge, MA 02139, USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.
```

Se si accettano i termini della licenza d'uso scegliere Accetto per continuare. È necessario accettare i termini della licenza d'uso per installare Dev-C++ 5 beta 9 release (4.9.9.2).

Complete the following installation steps and start Dev-CPP. In the next stage we'll configure the IDE to work with both the PSP and the PSPE emulator


Dev-C++ 5 beta 9 release (4.9.9.2)

Selezione dei componenti

Selezionare i componenti di Dev-C++ 5 beta 9 release (4.9.9.2) che si desidera installare.

Choose components

Seleziona il tipo d'installazione: Full

Oppure, seleziona i componenti opzionali che vuoi installare:

- ✓ Dev-C++ progran
- ✓ Example files
- ✓ Help files
- ✓ Icon files
- ✓ Mingw compiler sy
- ✓ Language files
- ✓ Associate C and C
- ✓ Create shortcuts i

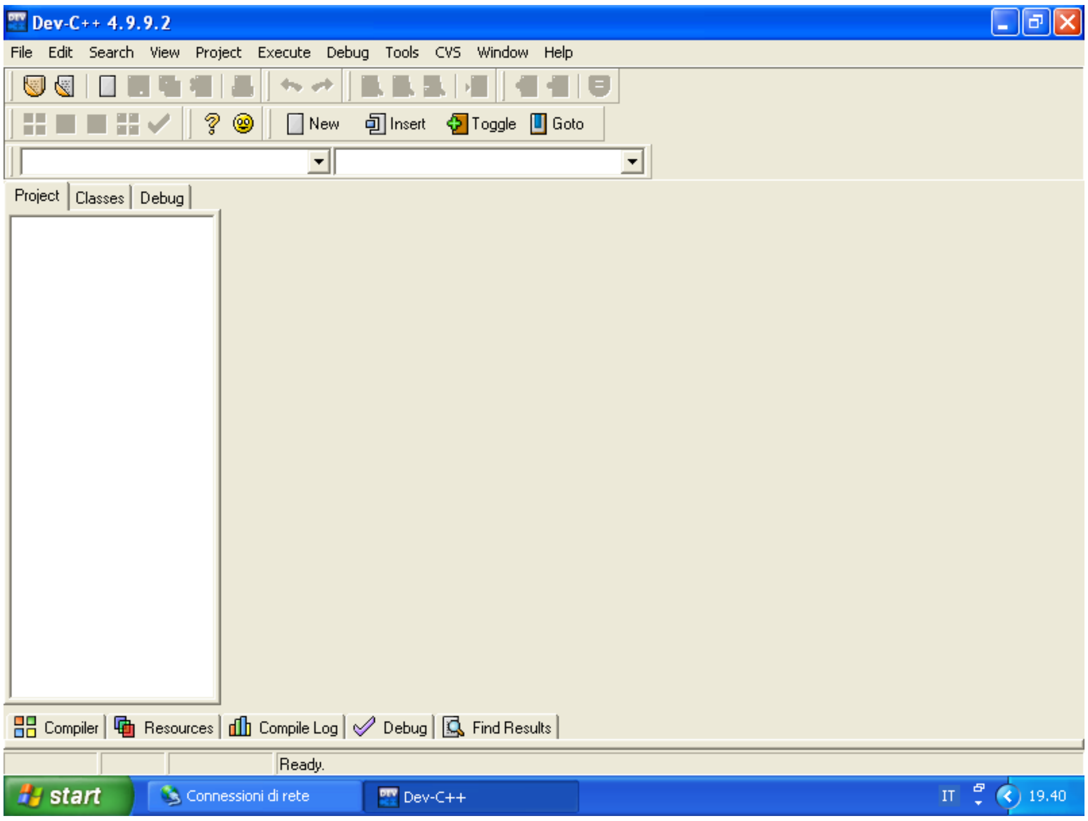
Descrizione: Posizionare il puntatore del mouse sul componente per vederne la descrizione.

Spazio richiesto: 59.2MB

Nullsoft Install System v2.0

Remember that you must choose **Full installation** of DEV-CPP.

When you start DEV-CPP for the first time this is what you'll see


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
Compiler Resources Compile Log Debug Find Results
Ready.
start
Connessioni di rete
Dev-C++
19.40

Stage 6: Install PSPE (PSP Emulator)

PSPE Emulator is a pseudo-emulator that emulates some functions of a real PSP. We can't redistribute the PSPE binary due to license issues

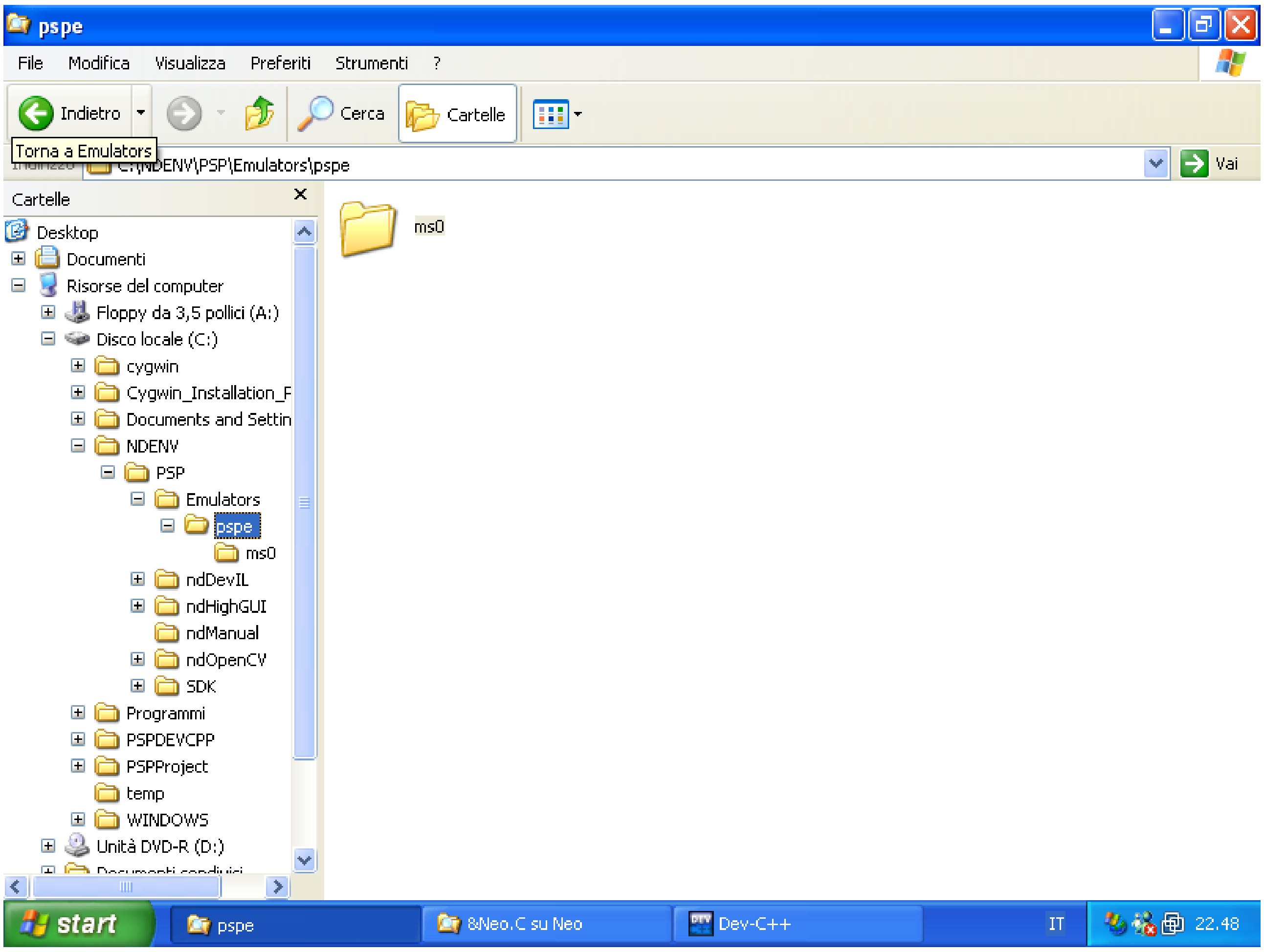
So you'll have to download the PSPE binary yourself.

You can download PSPE from:
http://psp-news.dcemu.co.uk/files/pspe09b.zip

Remember that you'll also need libsdl32 for win32, downloadable at:

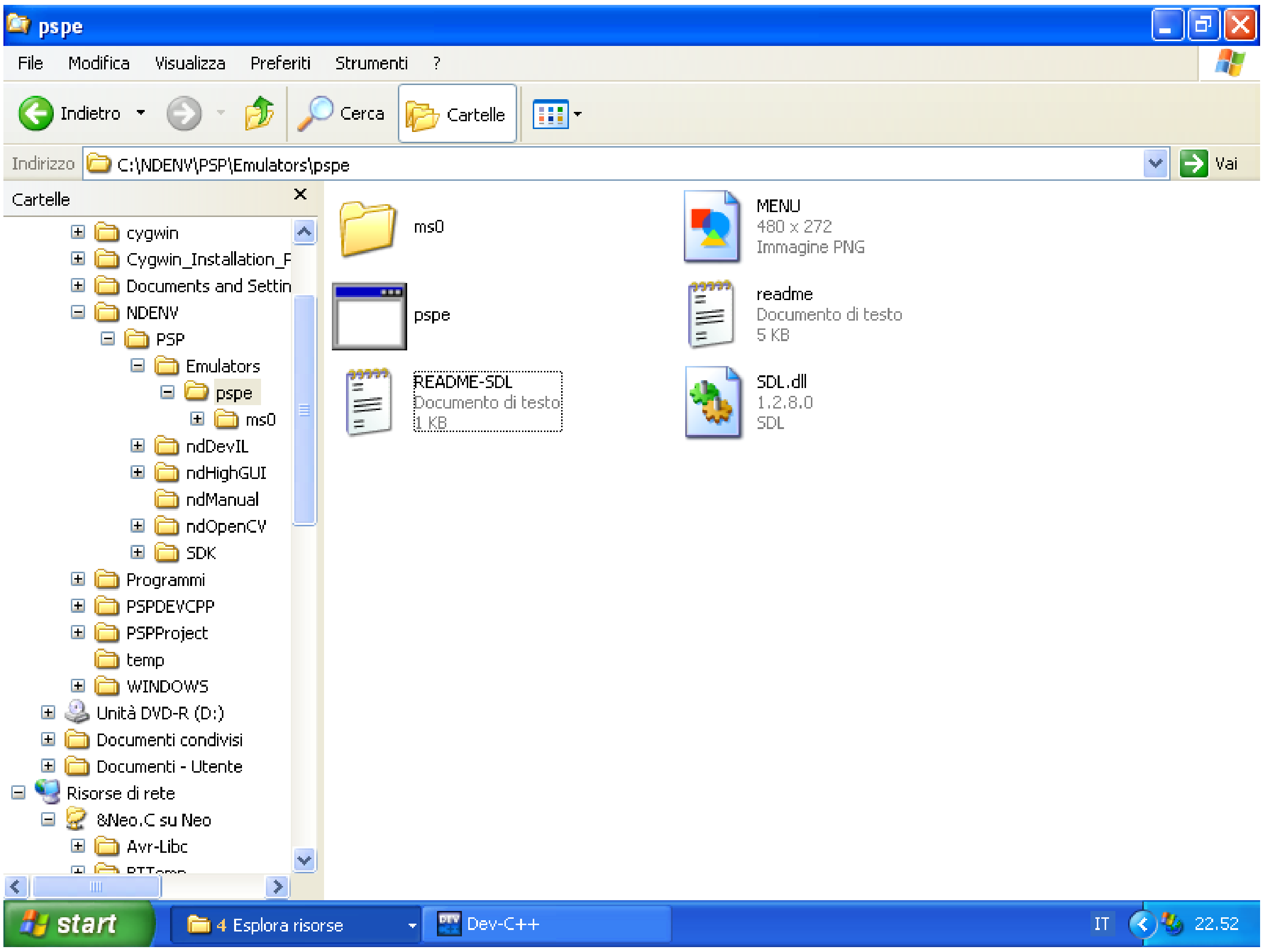
http://www.libsdl.org/release/SDL-1.2.8-win32.zip

Now you must go to a precise folder in the Nanodesktop file system:



In **<ndenv folder>/PSP/Emulators/pspe you'll find** the folder where you have to decompress the pspe.zip archive. Afterwards you'll have to decompress the sdl archive in the same folder and PSPE is ready to work.

This is how the folder should look upon completion:



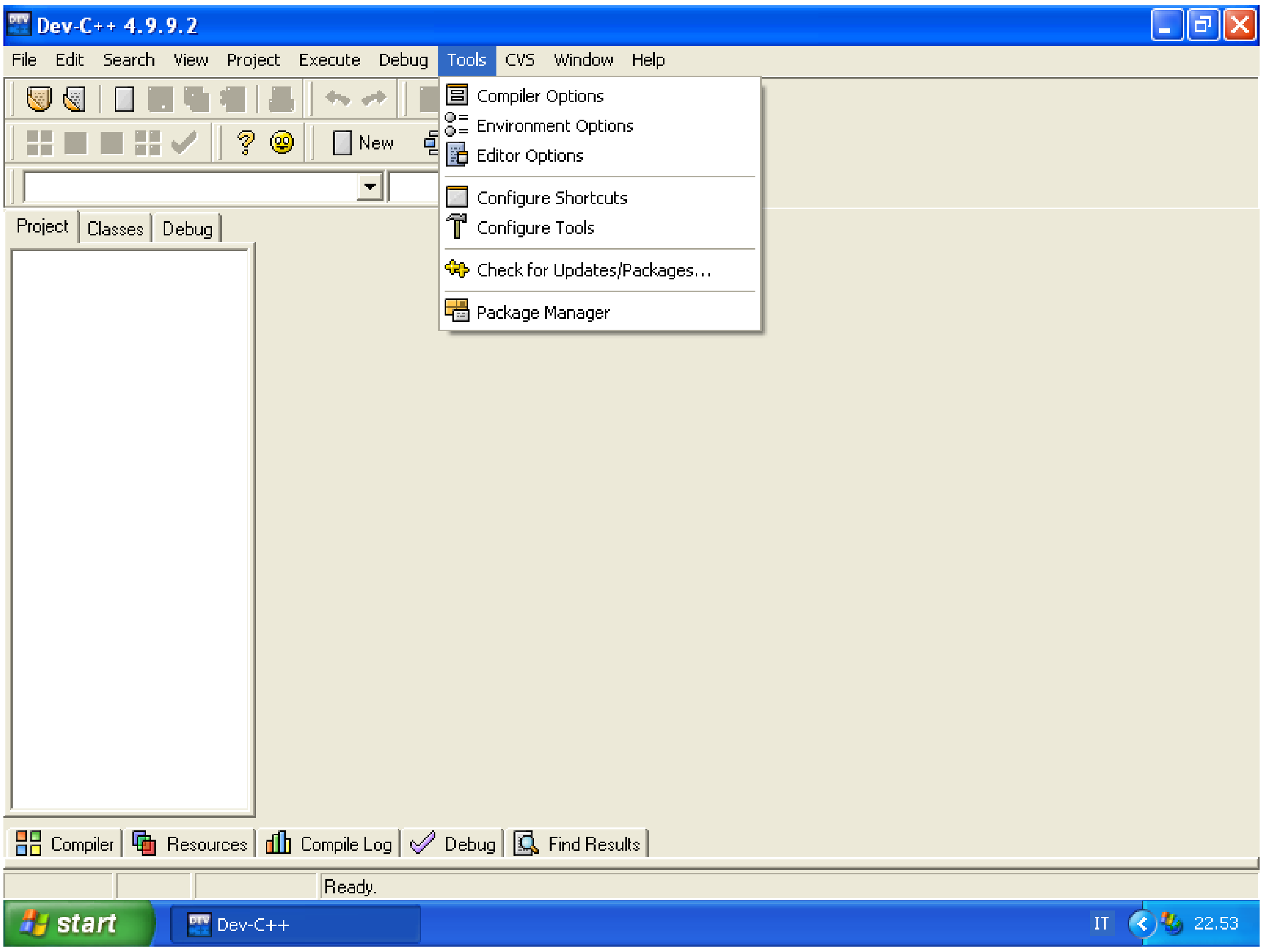
Stage 7: Configure Dev-CPP to work with Nanodesktop

Now we are ready to configure Dev-CPP.

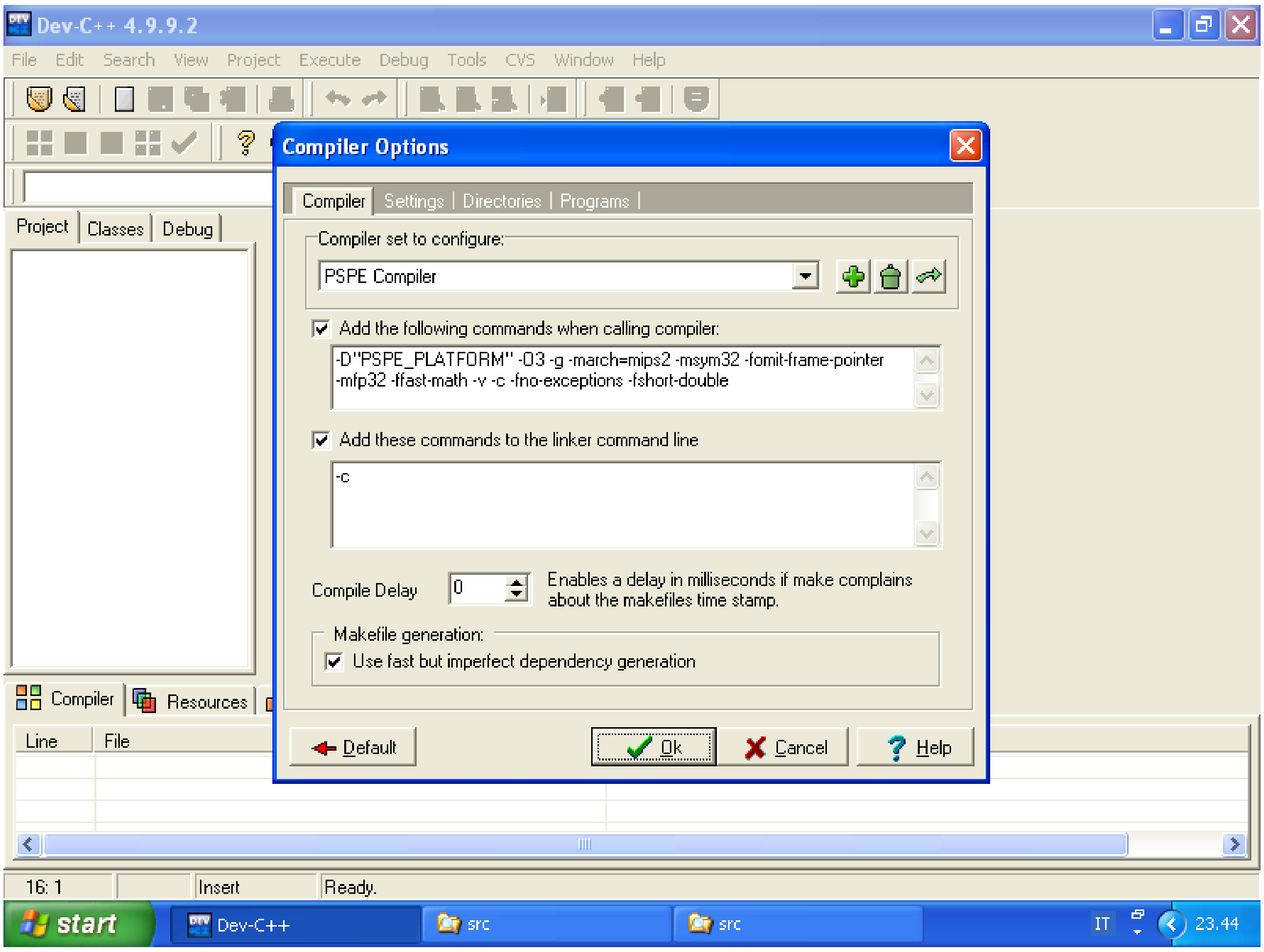
The first thing we have to do is create two new *compiler sets,* one for PSPE and another for the PSP.

Compiler set for PSPE

Go to *Compiler options:*



Now we can create the first compiler set


Compiler set to configure: PSPE Compiler

☑ Add the following commands when calling compiler:

```
-D"PSPE_PLATFORM" -O3 -g -march=mips2 -msym32 -fomit-frame-pointer
-mfp32 -ffast-math -v -c -fno-exceptions -fshort-double
```

☑ Add these commands to the linker command line

```
-c
```

Compile Delay 0 — Enables a delay in milliseconds if make complains about the makefiles time stamp.

Makefile generation: ☑ Use fast but imperfect dependency generation

Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
Compiler Options
Compiler | Settings | Directories | Programs
Project Classes Debug
Compiler Resources
Line File
Default
Ok
Cancel
Help
16: 1 Insert Ready.
start Dev-C++ src src IT 23.44

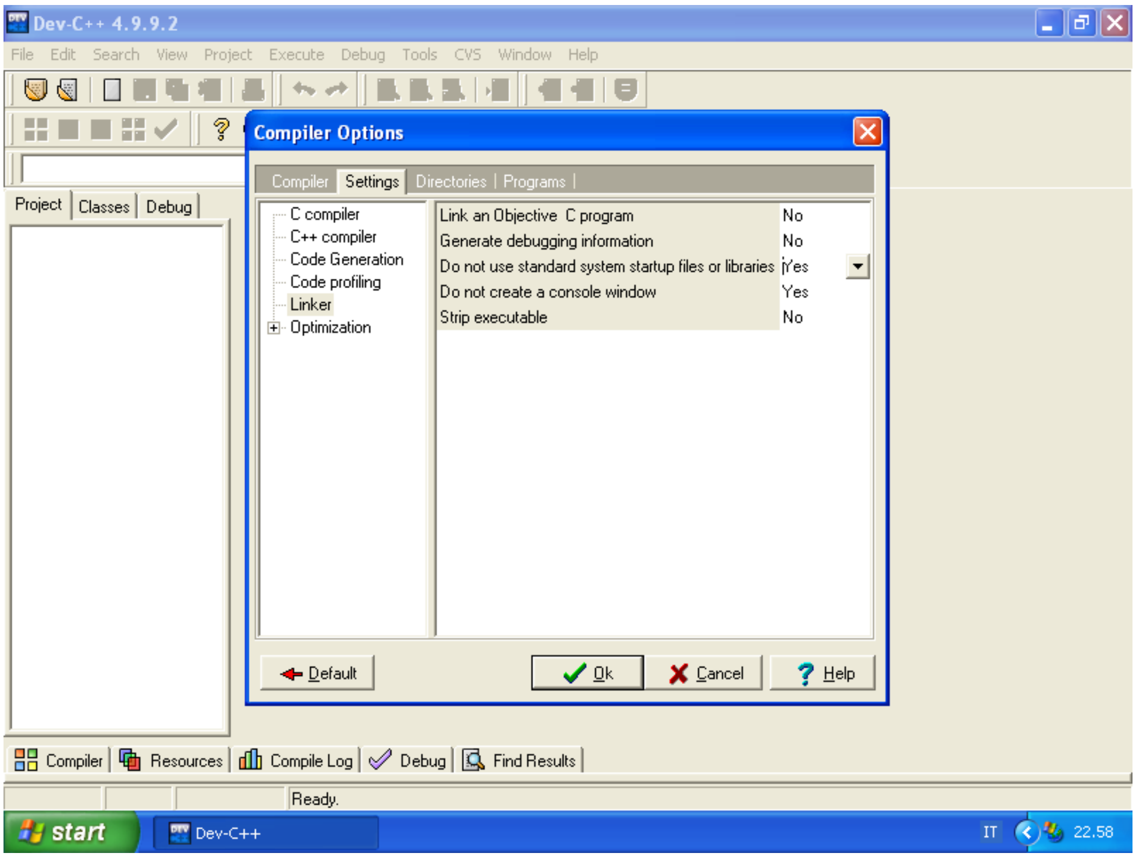
Create a new set called **PSPE Compiler set**

Compiler flags must be:

-D"PSPE_PLATFORM" -O3 -g -march=mips2 -msym32 -fomit-frame-pointer -mfp32 -ffast-math -v -c -fno-exceptions -fshort-double

Linker options must be:

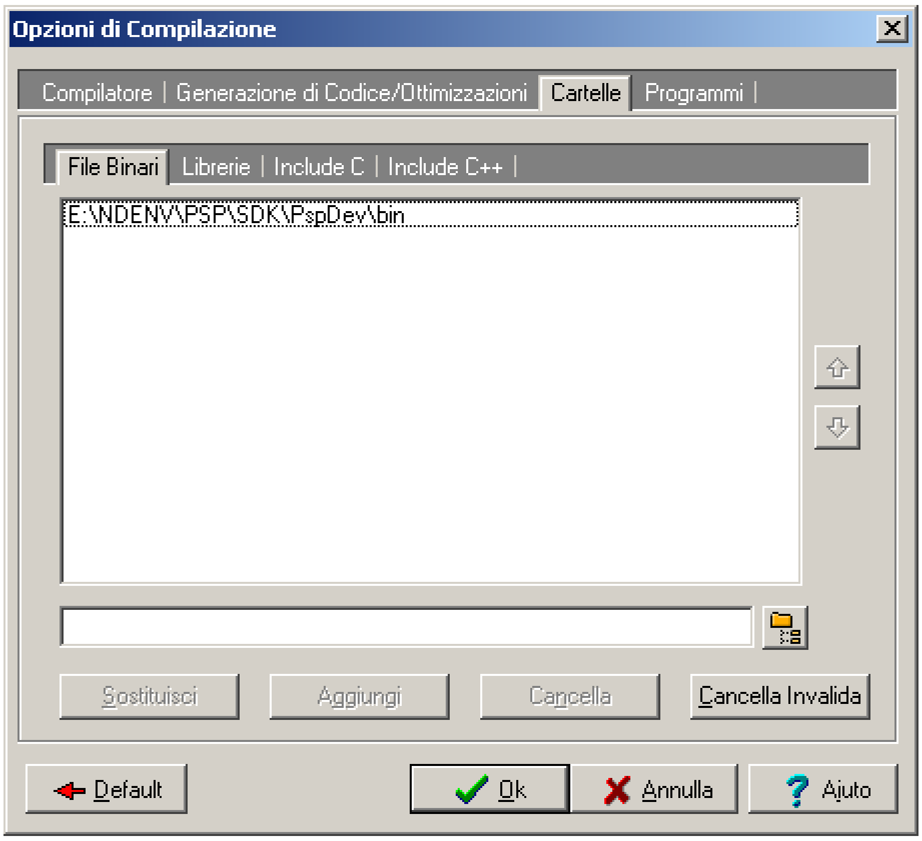
-c (this will avoid Dev-CPP calling its own linker to link binary code).


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
Compiler Options
Compiler | Settings | Directories | Programs
C compiler
C++ compiler
Code Generation
Code profiling
Linker
Optimization
Default
Ok
Cancel
Help
Project | Classes | Debug
Compiler | Resources | Compile Log | Debug | Find Results
Ready.
start
Dev-C++
IT 22.58

Link an Objective C program	No
Generate debugging information	No
Do not use standard system startup files or libraries	Yes
Do not create a console window	Yes
Strip executable	No

Configure linker using

> **Don’t use standard lib for system boot**
> **Don’t open console**


Opzioni di Compilazione
Compilatore | Generazione di Codice/Ottimizzazioni | Cartelle | Programmi
File Binari | Librerie | Include C | Include C++
Sostituisci
Aggiungi
Cancella
Cancella Invalida
Default
Ok
Annulla
Aiuto

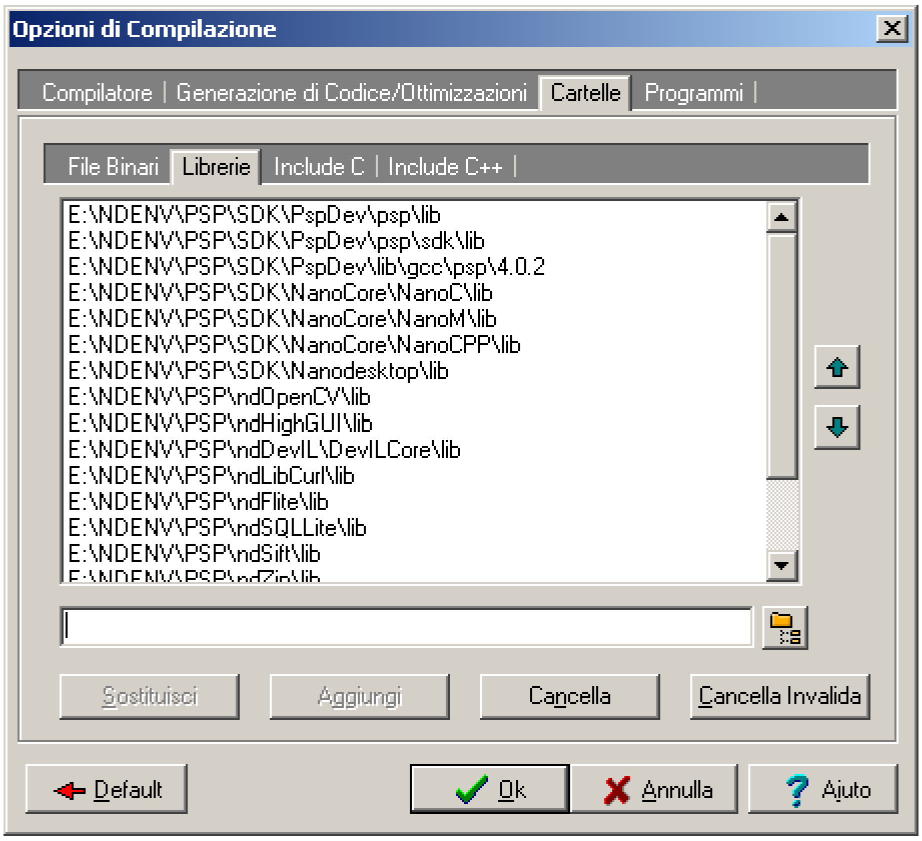
E:\NDENV\PSP\SDK\PspDev\bin

Now to configure folders: consider that I have installed (in my example) Nanodesktop in E:\NDENV. Replace the path with your chosen path.

If you have installed Nanodesktop in <ndenv folder> the first entry that you must enter is

<ndenv folder>\PSP\SDK\PspDev\bin

Now we have to configure the folders in which the system must look for the libraries.



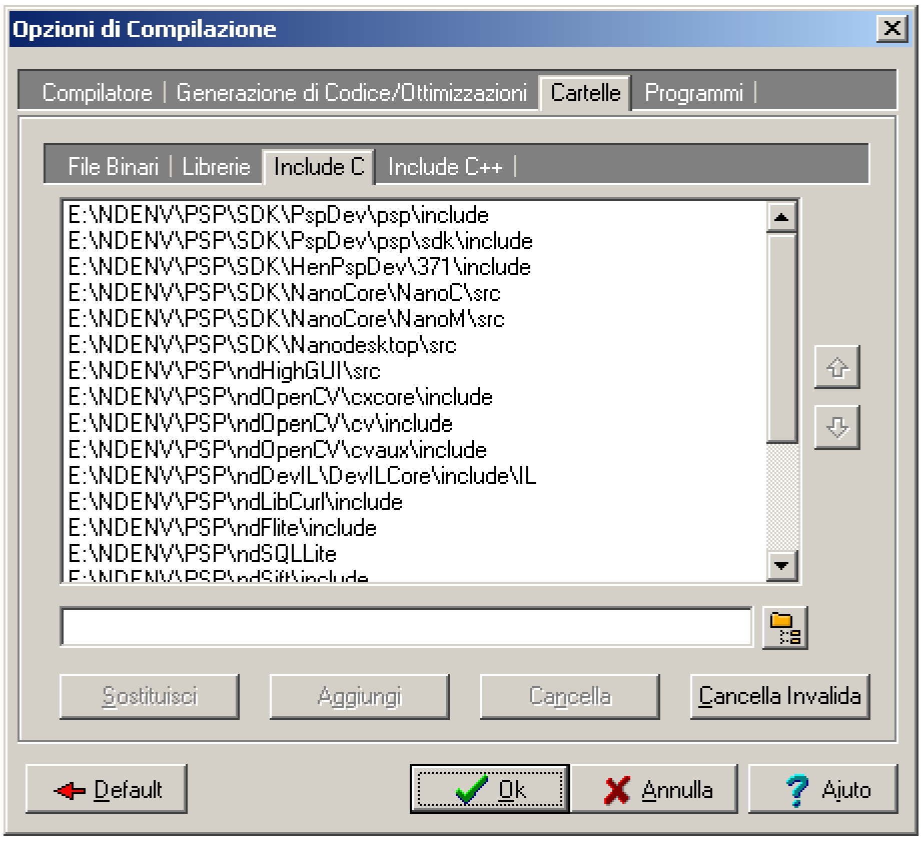
```
E:\NDENV\PSP\SDK\PspDev\psp\lib
E:\NDENV\PSP\SDK\PspDev\psp\sdk\lib
E:\NDENV\PSP\SDK\PspDev\lib\gcc\psp\4.0.2
E:\NDENV\PSP\SDK\NanoCore\NanoC\lib
E:\NDENV\PSP\SDK\NanoCore\NanoM\lib
E:\NDENV\PSP\SDK\NanoCore\NanoCPP\lib
E:\NDENV\PSP\SDK\Nanodesktop\lib
E:\NDENV\PSP\ndOpenCV\lib
E:\NDENV\PSP\ndHighGUI\lib
E:\NDENV\PSP\ndDevIL\DevILCore\lib
E:\NDENV\PSP\ndLibCurl\lib
E:\NDENV\PSP\ndFlite\lib
E:\NDENV\PSP\ndSQLLite\lib
E:\NDENV\PSP\ndSift\lib
E:\NDENV\PSP\ndZip\lib
```

The folders are:

<ndenv folder>\PSP\SDK\PspDev\psp\lib
<ndenv folder>\PSP\SDK\PspDev\psp\sdk\lib
<ndenv folder>\PSP\SDK\PspDev\lib\gcc\psp\4.0.2
<ndenv folder>\PSP\SDK\NanoCore\NanoC\lib
<ndenv folder>\PSP\SDK\NanoCore\NanoM\lib
<ndenv folder>\PSP\SDK\NanoCore\NanoCPP\lib
<ndenv folder>\PSP\SDK\Nanodesktop\lib
<ndenv folder>\PSP\ndOpenCV\lib
<ndenv folder>\PSP\ndHighGUI\lib
<ndenv folder>\PSP\ndDevIL\DevILCore\lib
<ndenv folder>\PSP\ndLibCurl\lib
<ndenv folder>\PSP\ndFLite\lib
<ndenv folder>\PSP\ndSQLLite\lib
<ndenv folder>\PSP\ndSIFT\lib
<ndenv folder>\PSP\ndZip\lib
<ndenv folder>\PSP\ndFreeType\lib
<ndenv folder>\PSP\ndOCRAD\lib
<ndenv folder>\PSP\ndGOCR\lib

Replace <ndenv folder> with the folder which the Nanodesktop package is installed to. (the folder in blue can be skipped if you don't use the associated component).

Now we'll set the include folders for the C compiler:

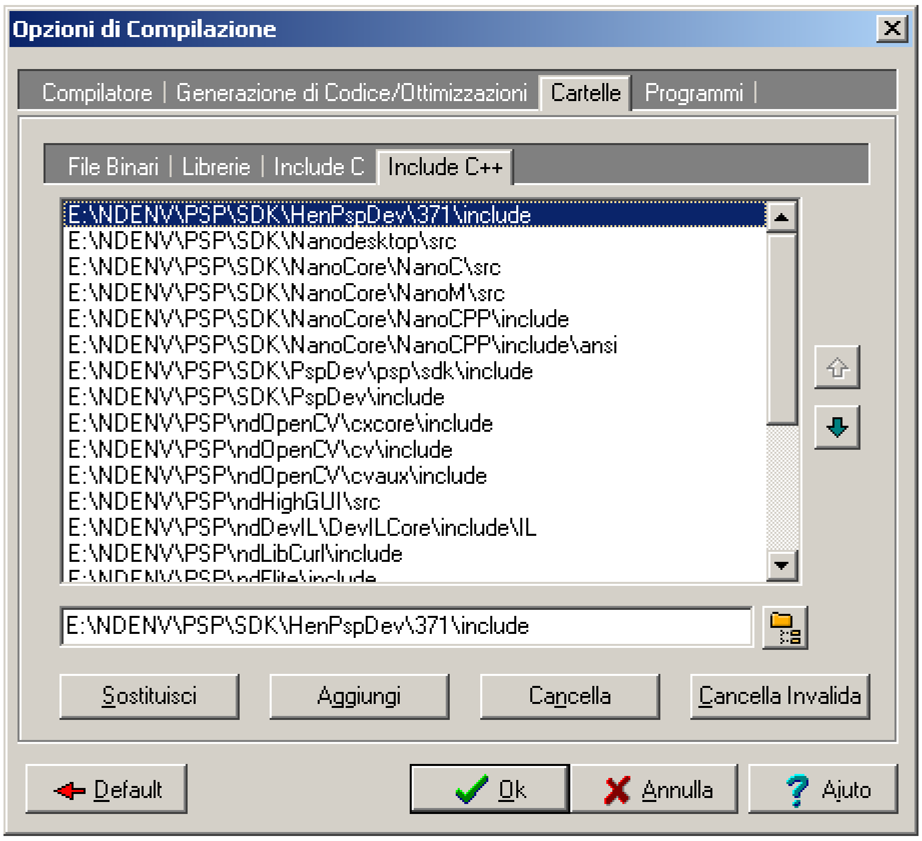

Opzioni di Compilazione
Compilatore | Generazione di Codice/Ottimizzazioni | Cartelle | Programmi
File Binari | Librerie | Include C | Include C++
Sostituisci
Aggiungi
Cancella
Cancella Invalida
Default
Ok
Annulla
Aiuto

```
E:\NDENV\PSP\SDK\PspDev\psp\include
E:\NDENV\PSP\SDK\PspDev\psp\sdk\include
E:\NDENV\PSP\SDK\HenPspDev\371\include
E:\NDENV\PSP\SDK\NanoCore\NanoC\src
E:\NDENV\PSP\SDK\NanoCore\NanoM\src
E:\NDENV\PSP\SDK\Nanodesktop\src
E:\NDENV\PSP\ndHighGUI\src
E:\NDENV\PSP\ndOpenCV\cxcore\include
E:\NDENV\PSP\ndOpenCV\cv\include
E:\NDENV\PSP\ndOpenCV\cvaux\include
E:\NDENV\PSP\ndDevIL\DevILCore\include\IL
E:\NDENV\PSP\ndLibCurl\include
E:\NDENV\PSP\ndFlite\include
E:\NDENV\PSP\ndSQLLite
E:\NDENV\PSP\ndSift\include
```

The folders are:

<ndenv folder>\PSP\SDK\PspDev\psp\include
<ndenv folder>\PSP\SDK\PspDev\psp\sdk\include
<ndenv folder>\PSP\SDK\HenPspDev\371\include
<ndenv folder>\PSP\SDK\NanoCore\NanoC\src
<ndenv folder>\PSP\SDK\NanoCore\NanoM\src
<ndenv folder>\PSP\SDK\Nanodesktop\src
<ndenv folder>\PSP\ndHighGUI\src
<ndenv folder>\PSP\ndOpenCV\cxcore\include
<ndenv folder>\PSP\ndOpenCV\cv\include
<ndenv folder>\PSP\ndOpenCV\cvaux\include
<ndenv folder>\PSP\ndDevIL\DevILCore\include\IL
<ndenv folder>\PSP\ndLibCurl\include
<ndenv folder>\PSP\ndFLite\include
<ndenv folder>\PSP\ndFLite\include
<ndenv folder>\PSP\ndSIFT\include
<ndenv folder>\PSP\ndFreeType\include
<ndenv folder>\PSP\ndFreeType\include\freetype
<ndenv folder>\PSP\ndGOCR\include
<ndenv folder>\PSP\ndOCRAD
<ndenv folder>\PSP\ndZip
<ndenv folder>\PSP\ndISpell

(the folder in blue can be skipped if you don't use the associated component).


Opzioni di Compilazione
Compilatore | Generazione di Codice/Ottimizzazioni | Cartelle | Programmi
File Binari | Librerie | Include C | Include C++
Sostituisci
Aggiungi
Cancella
Cancella Invalida
Default
Ok
Annulla
Aiuto

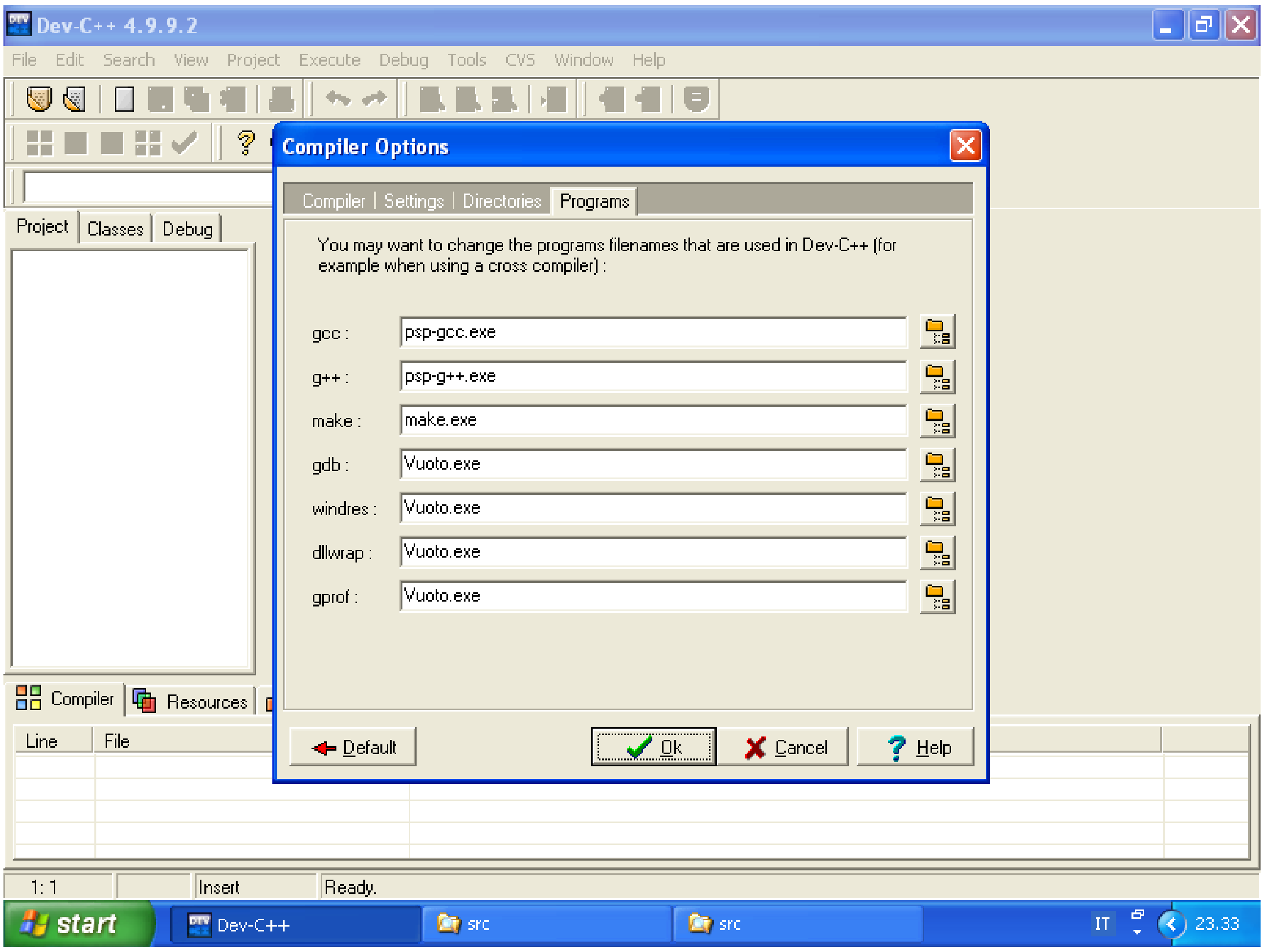
```
E:\NDENV\PSP\SDK\HenPspDev\371\include
E:\NDENV\PSP\SDK\Nanodesktop\src
E:\NDENV\PSP\SDK\NanoCore\NanoC\src
E:\NDENV\PSP\SDK\NanoCore\NanoM\src
E:\NDENV\PSP\SDK\NanoCore\NanoCPP\include
E:\NDENV\PSP\SDK\NanoCore\NanoCPP\include\ansi
E:\NDENV\PSP\SDK\PspDev\psp\sdk\include
E:\NDENV\PSP\SDK\PspDev\include
E:\NDENV\PSP\ndOpenCV\cxcore\include
E:\NDENV\PSP\ndOpenCV\cv\include
E:\NDENV\PSP\ndOpenCV\cvaux\include
E:\NDENV\PSP\ndHighGUI\src
E:\NDENV\PSP\ndDevIL\DevILCore\include\IL
E:\NDENV\PSP\ndLibCurl\include
E:\NDENV\PSP\ndFlite\include

E:\NDENV\PSP\SDK\HenPspDev\371\include
```

The folders for C++ are (respect the order!)

<ndenv folder>\PSP\SDK\HenPspDev\371\include
<ndenv folder>\PSP\SDK\Nanodesktop\src
<ndenv folder>\PSP\SDK\NanoCore\NanoC\src
<ndenv folder>\PSP\SDK\NanoCore\NanoM\src
<ndenv folder>\PSP\SDK\NanoCore\NanoCPP\include
<ndenv folder>\PSP\SDK\NanoCore\NanoCPP\include\ansi
<ndenv folder>\PSP\SDK\PspDev\psp\sdk\include
<ndenv folder>\PSP\SDK\PspDev\include
<ndenv folder>\PSP\ndOpenCV\cxcore\include
<ndenv folder>\PSP\ndOpenCV\cv\include
<ndenv folder>\PSP\ndOpenCV\cvaux\include
<ndenv folder>\PSP\ndHighGUI\src
<ndenv folder>\PSP\ndDevIL\DevILCore\include\IL
<ndenv folder>\PSP\ndLibCurl\include
<ndenv folder>\PSP\ndFlite\include
<ndenv folder>\PSP\ndSQLLite
<ndenv folder>\PSP\ndSift\include
<ndenv folder>\PSP\ndFreeType\include
<ndenv folder>\PSP\ndFreeType\include\freetype
<ndenv folder>\PSP\ndGOCR\include
<ndenv folder>\PSP\ndOCRAD
<ndenv folder>\PSP\ndZip
<ndenv folder>\PSP\ndIspell

(the folder in blue can be skipped if you don't use the associated component).

Configuring the programs:


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
Compiler Options
Compiler | Settings | Directories | Programs
Project Classes Debug
Compiler Resources
Line File
Default Ok Cancel Help
1: 1 Insert Ready.
start Dev-C++ src src IT 23.33

You may want to change the programs filenames that are used in Dev-C++ (for example when using a cross compiler) :

gcc : psp-gcc.exe
g++ : psp-g++.exe
make : make.exe
gdb : Vuoto.exe
windres : Vuoto.exe
dllwrap : Vuoto.exe
gprof : Vuoto.exe

The programs psp-gcc and psp-g++ are in:

<ndenv folder>\PSP\SDK\PspDev\bin

The program *vuoto.exe* is provided with Nanodesktop and it is in:

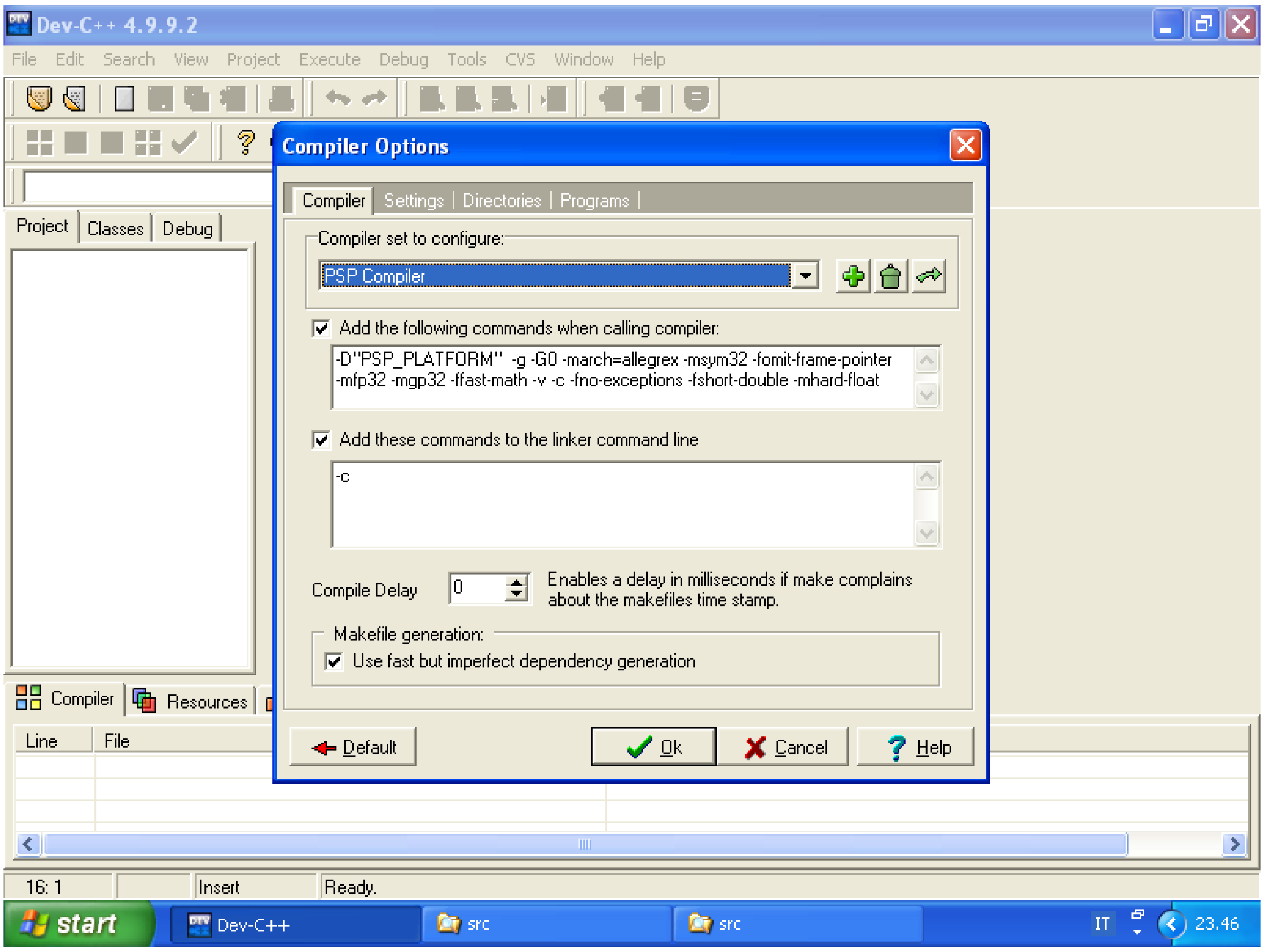
<ndenv folder>\PSP\SDK\ServiceFiles

The program *make.exe* is provided by cygwin and it is in:

<cygwin folder>\bin

Compiler set for PSP

The procedure to create a compiler set for the PSP is the same used for the PSPE emulator. The only difference is in *Compiler options for PSP-GCC.*



Compiler set to configure: PSP Compiler

[✓] Add the following commands when calling compiler:

```
-D"PSP_PLATFORM"  -g -G0 -march=allegrex -msym32 -fomit-frame-pointer
-mfp32 -mgp32 -ffast-math -v -c -fno-exceptions -fshort-double -mhard-float
```

[✓] Add these commands to the linker command line

```
-c
```

Compile Delay 0 — Enables a delay in milliseconds if make complains about the makefiles time stamp.

Makefile generation:
[✓] Use fast but imperfect dependency generation

For the PSP you must use:

-D"PSP_PLATFORM" -O2 -g -G0 -march=allegrex -fomit-frame-pointer -fno-exceptions -mgp32 -mfp32 -msym32 -ffast-math -mhard-float -v –c -fshort-double

When you have created the PSP and PSPE Compiler sets you are ready to compile your first Nanodesktop program.

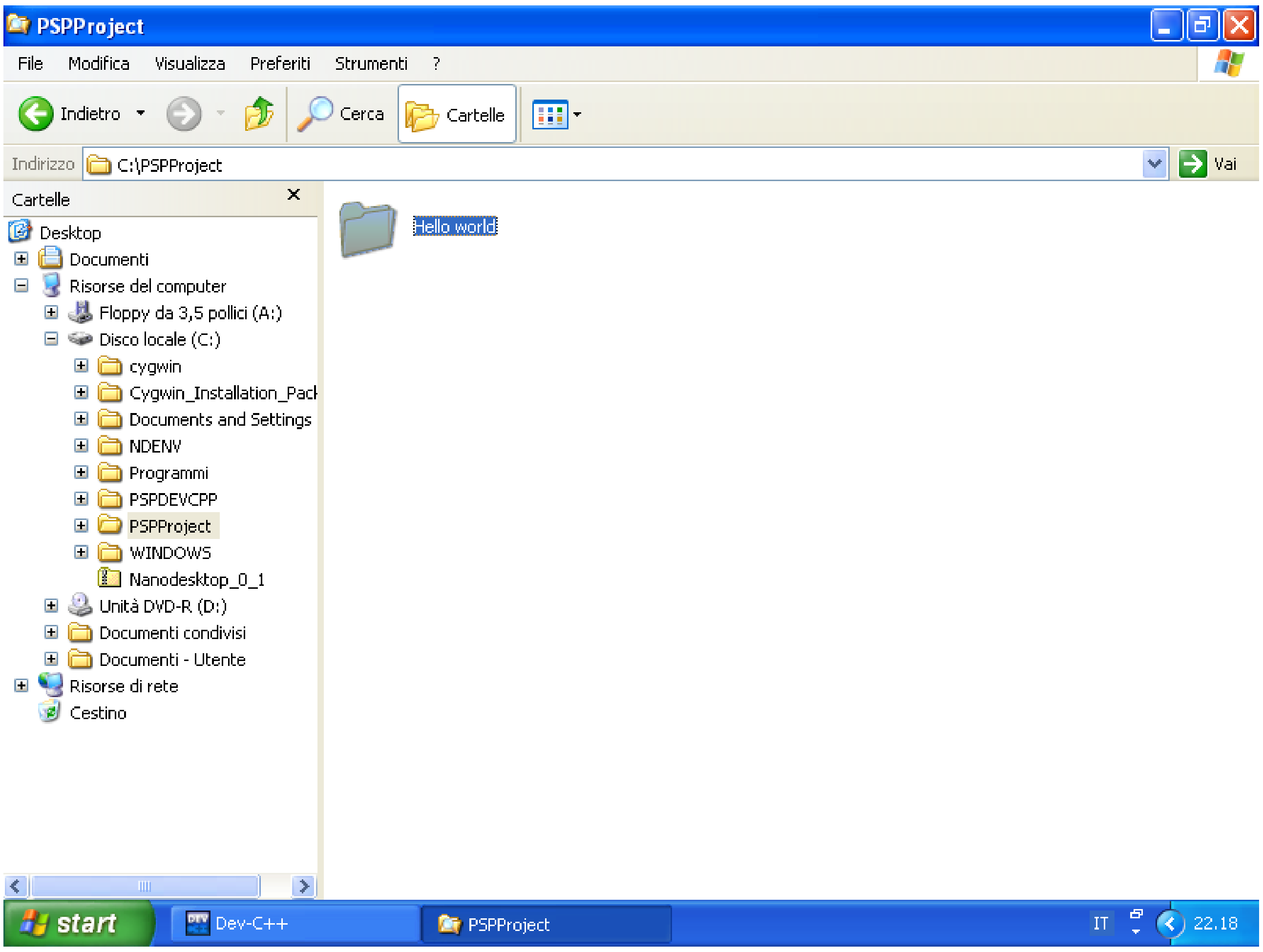
Note: You can use also -O3 to improve optimization of the program, but it can (in rare case) determinates hangs of your program at startup.

Chapter 3

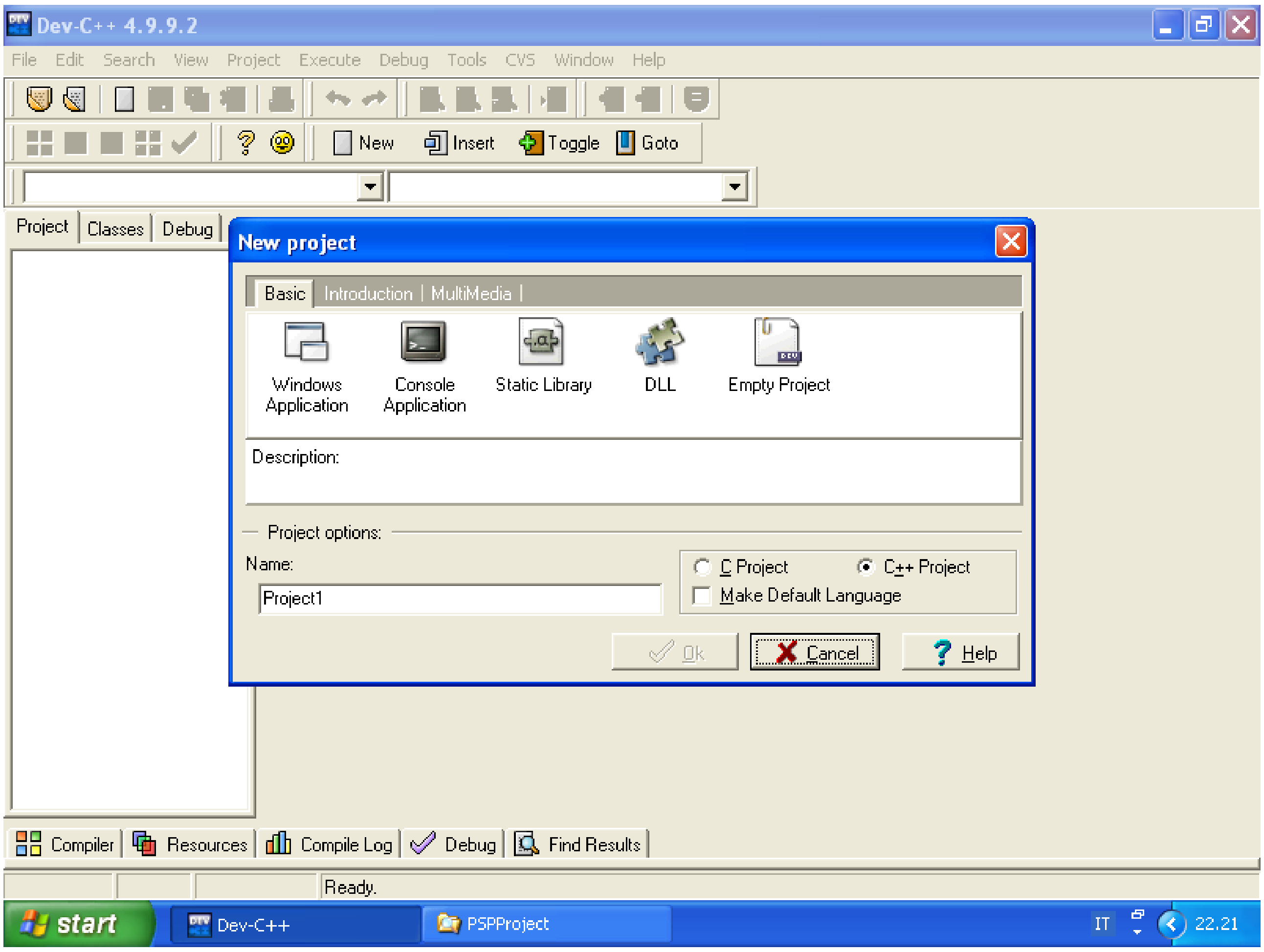
Your first Nanodesktop program

Now you are ready to create your first Nanodesktop program. Create a folder for your PSP Projects and in that folder a subfolder for your first program.

All Nanodesktop programs must have a separate folder.



Now, open Dev-C++ and choose *File/New* and select *New Project:*

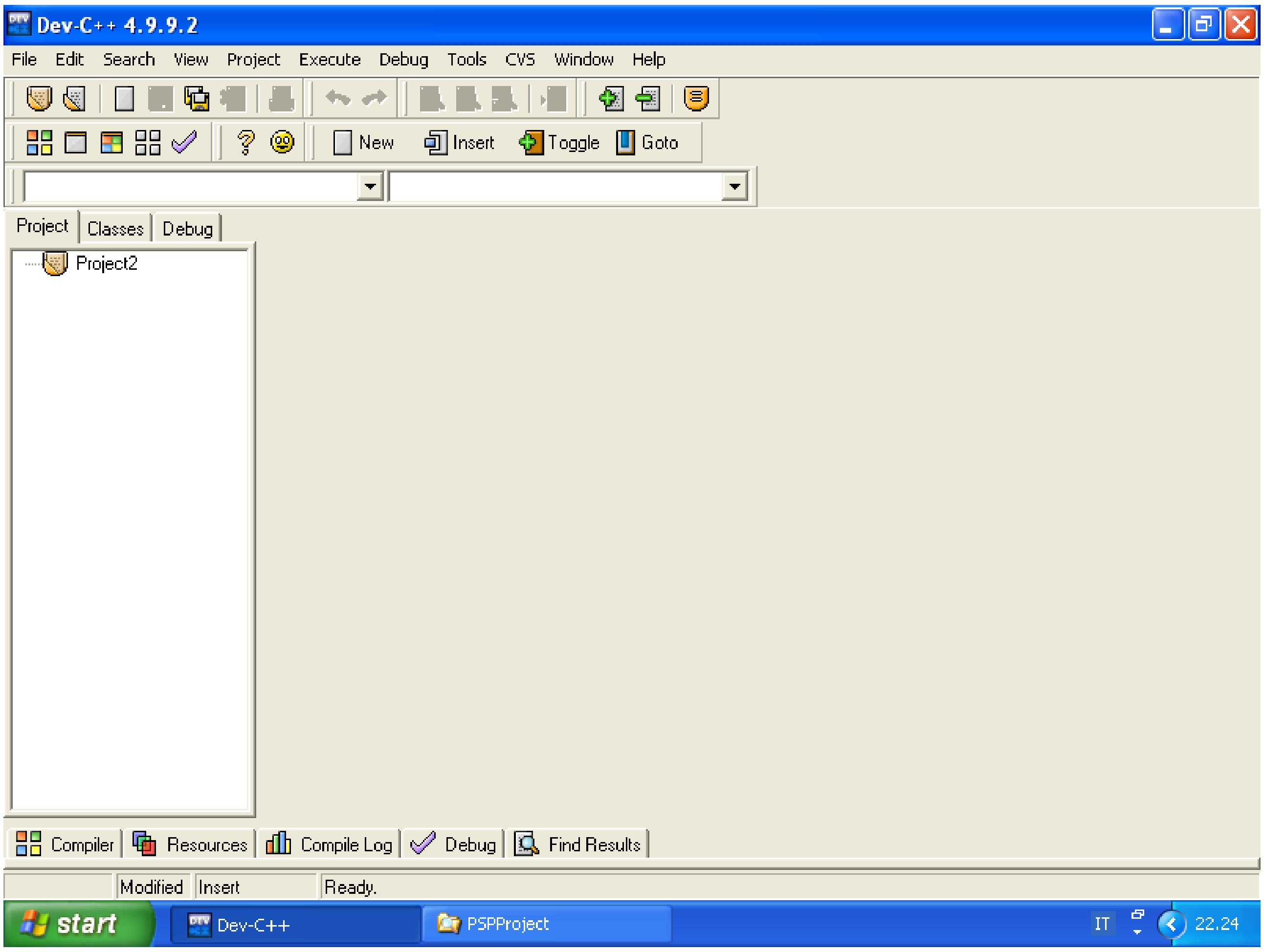


Name: Project1

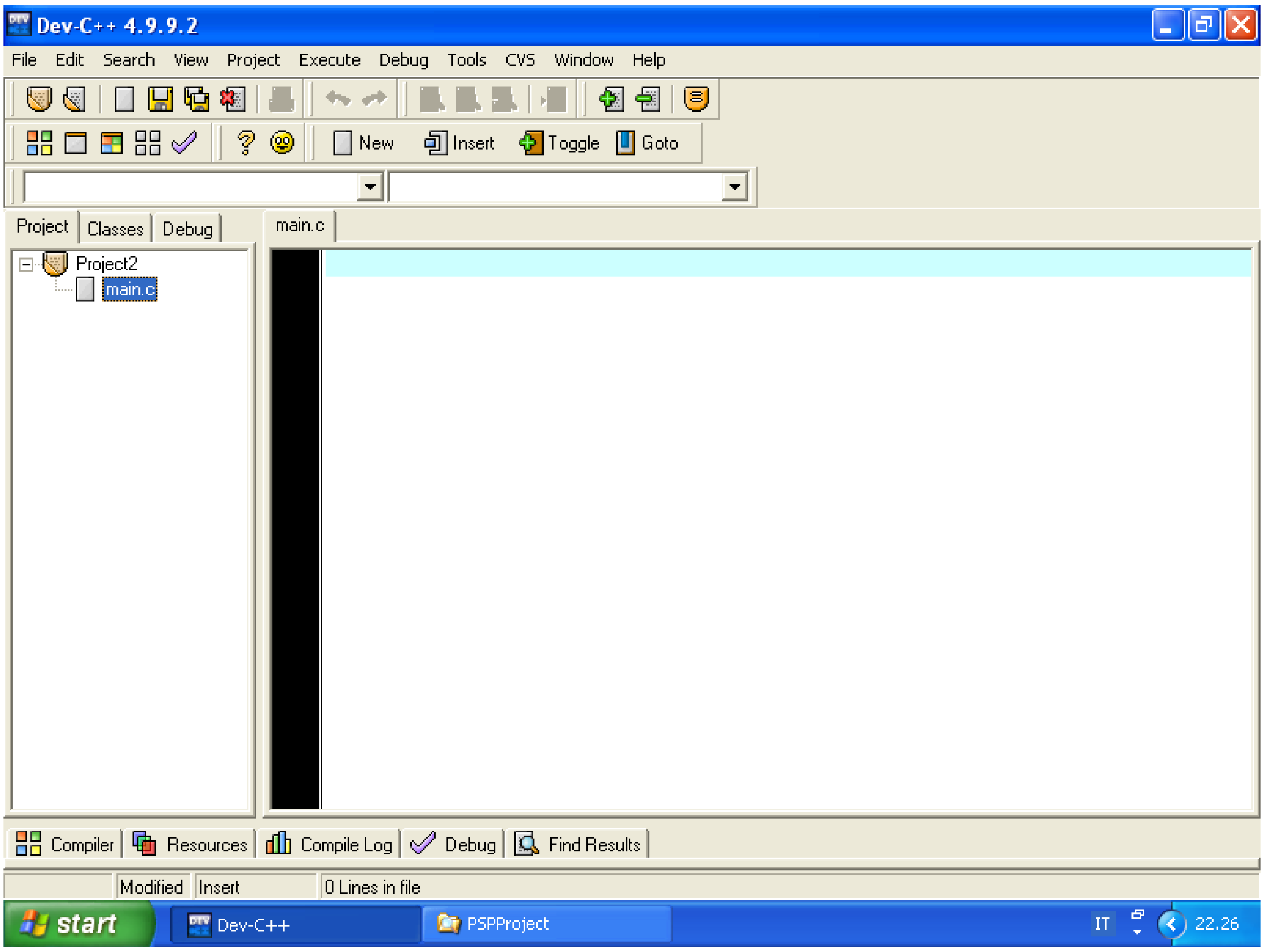
C Project [✓ C++ Project]

Make Default Language

Choose *Empty Project:*


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
Project2
Compiler Resources Compile Log Debug Find Results
Modified Insert Ready.
start Dev-C++ PSPProject IT 22.24

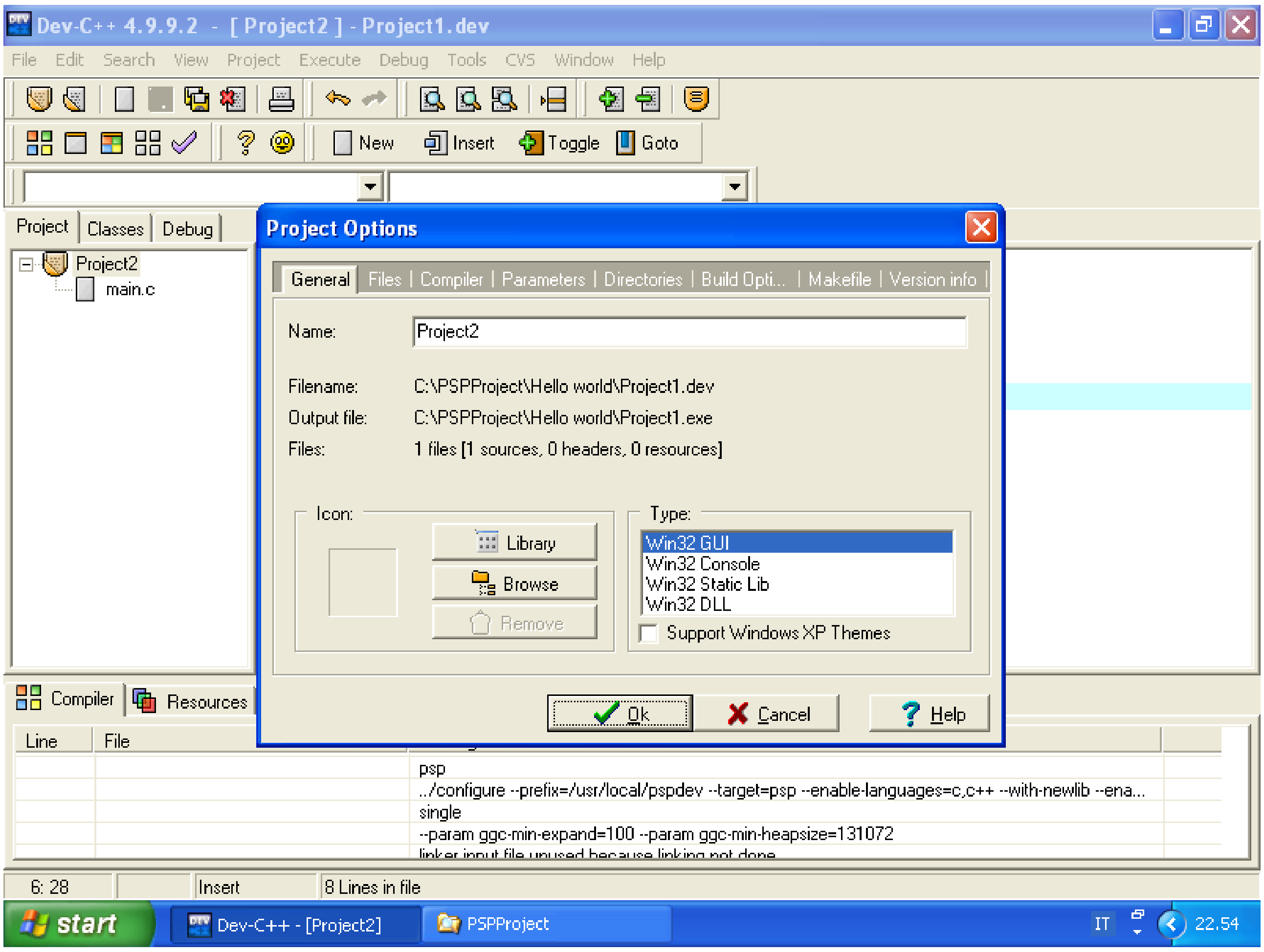
*Ok, now add a new C file to your project. Call it **main.c***


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
main.c
Project2
main.c
Compiler Resources Compile Log Debug Find Results
Modified Insert 0 Lines in file
start Dev-C++ PSPProject
IT 22.26

You can insert your first program:

```
#include <nanodesktop.h>

int ndMain ()
{
        ndInitSystem ();
        printf ("Hello world \n");
}
```

Now save your project and prepare to build it. Go to Project and select *Options:*



Dev-C++ 4.9.9.2 - [Project2] - Project1.dev
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
Project2
main.c
Project Options
General Files Compiler Parameters Directories Build Opti... Makefile Version info
Icon:
Library
Browse
Remove
Ok
Cancel
Help
Compiler Resources
Line File
Insert
start
Dev-C++ - [Project2]
PSPProject
IT
22.54

Name: Project2

Filename: C:\PSPProject\Hello world\Project1.dev

Output file: C:\PSPProject\Hello world\Project1.exe

Files: 1 files [1 sources, 0 headers, 0 resources]

Type:
[✓ Win32 GUI]
Win32 Console
Win32 Static Lib
Win32 DLL

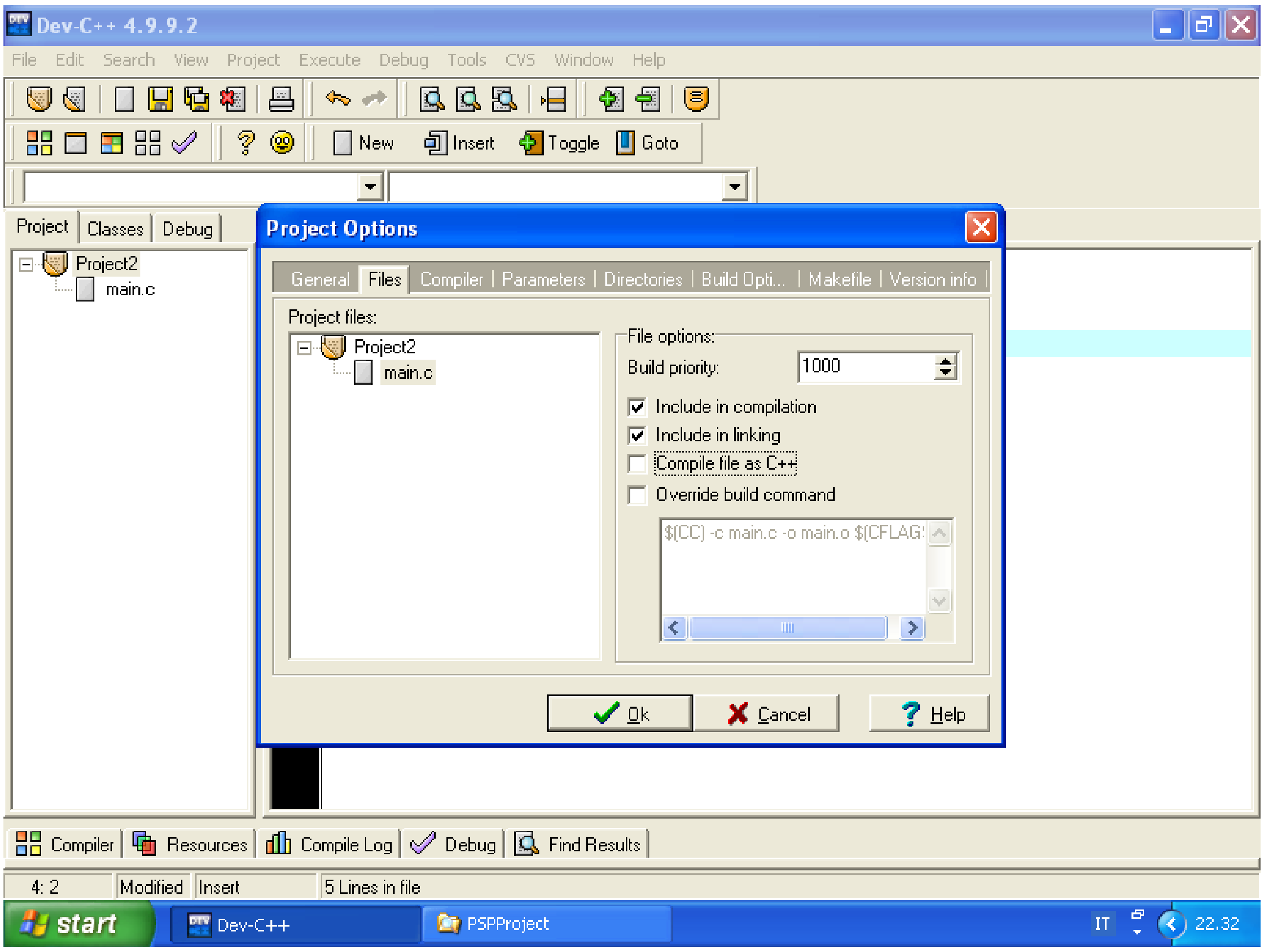
☐ Support Windows XP Themes

```
psp
../configure --prefix=/usr/local/pspdev --target=psp --enable-languages=c,c++ --with-newlib --ena...
single
--param ggc-min-expand=100 --param ggc-min-heapsize=131072
linker input file unused because linking not done
```

6: 28 | Insert | 8 Lines in file

BE CAREFUL:

'Type' MUST BE Win32 GUI. If you choose a different type, like Win32 Console, Dev-C++ will pass to psp-gcc the parameter –windows and you'll obtain an error during compilation.


Project files: Project2, main.c

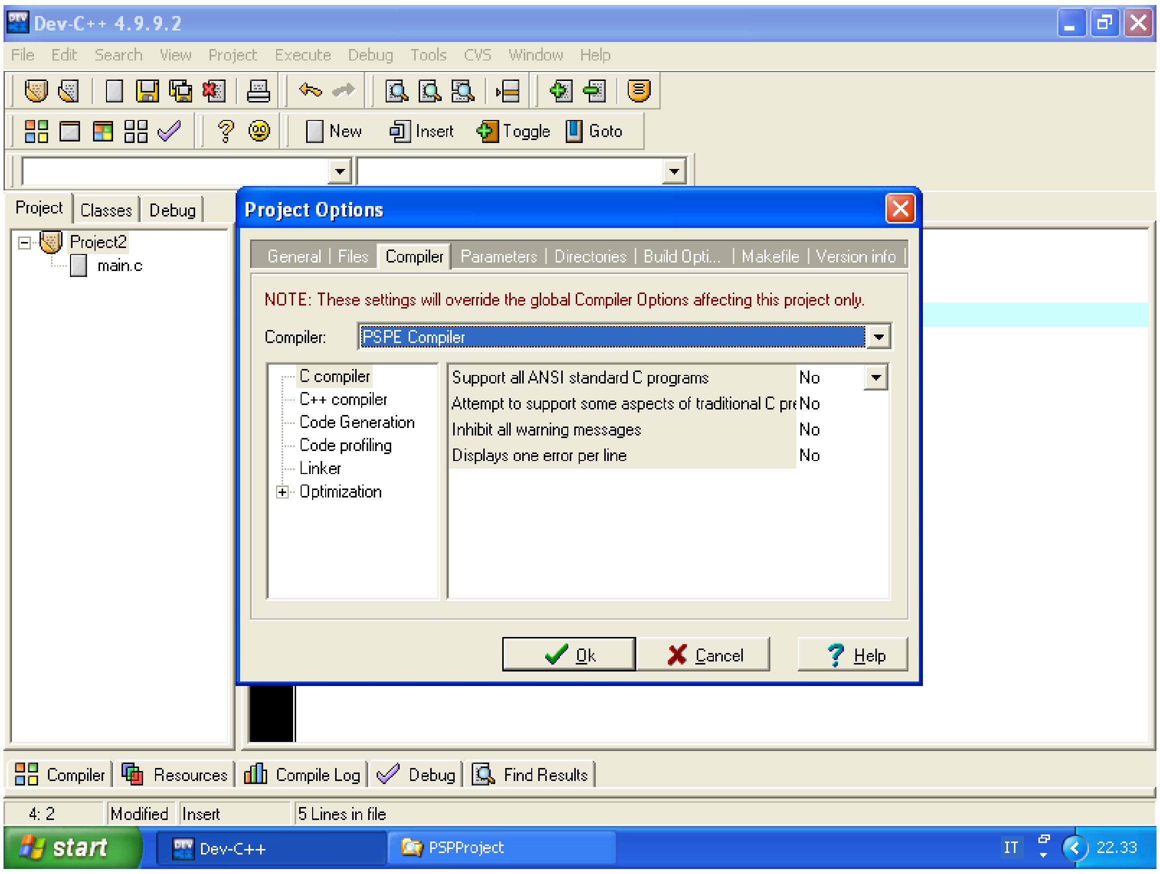
File options:

Build priority: 1000

- [✓] Include in compilation
- [✓] Include in linking
- [ ] Compile file as C++
- [ ] Override build command

```
$(CC) -c main.c -o main.o $(CFLAG
```

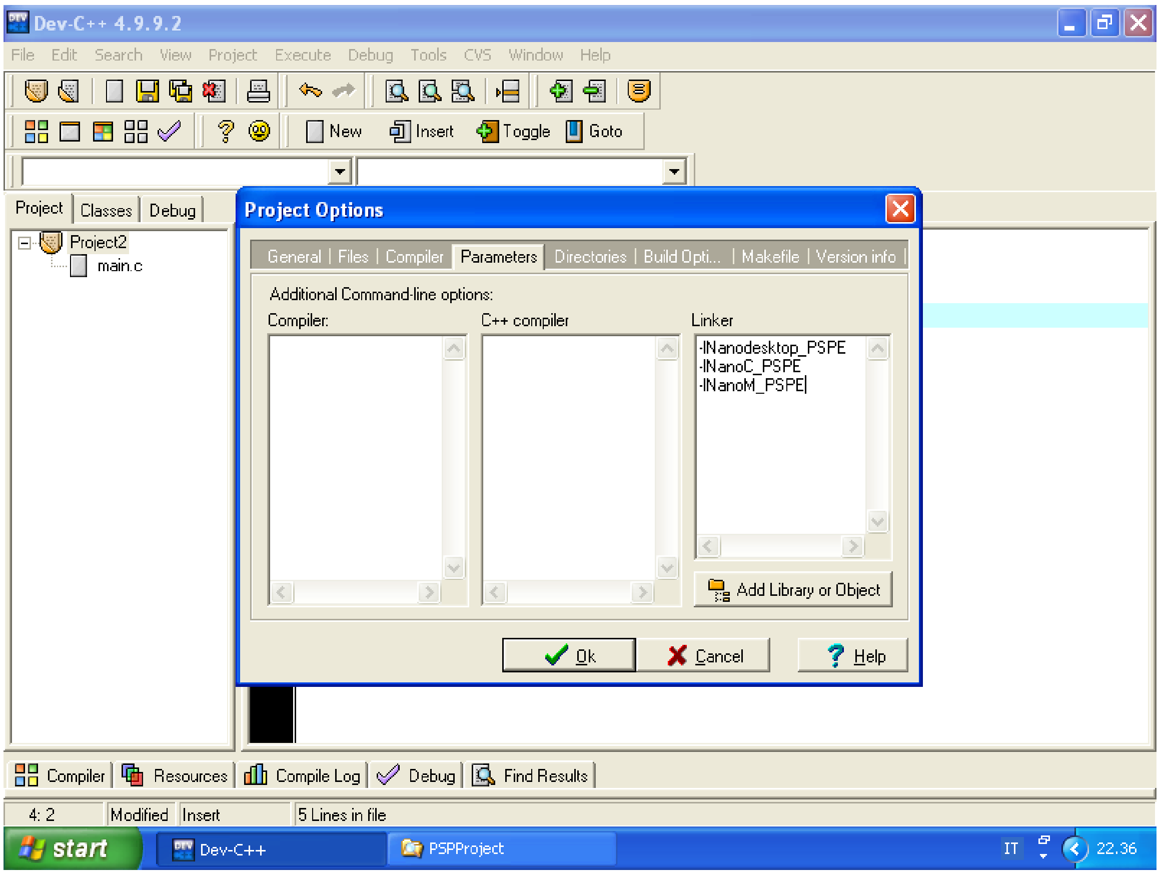
Now choose the PSPE Compiler set:


NOTE: These settings will override the global Compiler Options affecting this project only.

Compiler: PSPE Compiler

Support all ANSI standard C programs	No
Attempt to support some aspects of traditional C pre	No
Inhibit all warning messages	No
Displays one error per line	No

Now, choose the required libraries. For now, you can choose **–lNanodesktop_PSPE**, **-lNanoC_PSPE**, **-lNanoM_PSPE** (-l means -ELLE)


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
Project2
main.c
Project Options
General Files Compiler Parameters Directories Build Opti... Makefile Version info
Add Library or Object
Ok Cancel Help
Compiler Resources Compile Log Debug Find Results
4: 2 Modified Insert 5 Lines in file
start Dev-C++ PSPProject IT 22.36

Additional Command-line options:

Compiler:

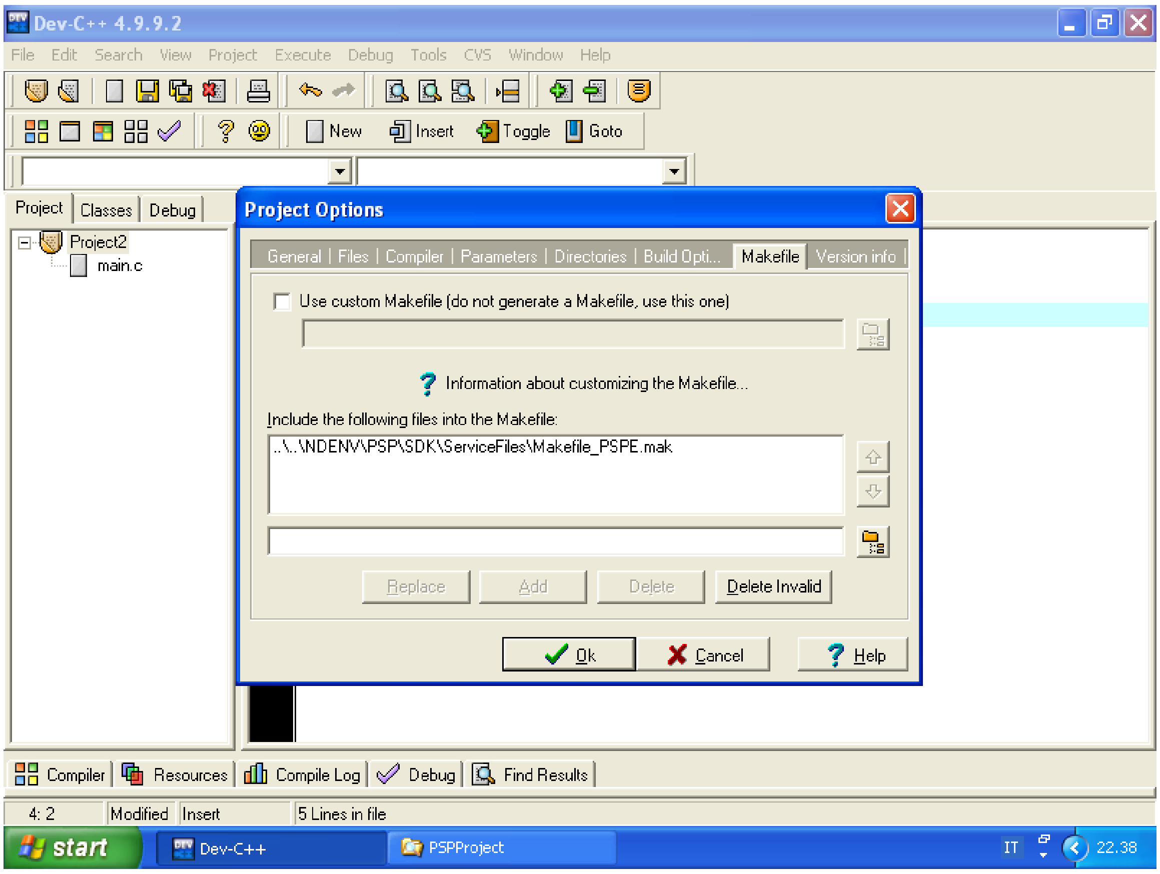
C++ compiler

Linker

```
-lNanodesktop_PSPE
-lNanoC_PSPE
-lNanoM_PSPE
```

Now this is the most important stage: you must tell Dev-CPP to use a custom Makefile for PSPE.

This Makefile is in <ndenv folder>\PSP\SDK\ServiceFiles\Makefile_PSPE.mak.


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
Project2
main.c
Project Options
General Files Compiler Parameters Directories Build Opti... Makefile Version info
Information about customizing the Makefile...
Replace Add Delete Delete Invalid
Ok Cancel Help
Compiler Resources Compile Log Debug Find Results
start Dev-C++ PSPProject IT 22.38

☐ Use custom Makefile (do not generate a Makefile, use this one)

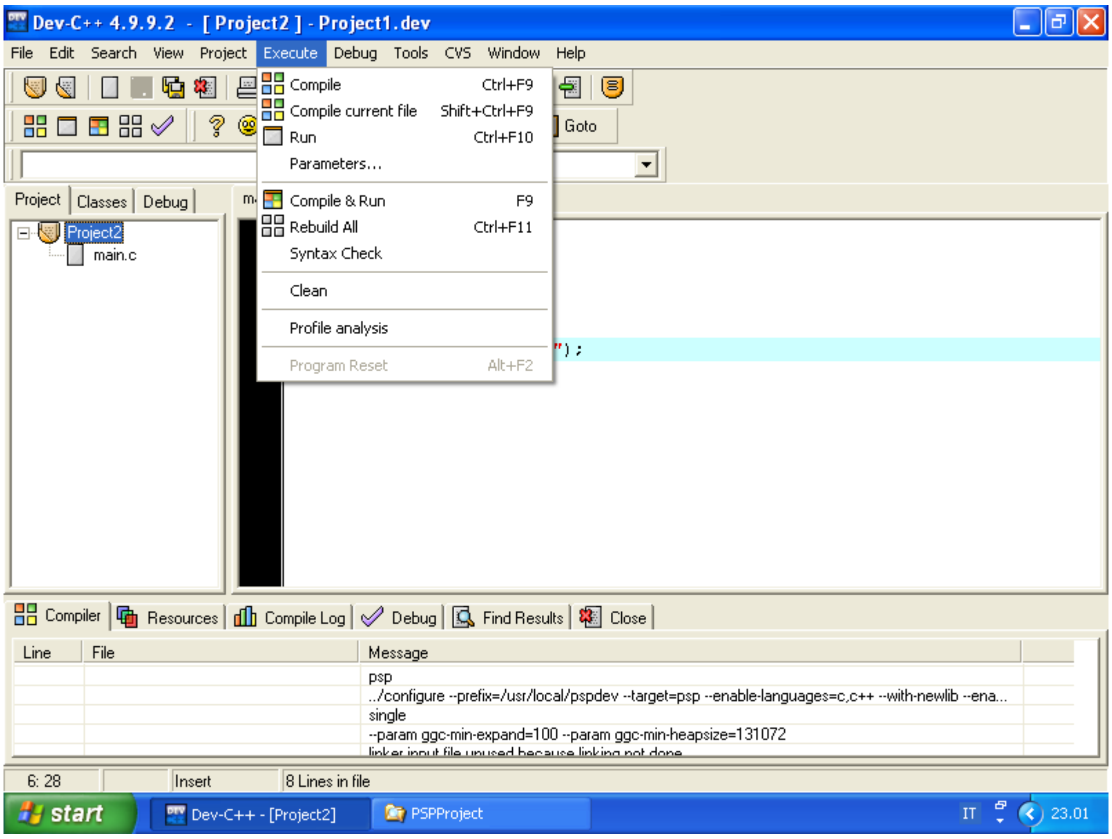
Include the following files into the Makefile:

..\..\NDENV\PSP\SDK\ServiceFiles\Makefile_PSPE.mak

4: 2 Modified Insert 5 Lines in file

This file will recall the PSPE emulator. When you want to recompile for the real Playstation Portable, you will have to change the libraries (**_PSP** instead of **_PSPE**) and the makefile (**Makefile_PSP.mak** instead of **Makefile_PSPE.mak**).

We are now ready to compile. Choose Execute/Rebuid All:

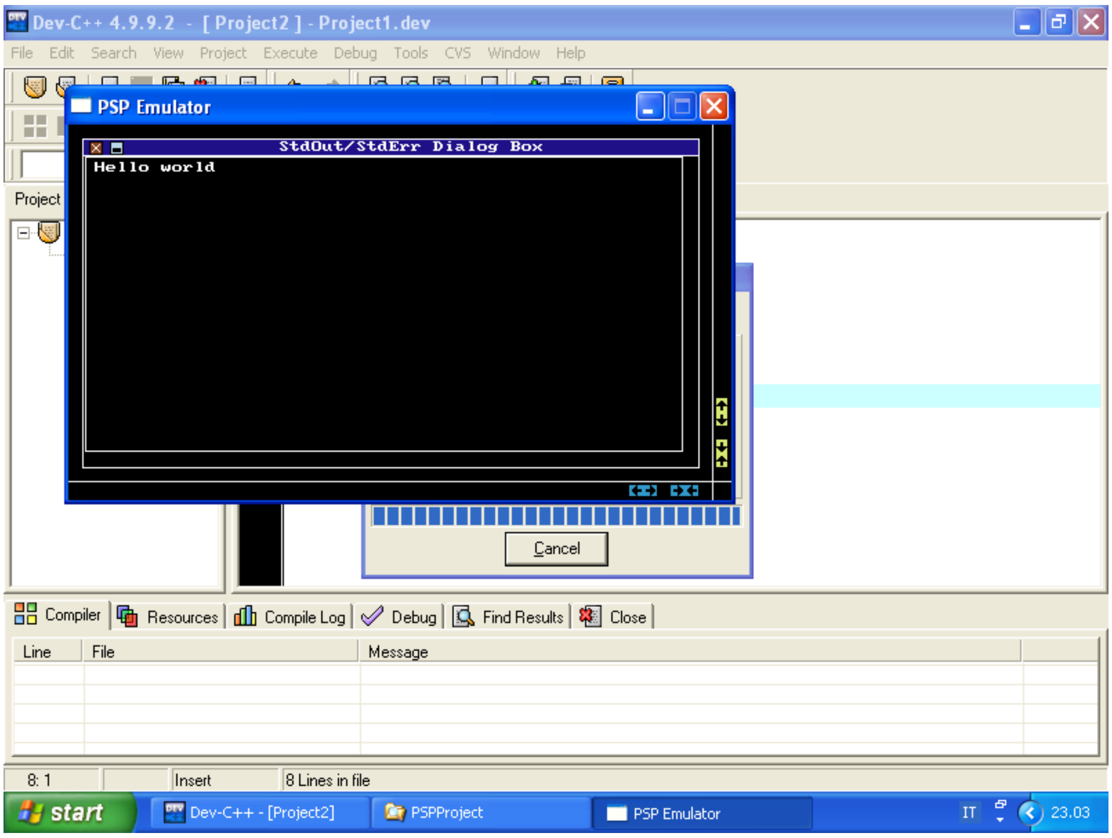

Dev-C++ 4.9.9.2 - [Project2] - Project1.dev
File Edit Search View Project Execute Debug Tools CVS Window Help
Compile Ctrl+F9
Compile current file Shift+Ctrl+F9
Run Ctrl+F10
Parameters...
Compile & Run F9
Rebuild All Ctrl+F11
Syntax Check
Clean
Profile analysis
Program Reset Alt+F2
Goto
Project Classes Debug
Project2
main.c
Compiler Resources Compile Log Debug Find Results Close
start Dev-C++ - [Project2] PSPProject IT 23.01

```
Line  File  Message
            psp
            ../configure --prefix=/usr/local/pspdev --target=psp --enable-languages=c,c++ --with-newlib --ena...
            single
            --param ggc-min-expand=100 --param ggc-min-heapsize=131072
            linker input file unused because linking not done
```

6: 28 Insert 8 Lines in file

Dev-CPP will compile your source.

The system will create a subfolder in the project folder named *pbp_for_pspe:* this will contain the executable EBOOT_PSPE.PBP. The system will also be able to run PSPE automatically:



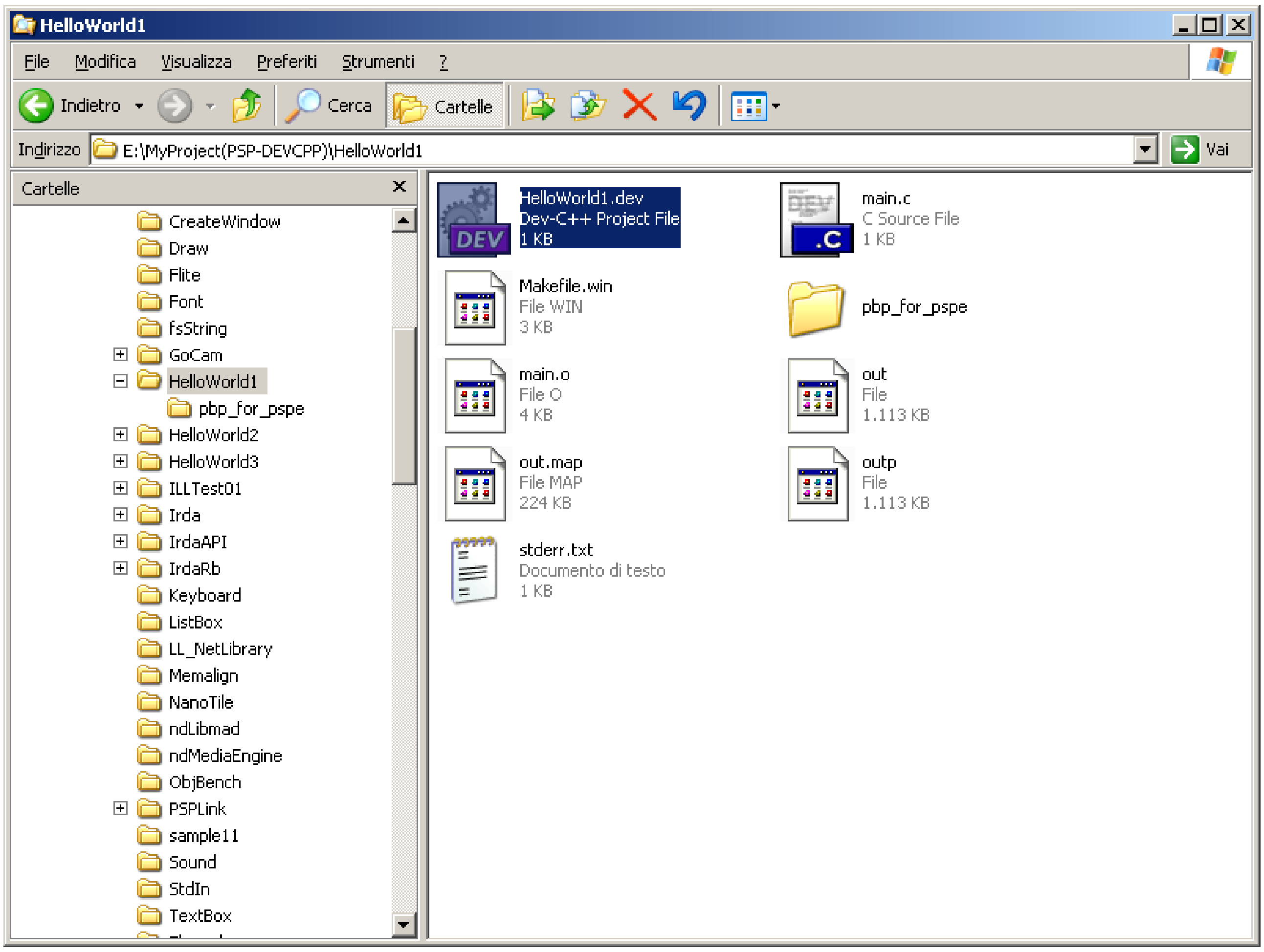
```
Hello world
```

Chapter 4

Recompile your program to work with PSP

In the previous chapter we learned how to create a simple Hello world for PSPE. Now we want to recompile it to work with the Sony PSP.

Firstly, go to the folder that you had assigned to your first project:



The compiler has created the file EBOOT_PSPE.PBP. This contains our Hello world for the PSPE emulator. To create the same hello world PBP for the Playstation Portable, we have to modify options in Dev-CPP Project.

There are three ways to create a nd application for a real Sony Playstation Portable.

a) to create a *simple PSP application.* These applications are very simple, they can work on a PSP FAT or on a PSP SLIM, but they cannot access the most advanced features of the console. These applications cannot access media support, webcam support, irda/raw-ir support or network functions. So you would have to create a simple PSP application only when you want to create an executable that uses graphical functionalities only.

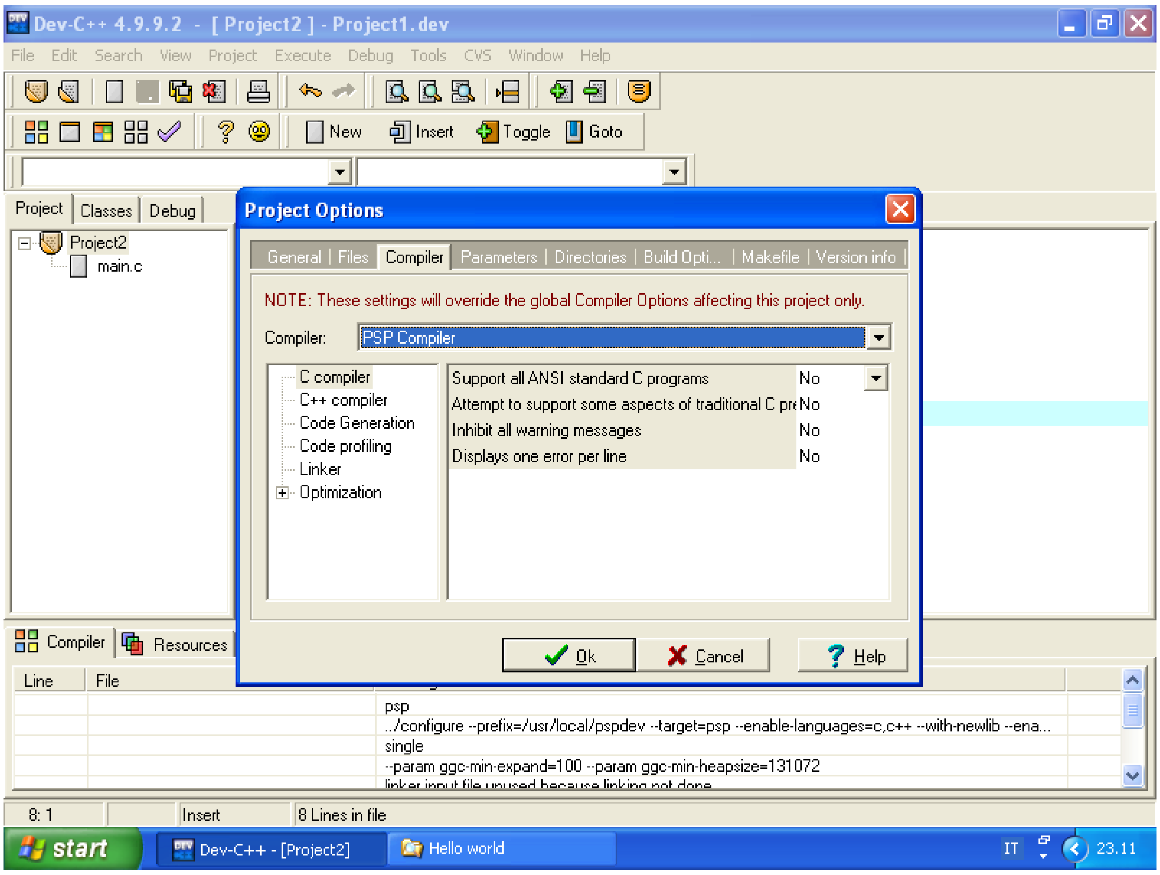
b) to create a *KSU application*. KSU stands for *Kernel Services to User Main.* This mode is able to access to all advanced functions of the system (raw-ir, irda, network, sound, media support, etc.) without the programmer having to manage the interactions between PSP kernel mode and PSP user mode. KSU applications can work only on a PSP FAT with firmware Sony 1.50 (official). The execution of a KSU application on different firmwares (such as those above 1.50) isn't guaranteed and it can cause the system to hang.

c) to create a *CFW application.* CFW stands for *custom firmware.* This mode is able to provide access to all advanced functions of the system (raw-ir, irda, network, sound, media support, etc.) using the custom firmware. This type of applications can work either on a PSP FAT or on a PSP-SLIM. On a PSP-SLIM, nd will recognize automatically the 64 Mb of ram and will use it automatically. Nanodesktop 0.3.4 has been tested on the custom firmware 3.71 M33-4, but the applications can work also on the newer versions of the PSP custom firmwares.

A) To create a simple PSP application

In order to create a simple PSP application, you must do the following:

a) change *compilation profile* from “PSPE Compiler” to “PSP Compiler”:


Dev-C++ 4.9.9.2 - [Project2] - Project1.dev
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
Project2
main.c
Project Options
General Files Compiler Parameters Directories Build Opti... Makefile Version info
C compiler
C++ compiler
Code Generation
Code profiling
Linker
Optimization
Ok Cancel Help
Compiler Resources
Line File
8: 1 Insert 8 Lines in file
start Dev-C++ - [Project2] Hello world IT 23.11

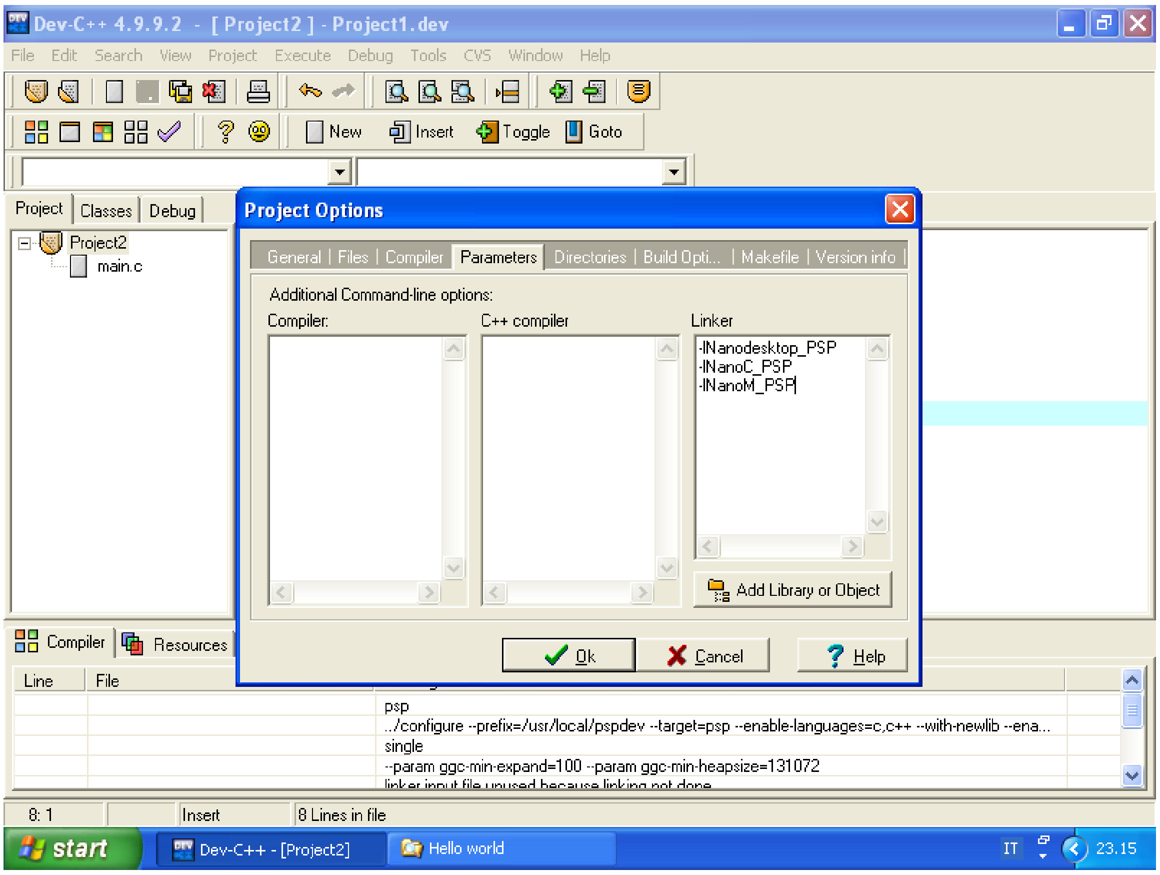
NOTE: These settings will override the global Compiler Options affecting this project only.

Compiler: PSP Compiler

Support all ANSI standard C programs	No
Attempt to support some aspects of traditional C pre	No
Inhibit all warning messages	No
Displays one error per line	No

```
psp
../configure --prefix=/usr/local/pspdev --target=psp --enable-languages=c,c++ --with-newlib --ena...
single
--param ggc-min-expand=100 --param ggc-min-heapsize=131072
linker input file unused because linking not done
```

b) Change all libraries from _PSPE to _PSP version:

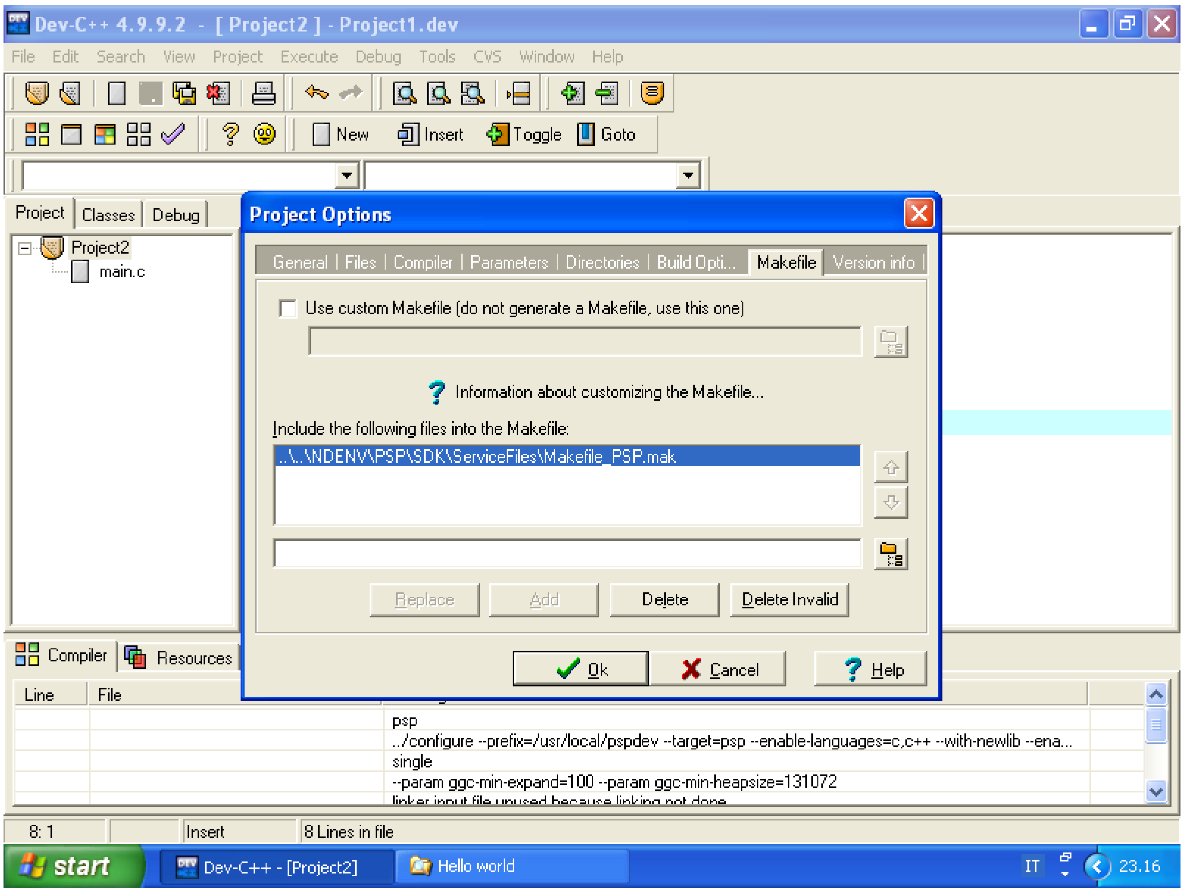

Additional Command-line options:

Linker:

```
-lNanodesktop_PSP
-lNanoC_PSP
-lNanoM_PSP
```

```
psp
../configure --prefix=/usr/local/pspdev --target=psp --enable-languages=c,c++ --with-newlib --ena...
single
--param ggc-min-expand=100 --param ggc-min-heapsize=131072
linker input file unused because linking not done
```

c) Change makefile from _PSPE to _PSP version:


Use custom Makefile (do not generate a Makefile, use this one)

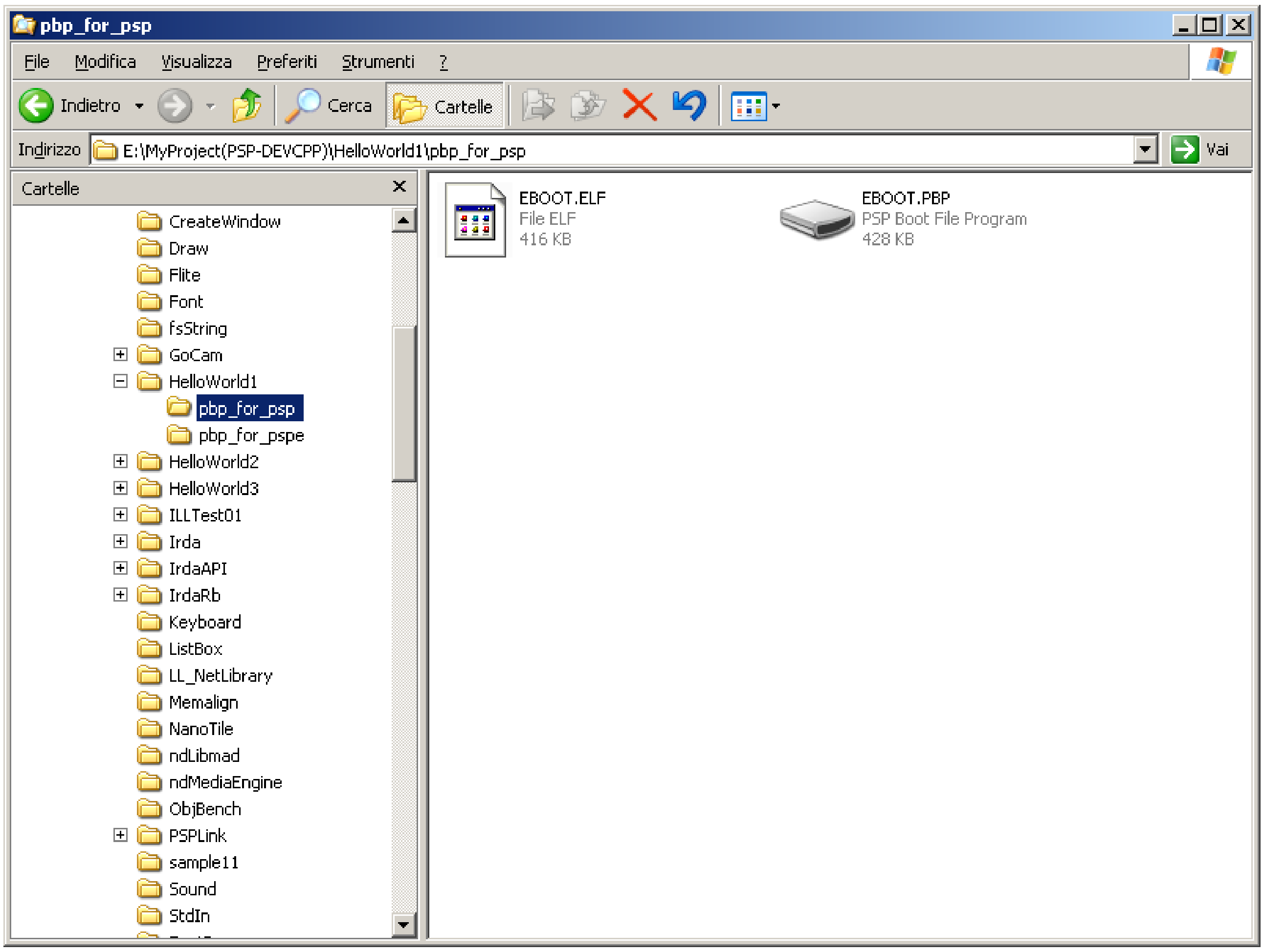
Include the following files into the Makefile:

```
..\..\NDENV\PSP\SDK\ServiceFiles\Makefile_PSP.mak
```

```
psp
../configure --prefix=/usr/local/pspdev --target=psp --enable-languages=c,c++ --with-newlib --ena...
single
--param ggc-min-expand=100 --param ggc-min-heapsize=131072
linker input file unused because linking not done
```

Now you can press “OK” and after that you can press “Rebuild all”.

This time, the IDE doesn’t start the emulator, but only creates a subfolder *pbp_for_psp,* that contains a file called EBOOT.PBP.



Indirizzo E:\MyProject(PSP-DEVCPP)\HelloWorld1\pbp_for_psp

EBOOT.ELF
File ELF
416 KB

EBOOT.PBP
PSP Boot File Program
428 KB

This application can be installed on Sony Memory Stick for the execution. There are two ways to do this:

a) if your PSP uses Sony PSP firmware 1.5, you can use the utility *PSPBrew* for Windows. This utility is able to use the *kxploit* method to install user code on Sony PSP memory stick.

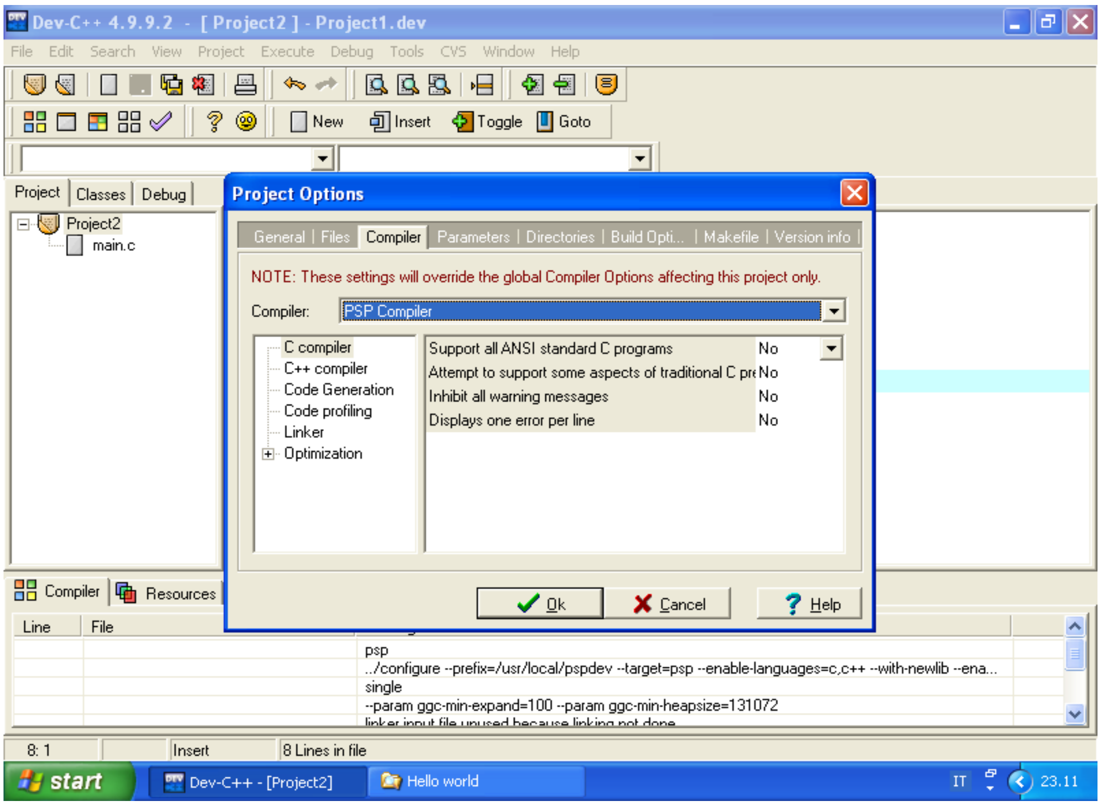
b) if your PSP uses a custom firmware, you can simply use the *copy and paste* method to install the new application. Create a new folder for your program in PSP/GAME/ folder on the memory stick and copy the binary to this new folder.

In either cases, Sony interface will recognize the new program and will execute it.

B) To create a KSU PSP application

KSU applications can work only on Sony PSP firmware version 1.5.
In order to create a KSU PSP application, you must do this:

a) change *compilation profile* from “PSPE Compiler” to “PSP Compiler”:


Dev-C++ 4.9.9.2 - [Project2] - Project1.dev
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
Project2
main.c
Project Options
General Files Compiler Parameters Directories Build Opti... Makefile Version info
C compiler
C++ compiler
Code Generation
Code profiling
Linker
Optimization
Ok Cancel Help
Compiler Resources
Line File
start Dev-C++ - [Project2] Hello world
IT 23.11

NOTE: These settings will override the global Compiler Options affecting this project only.

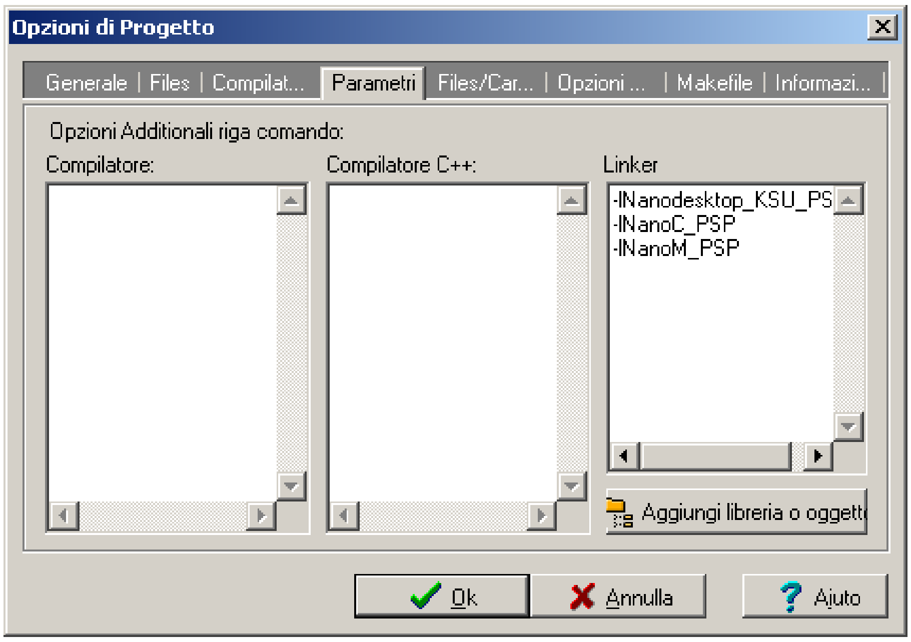
Compiler: PSP Compiler

Support all ANSI standard C programs	No
Attempt to support some aspects of traditional C pre	No
Inhibit all warning messages	No
Displays one error per line	No

```
psp
../configure --prefix=/usr/local/pspdev --target=psp --enable-languages=c,c++ --with-newlib --ena...
single
--param ggc-min-expand=100 --param ggc-min-heapsize=131072
linker input file unused because linking not done
```

8: 1 Insert 8 Lines in file

b) Change all libraries from PSPE to PSP version. Furthermore, you must choose the version of Nanodesktop library that is dedicated to KSU mode (**Nanodesktop_KSU_PSP**).


Opzioni di Progetto
Generale Files Compilat... Parametri Files/Car... Opzioni ... Makefile Informazi...
Aggiungi libreria o oggetti
Ok Annulla Aiuto

Opzioni Additionali riga comando:

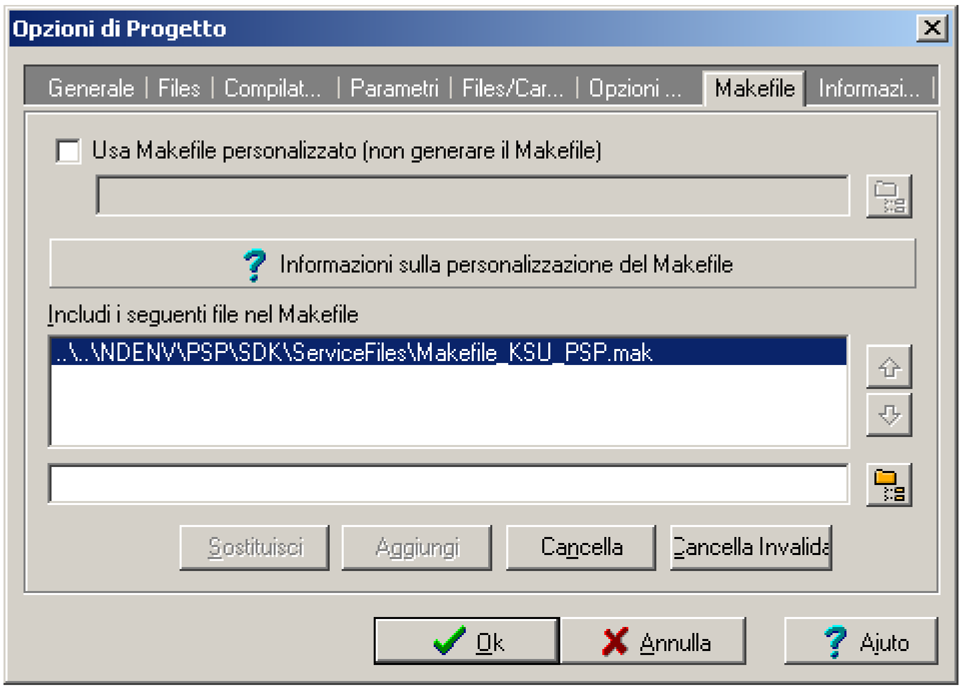
Compilatore:

Compilatore C++:

Linker

```
-lNanodesktop_KSU_PS
-lNanoC_PSP
-lNanoM_PSP
```

c) Change makefile to the following version **Makefile_KSU_PSP.mak**:


Opzioni di Progetto
Generale | Files | Compilat... | Parametri | Files/Car... | Opzioni ... | Makefile | Informazi...
Sostituisci
Aggiungi
Cancella
Cancella Invalidi
Ok
Annulla
Aiuto

Usa Makefile personalizzato (non generare il Makefile)

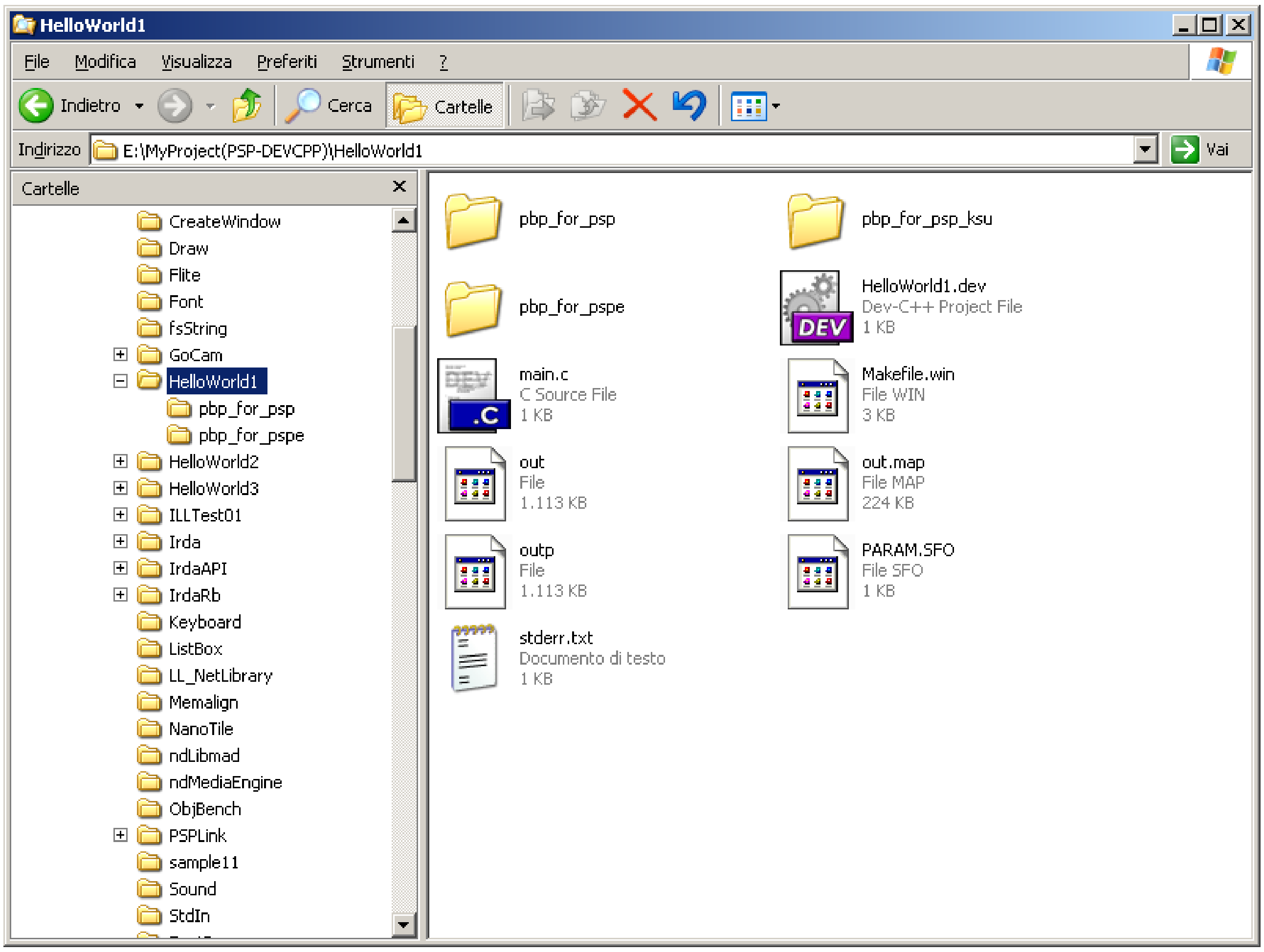
Informazioni sulla personalizzazione del Makefile

Includi i seguenti file nel Makefile

..\..\NDENV\PSP\SDK\ServiceFiles\Makefile_KSU_PSP.mak

Now you can press on “OK” and after that you can make “Rebuild all”.

The IDE will provide to create a new subfolder called *pbp_for_ksu_psp,* that will contain a file called EBOOT.PBP.



E:\MyProject(PSP-DEVCPP)\HelloWorld1

CreateWindow
Draw
Flite
Font
fsString
GoCam
HelloWorld1
pbp_for_psp
pbp_for_pspe
HelloWorld2
HelloWorld3
ILLTest01
Irda
IrdaAPI
IrdaRb
Keyboard
ListBox
LL_NetLibrary
Memalign
NanoTile
ndLibmad
ndMediaEngine
ObjBench
PSPLink
sample11
Sound
StdIn

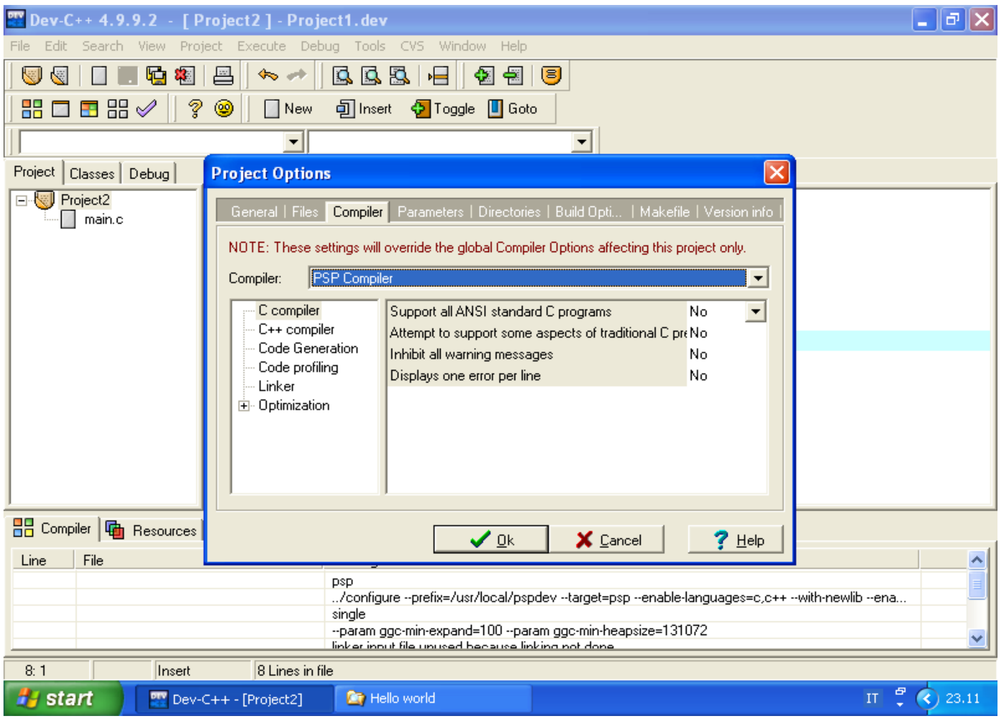
pbp_for_psp
pbp_for_psp_ksu
pbp_for_pspe
HelloWorld1.dev Dev-C++ Project File 1 KB
main.c C Source File 1 KB
Makefile.win File WIN 3 KB
out File 1.113 KB
out.map File MAP 224 KB
outp File 1.113 KB
PARAM.SFO File SFO 1 KB
stderr.txt Documento di testo 1 KB

Now, you can use PSP-Brew utility to install the new application on the Sony PSP with firmware version 1.5.

C) To create a CFW PSP application

If you want to create a custom firmware PSP application, you must do this:

a) change *compilation profile* from “PSPE Compiler” to “PSP Compiler”:



Dev-C++ 4.9.9.2 - [Project2] - Project1.dev
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
Project2
main.c
Project Options
General Files Compiler Parameters Directories Build Opti... Makefile Version info
Ok Cancel Help
Compiler Resources
Line File
start Dev-C++ - [Project2] Hello world
IT 23.11

NOTE: These settings will override the global Compiler Options affecting this project only.

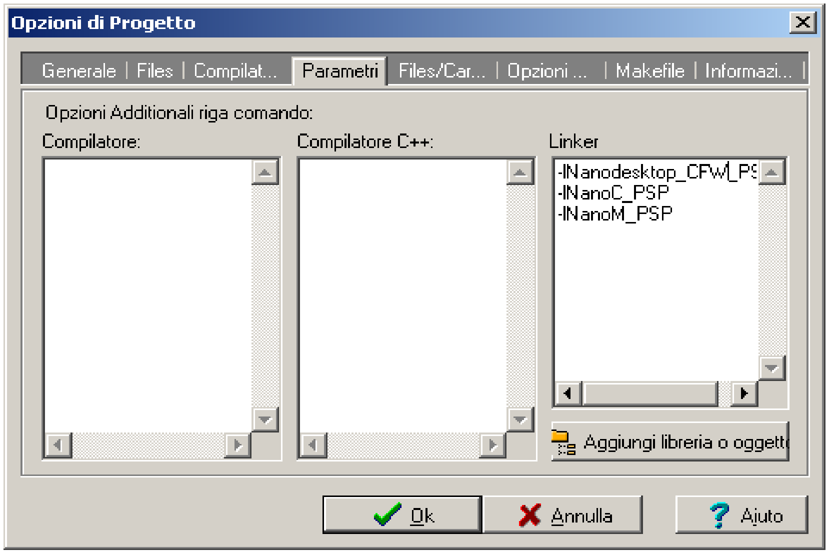
Compiler: PSP Compiler

C compiler	Support all ANSI standard C programs	No
C++ compiler	Attempt to support some aspects of traditional C pre	No
Code Generation	Inhibit all warning messages	No
Code profiling	Displays one error per line	No
Linker		
Optimization		

```
psp
../configure --prefix=/usr/local/pspdev --target=psp --enable-languages=c,c++ --with-newlib --ena...
single
--param ggc-min-expand=100 --param ggc-min-heapsize=131072
linker input file unused because linking not done
```

8: 1 Insert 8 Lines in file

b) Change all libraries from PSPE to PSP version. Furthermore, you must choose the version of Nanodesktop library that is dedicated to CFW mode (**Nanodesktop_CFW_PSP**).



Opzioni di Progetto
Generale Files Compilat... Parametri Files/Car... Opzioni ... Makefile Informazi...
Aggiungi libreria o oggett
Ok Annulla Aiuto

Opzioni Additionali riga comando:

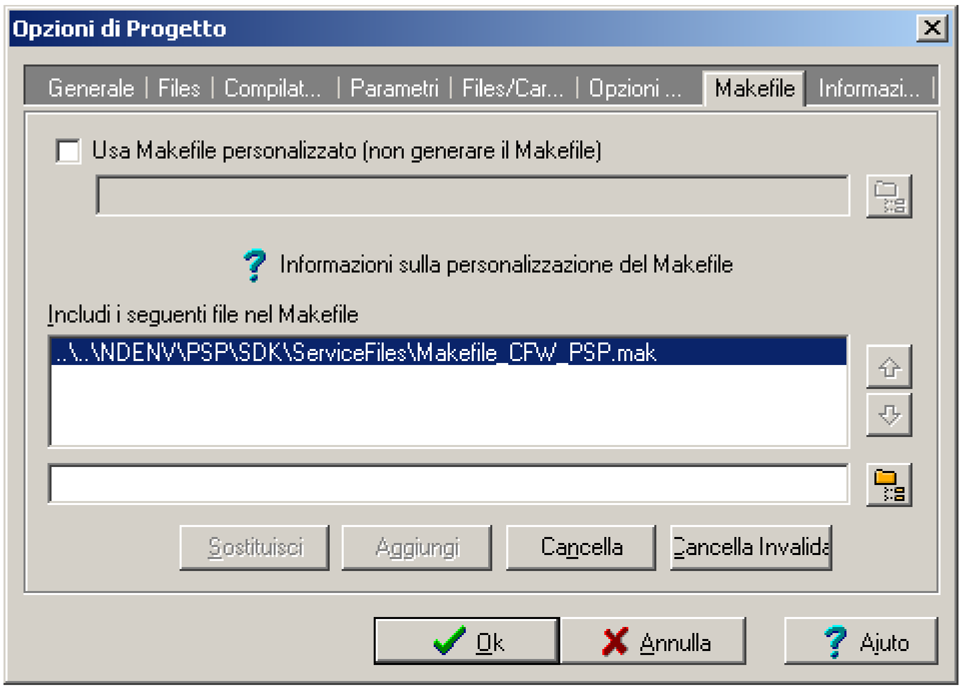
Compilatore:

Compilatore C++:

Linker

```
-lNanodesktop_CFW_PS
-lNanoC_PSP
-lNanoM_PSP
```

c) Change makefile to the following version **Makefile_CFW_PSP.mak**:


Opzioni di Progetto
Generale | Files | Compilat... | Parametri | Files/Car... | Opzioni ... | Makefile | Informazi...

☐ Usa Makefile personalizzato (non generare il Makefile)

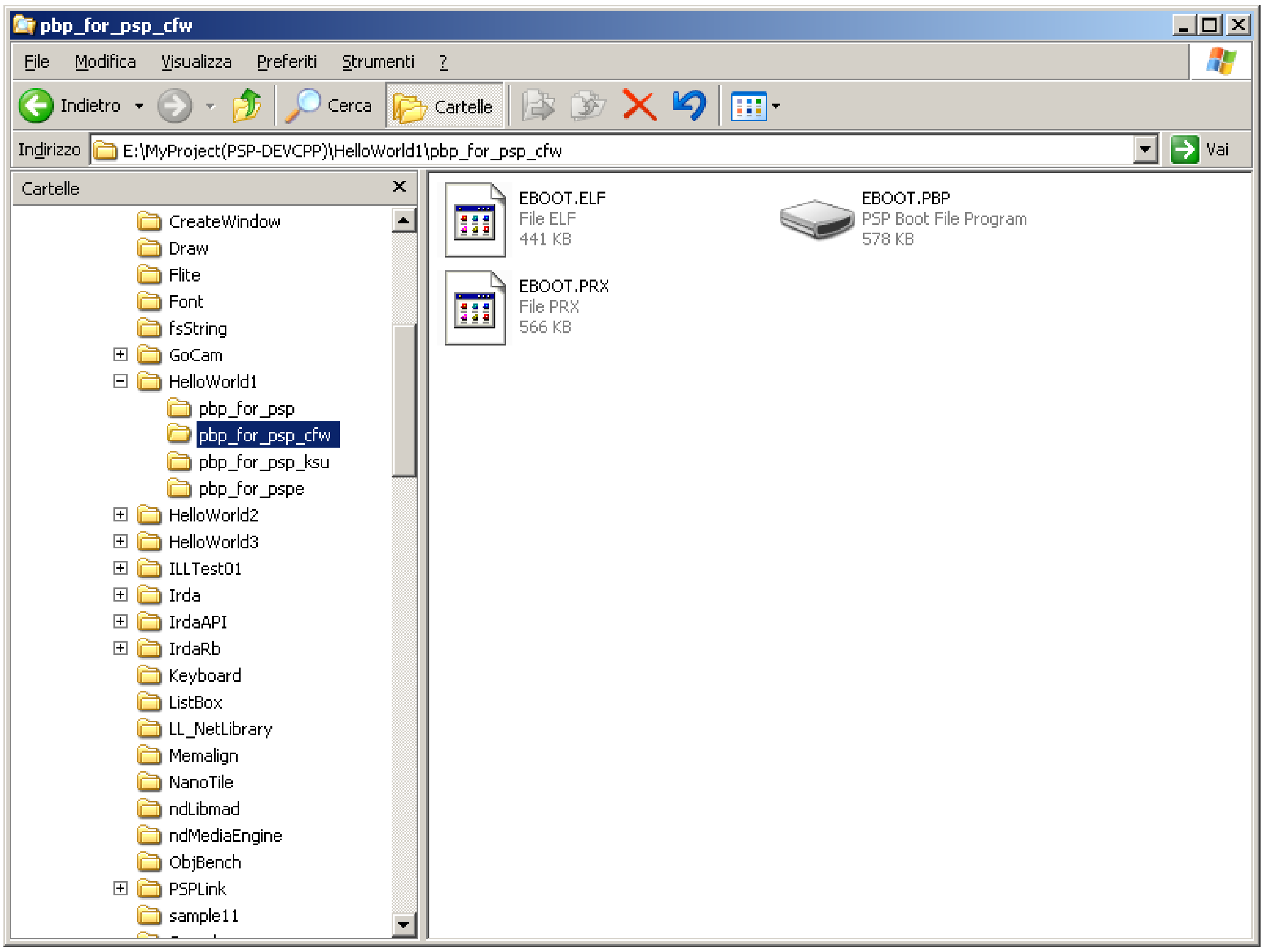
Informazioni sulla personalizzazione del Makefile

Includi i seguenti file nel Makefile

```
..\..\NDENV\PSP\SDK\ServiceFiles\Makefile_CFW_PSP.mak
```

Sostituisci
Aggiungi
Cancella
Cancella Invalida
Ok
Annulla
Aiuto

Choose *Rebuild All*. The system will create a subfolder called *pbp_for_psp_cfw*, that will contain the executable EBOOT.PBP.



E:\MyProject(PSP-DEVCPP)\HelloWorld1\pbp_for_psp_cfw

EBOOT.ELF
File ELF
441 KB

EBOOT.PBP
PSP Boot File Program
578 KB

EBOOT.PRX
File PRX
566 KB

Be careful: Dev-C++ could signal a message like this:

warning: cannot find entry symbol module_start; defaulting to 0000000000000000

You can simply ignore it.

The executable EBOOT.PBP can simply be copied in a subfolder of PSP\GAME folder on the memory stick (use *copy and paste method).*

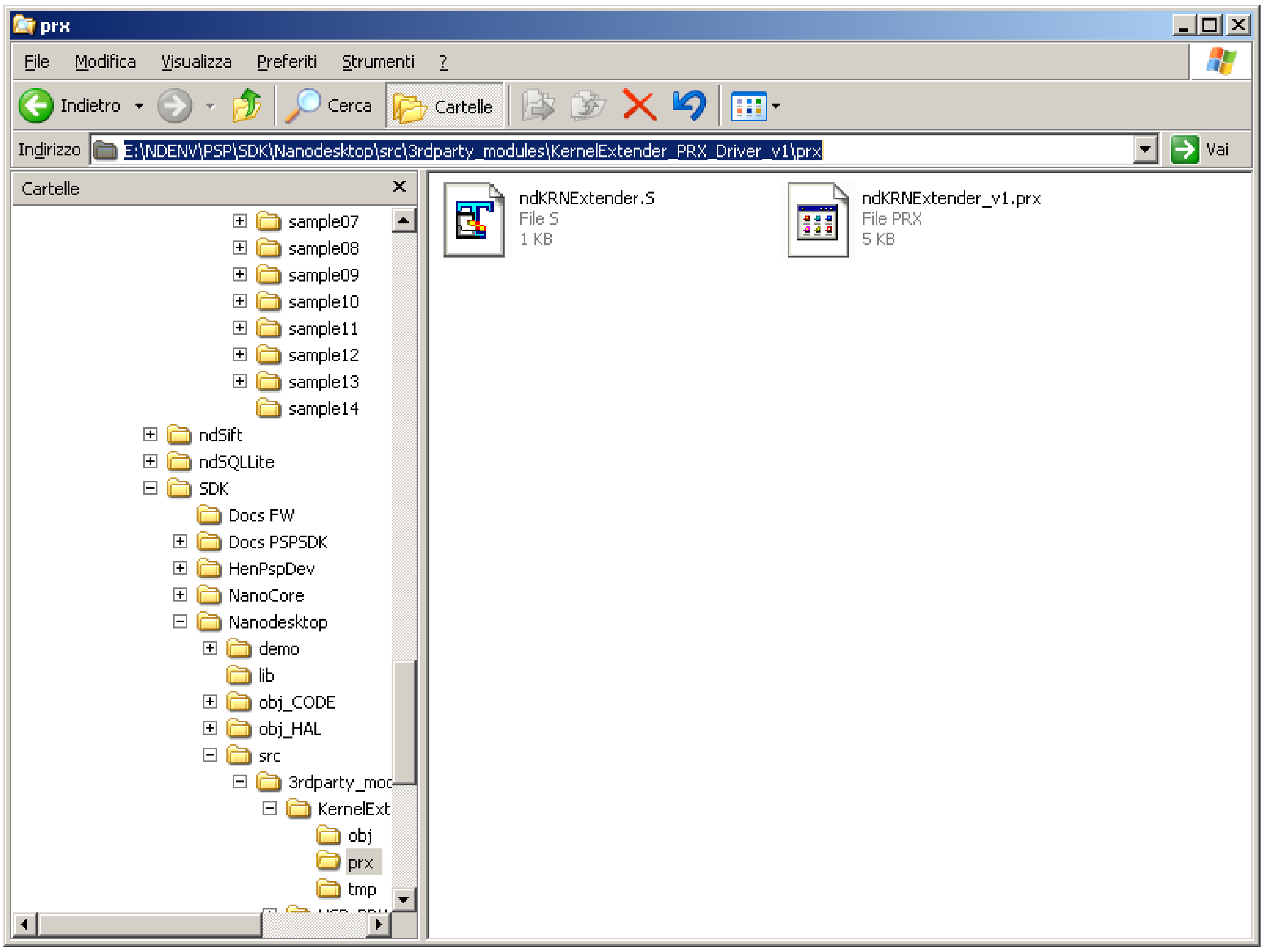
The kernel extender

Be careful to this: *all Nanodesktop applications for custom firmware, need a special driver called* ***Kernel Extender,*** *to work correctly. If you want to use your PSP application for custom firmware, you must copy the extender in the root folder of the memory stick first.*

Go to

<ndenv path>\PSP\SDK\Nanodesktop\src\3rdparty_modules\KernelExtender_PRX_Driver_v1\prx

And here you will find the *kernel extender driver:*



Copy the file **ndKRNExtender_v1.prx** on the root folder of the Memory Stick. The driver is the same for all applications for custom firmware.

Now, you can execute all CFW applications that you want.

Chapter 5
Where you can find help

If you have any problems with the Nanodesktop installation, or during its use, you can find support
and other information at Nanodesktop's official website:

http://visilab.unime.it/~filippo/

or at the Nanodesktop official support forum:

http://www.psp-ita.com/forum

There is also a manual (Nanodesktop user guide) that illustrates Nanodesktop features with examples and images.

You can download it from the Nanodesktop website:

http://visilab.unime.it/~filippo/Nanodesktop\PSP_PSPE\Docs\Docs.htm

This manual is still incomplete. The author updates it continuously and you can find ever more advanced information about the software.

Chapter 7
A second program with Nanodesktop

One of the features of Nanodesktop is the *compatibility with ANSI C.* This means that all standard C programs, can be executed in the Nanodesktop environment, without any changes.

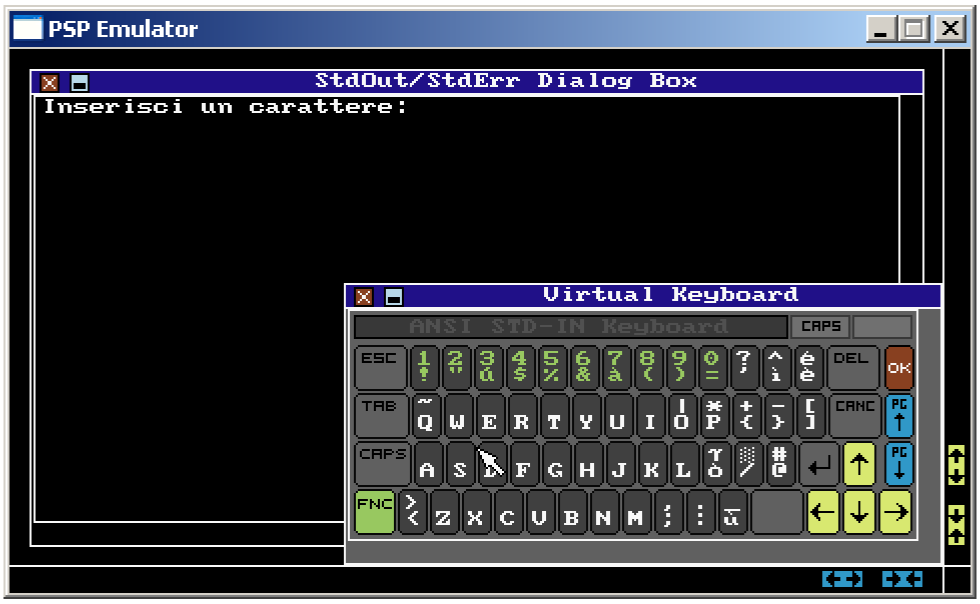
This compatibility is limited to ANSI C programs: you cannot use GTK+ sources or QT sources (obviously), but you can, in any case, write a simple test program in the C language and execute it under PSP.

Try this new program:

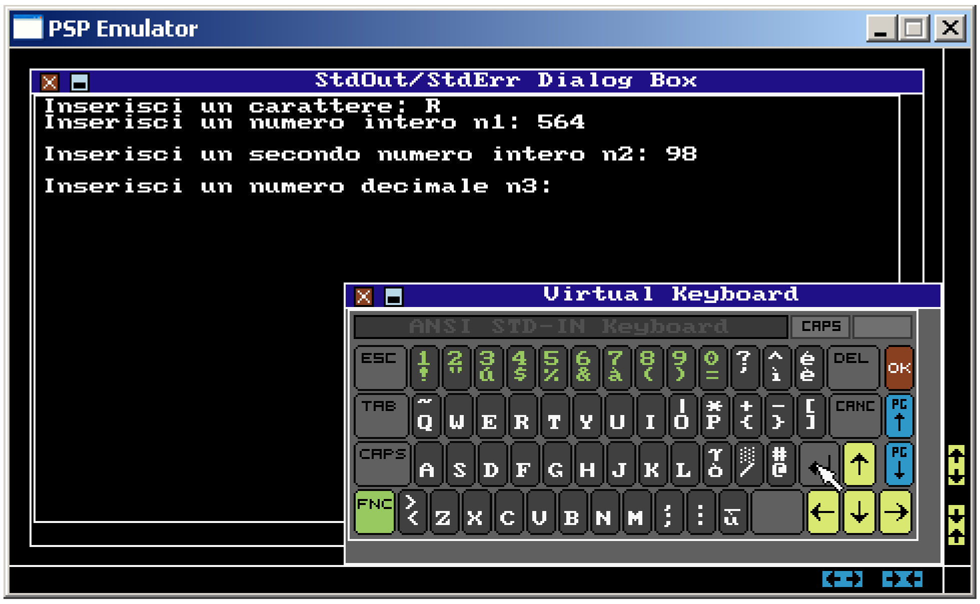
```
#include <nanodesktop.h>
#include <stdio.h>

int ndMain()
{
        char car;
        int n1, n2, totint;
        float f3, totfloat;

        ndInitSystem ();

        printf("Inserisci un carattere: ");
        scanf("%c", &car);
        printf("\nInserisci un numero intero n1: ");
        scanf("%i", &n1);
        printf("\nInserisci un secondo numero intero n2: ");
        scanf("%i", &n2);

        printf("\nInserisci un numero decimale n3: ");
        scanf("%f", &f3);
        printf("\n\n%c%c %c\n%c\n", car, car, car, car);
        totint=n1+n2;
        printf("n1+n2= %d\n\n", totint);
        totint=n1*n2;
        printf("n1*n2= %d\n\n", totint);
        totfloat=n1/n2;
        printf("n1/n2= %f\n\n", totfloat);
        totfloat=(float)n1/(float)n2;
        printf("(float)n1/(float)n2= %f\n\n", totfloat);
        totfloat=(float)n1/n2;
        printf("(float)n1/n2= %f\n\n", totfloat);
        totfloat=n1+f3;
        printf("n1+f3= %f\n\n", totfloat);
        return 0;
}
```

Nanodesktop emulates the keyboard device using its Virtual Keyboard. So when you try to open stdin stream, the system automatically opens the VirtualKeyboard.

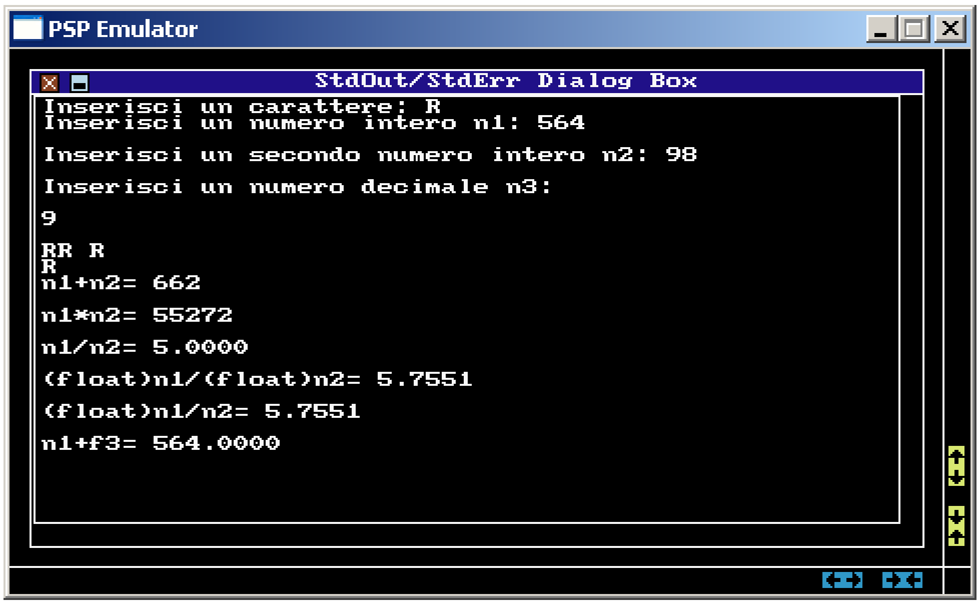
This is the result of the previous code:



```
Inserisci un carattere:
```



```
Inserisci un carattere: R
Inserisci un numero intero n1: 564

Inserisci un secondo numero intero n2: 98

Inserisci un numero decimale n3:
```



```
Inserisci un carattere: R
Inserisci un numero intero n1: 564

Inserisci un secondo numero intero n2: 98

Inserisci un numero decimale n3:

9

RR R
R
n1+n2= 662

n1*n2= 55272

n1/n2= 5.0000

(float)n1/(float)n2= 5.7551

(float)n1/n2= 5.7551

n1+f3= 564.0000
```

As you can see, a standard C program is executed on the PSP.

Chapter 8
The Base Screen (BS)

We come now to a new lesson. Instead of using the C standard libraries, now we will begin to use the abilities of Nanodesktop natively in order to create some simple graphical elements.

Let's try this code:

```
#include <nanodesktop.h>

int ndMain (void)
{
        int CounterX, CounterY;

        ndInitSystem ();

        for (CounterY=0; CounterY<4; CounterY++)
        {
                for (CounterX=0; CounterX<4; CounterX++)
                {
                      ndBS_DrawRectangle (10+40*CounterX, 10+40*CounterY,
                                          10+40*CounterX+30, 10+40*CounterY+30,
                                          COLOR_BLUE, COLOR_YELLOW, RENDER);

                }
         }
}
```

The new routine is **ndBS_DrawRectangle**.

This routine draws a rectangle on the screen. The first 4 parameters are the coordinates, the two following parameters are the inner color and the color of the edge. These two colors can be supplied like 16-bit values (but they can also be obtained using the result of **ndHAL_RGBToMagicNumber** starting from RGB coordinates).

However, in order to facilitate this task for developers, in

<ndenv>/PSP/SDK/Nanodesktop/src/HAL_PSPE/$$Variables.h,

there is a list of the symbolic constants for 16 colors and for 16 tones of grey. In this way, the developers can simply write COLOR_WHITE in place of 0xFFFF.

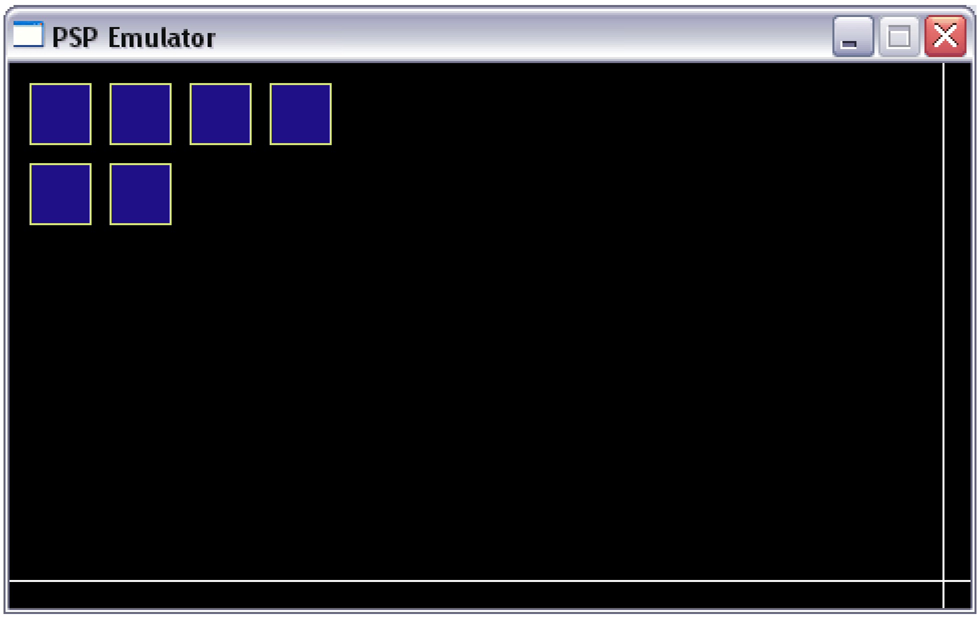
The key word RENDER, used as the last parameter executes immediately the rendering. That makes it immediately appear on the screen all that has been drawn before in the backscreen.

NdBS stands for BaseScreen. The BaseScreen is the screen that is outside of whichever window. If you want to draw something outside of the context of whichever window, you must use the routines ndBS.

You can find these routines, with the parameters and the error codes, in the file ***ndCODE_BaseScreen.c***.

When we compile our code and we make it run on PSPE

we will see the rectangles that are drawn one by one on the screen. Why?



The reason is simple: Nanodesktop draws the graphical elements in a hidden part of the screen and only after the programmer requires the rendering does it execute the transition that makes it all appear instantaneously. The parameter RENDER at the end of the call informs Nanodesktop that after the creation of the rectangle, it is necessary to execute the rendering immediately in a way that the rectangle appears on the screen. That is why the rectangles appear one at a time.

We now want all the rectangles to appear at the same time.

This is possible, but it is necessary to modify the code. We therefore try this:

```
#include <nanodesktop.h>

int ndMain (void)
{
        int CounterX, CounterY;

        ndInitSystem ();

        for (CounterY=0; CounterY<4; CounterY++)
        {
                for (CounterX=0; CounterX<4; CounterX++)
                {
                      ndBS_DrawRectangle (10+40*CounterX, 10+40*CounterY,
                                                     10+40*CounterX+30, 10+40*CounterY+30,
                                                     COLOR_BLUE, COLOR_YELLOW,
                                                     NORENDER};
                 }
        }

        BaseScreenRender();
}
```

As you can see, we have modified the call to ndBS_DrawRectangle so that the last parameter is NORENDER.

In this way, Nanodesktop draws the rectangles in the backscreen, but it does not visualize them on the screen. However, only at the end of the loop does it execute a unique BaseScreenRender that uses the function to visualize instantaneously all elements drawn in the backscreen: this means that the rectangles will appear all at the same time!

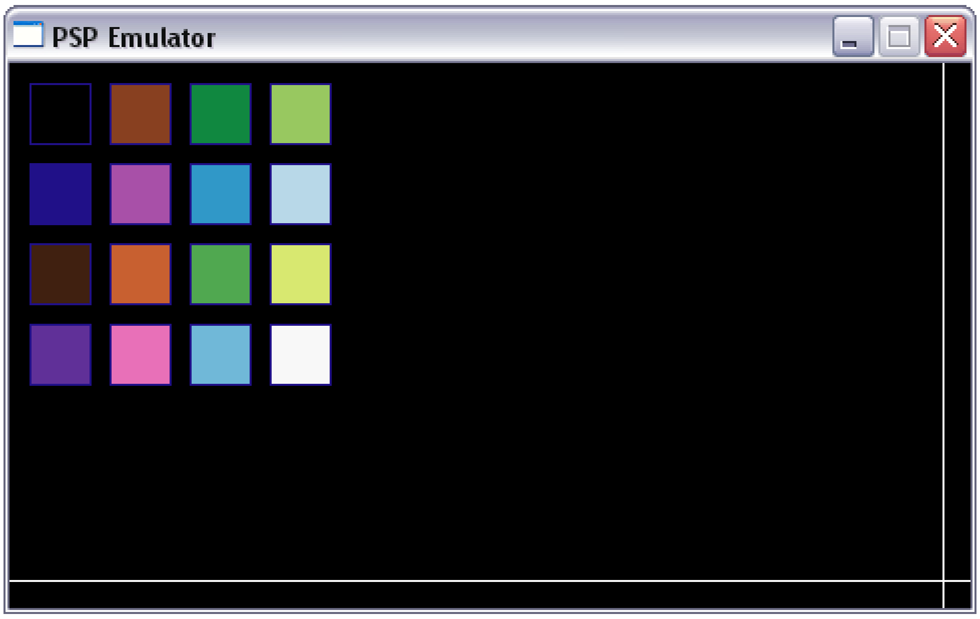
We will now try another experiment: we want to make the rectangles appear in various colors. An idea is to link the color of the rectangle to a numerical variable, that will be calculated rectangle by rectangle using the value of a counter.

Nanodesktop provides the predefined array **ndWndColorVector [x]** that contains the 16 fundamental colors of the graphical system..

Let’s try this code:

```
#include <nanodesktop.h>

int ndMain (void)
{
        int CounterX, CounterY;

        ndInitSystem ();

        for (CounterY=0; CounterY<4; CounterY++)
        {
                for (CounterX=0; CounterX<4; CounterX++)
                {
                      ndBS_DrawRectangle (10+40*CounterX, 10+40*CounterY,
                                                            10+40*CounterX+30, 10+40*CounterY+30,
                                                            ndWndColorVector [4*CounterY+CounterX],
                                                            COLOR_BLUE, NORENDER);
                   }
          }

        BaseScreenRender();
}
```

This is the result.



Note that rectangle color is ndWndColorVector [4*CounterY+CounterX]

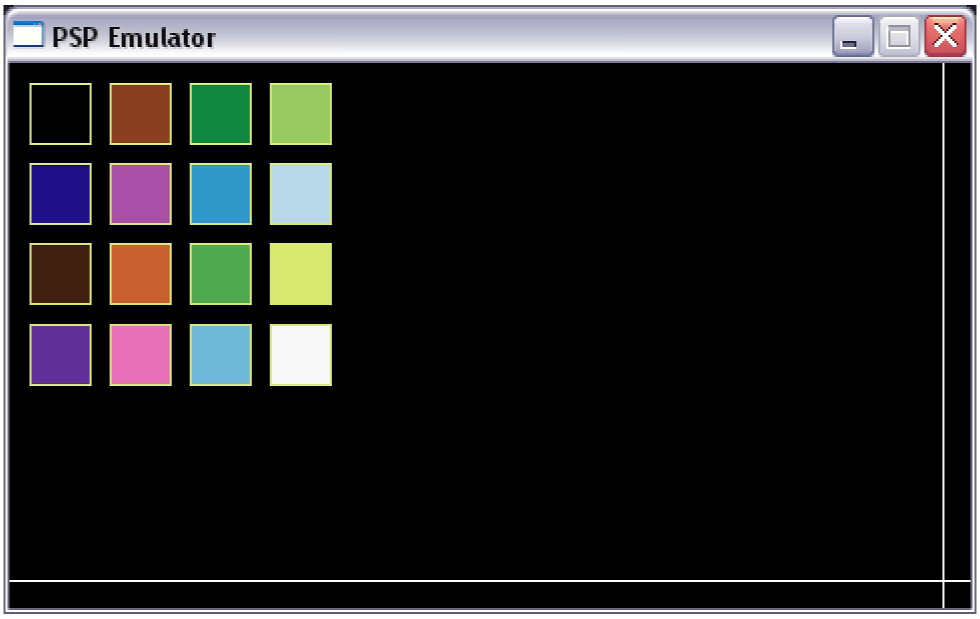
ndBS_DrawSpRectangle

ndBS_DrawRectangle is able exclusively to create rectangles with an edge of 1 pixel. It is very fast, but also limited. Nanodesktop offers an alternative routine, that is far more powerful but not as fast that can be used for this scope: **ndBS_DrawSpRectangle**. Sp is for “special”.

Try this new program:

```
#include <nanodesktop.h>

int ndMain (void)
{
    int CounterX, CounterY;

    ndInitSystem ();

    for (CounterY=0; CounterY<4; CounterY++)
    {
        for (CounterX=0; CounterX<4; CounterX++)
        {
            ndBS_DrawSpRectangle (10+40*CounterX, 10+40*CounterY,
                                                10+40*CounterX+30,  10+40*CounterY+30,
                                               ndWndColorVector [4*CounterY+CounterX],
                                               COLOR_YELLOW, 0, NORENDER);
         }
    }

    BaseScreenRender ();
}
```

This is the result:



The innovation is the presence of a 64 bit parameter before the parameter RenderNow. This parameter, in this example, has been set deliberately to 0. Such parameters call *Attribute* and its bits are mapped in a way that the programmer can set up the features of the rectangle that he wants to be created on the screen.

When ndBS_DrawSpRectangle sees that this parameter is set to 0, it begins to behave itself exactly like ndBS_DrawRectangle.

When, instead, attribute bits are set to values different than 0, the behaviour of DrawSpRectangle changes completely. Fortunately the programmer doesn't have to act on the single bit manually, because Nanodesktop provides some 64 bit functions that create the right bits sequences automatically. Such functions are called KEYS.

The programmer has only to recall the right KEY routine during the assignment of parameter Attribute. If he wants to use various attributes, it is sufficient to call various KEY routines and to make

OR on the results of each of them together, obtaining the final 64-bit value for Attribute that will signal to nd all features that he requires.

It seems difficult to explain, but try it yourself and you'll find it very elementary, when you understand how it works.

Now lets try another program:

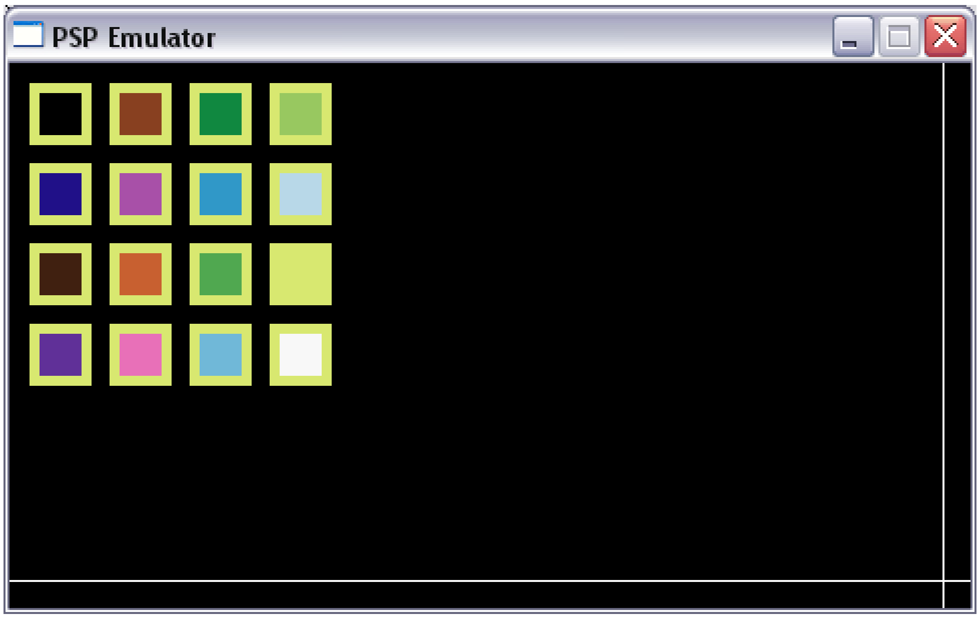
```
#include <nanodesktop.h>

int ndMain (void)
{
    int CounterX, CounterY;

    ndInitSystem ();

    for (CounterY=0; CounterY<4; CounterY++)
    {
        for (CounterX=0; CounterX<4; CounterX++)
        {
              ndBS_DrawSpRectangle (10+40*CounterX, 10+40*CounterY,
                                                10+40*CounterX+30, 10+40*CounterY+30,
                                                ndWndColorVector [4*CounterY+CounterX],
                                                COLOR_YELLOW,
                                                NDKEY_BORDER_SIZE (5), NORENDER);
        }
    }

    BaseScreenRender ();
}
```

Here, the source calls 64-bit function NDKEY_BORDER_SIZE (x): the result will be a group of bits that will be passed like 64-bit parameter Attribute to **ndBS_DrawSpRectangle().**

It's very simple.

In this case, NDKEY_BORDER_SIZE acts on the bits that set the edge of the rectangle: the result is a screen with the same rectangles as the previous program but with an edge of 5 pixels.

The result of the operation is this:



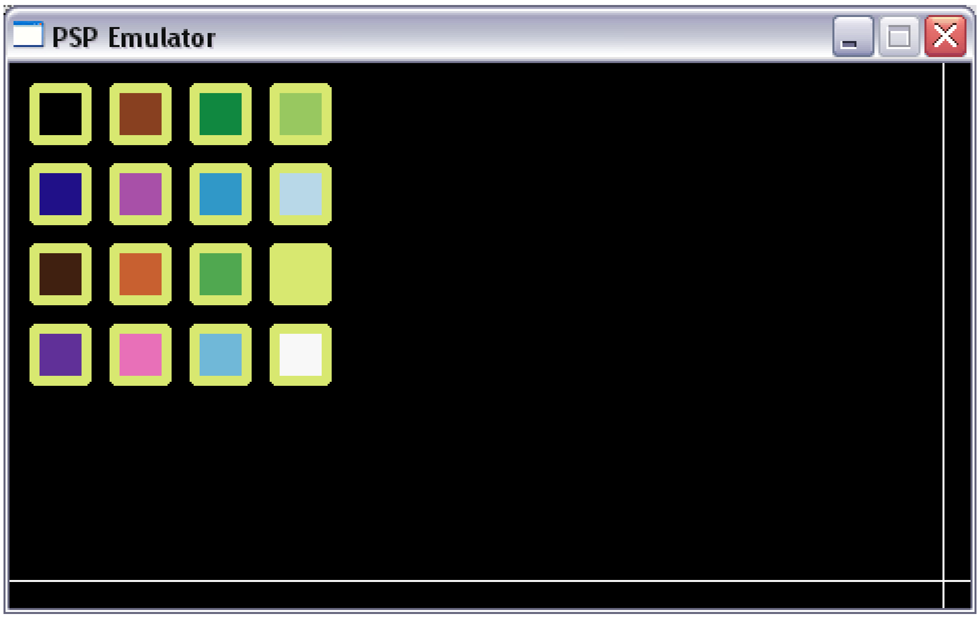
Ok. Now we will try another experiment. I want the new rectangles to have rounded edges. This is very simple to obtain, too.

You only need to modify the previous program, adding for Attribute parameter, an OR with manifest constant ND_ROUNDED.

This is the new program:

```
#include <nanodesktop.h>

int ndMain (void)
{
    int CounterX, CounterY;

    ndInitSystem ();

    for (CounterY=0; CounterY<4; CounterY++)
    {
        for (CounterX=0; CounterX<4; CounterX++)
        {
            ndBS_DrawSpRectangle (10+40*CounterX,  10+40*CounterY,
                                            10+40*CounterX+30, 10+40*CounterY+30,
                                             ndWndColorVector [4*CounterY+CounterX],
                                            COLOR_YELLOW,
                                             NDKEY_BORDER_SIZE (5) | ND_ROUNDED, NORENDER);
        }
    }

    BaseScreenRender ();
}
```

And this is the result:



Using the combination of a key and of an explicit constant you can obtain a rectangle with a rounded edge and a large 5 pixel border.

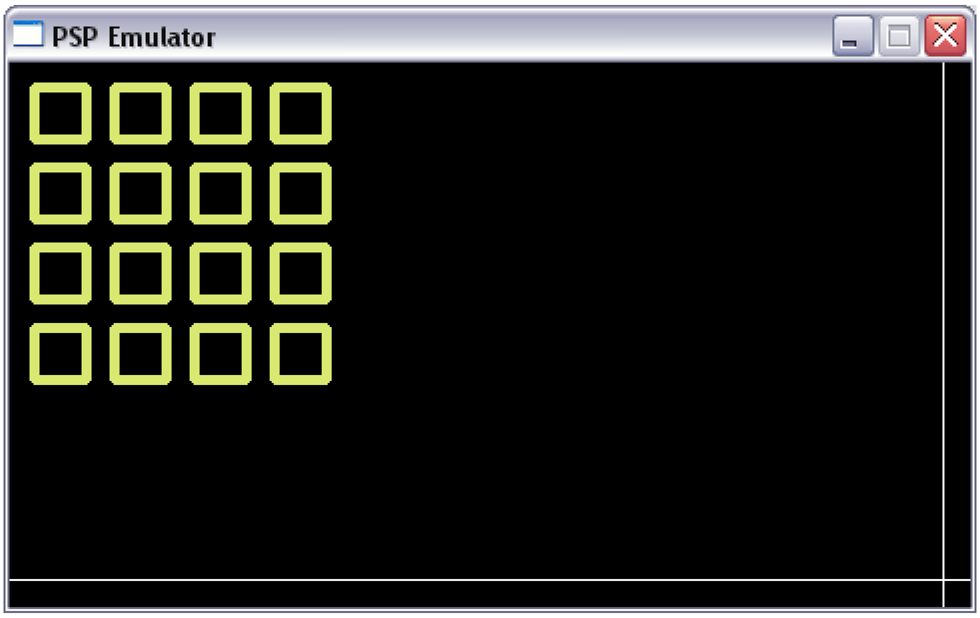
And if you want the rectangle to be empty? Setting to 0 the value of color of the rectangle is not the solution, because every pixel that is found within the rectangle is colored black, from the black pixels generated by the Nanodesktop graphical routines.

In reality, Nanodesktop supplies another explicit constant that creates “an empty” rectangle, i.e. a rectangle with only the contour and not the inside. It is sufficient to use **ndBS_DrawSpRectangle**, adding as Attribute parameter (or OR together other parameters as usual) the explicit constant ND_VOID.

We'll therefore try:

```
#include <nanodesktop.h>

int ndMain (void)
{
    int CounterX, CounterY;

    ndInitSystem ();

    for (CounterY=0; CounterY<4; CounterY++)
    {
        for (CounterX=0; CounterX<4; CounterX++)
        {
            ndBS_DrawSpRectangle (10+40*CounterX,  10+40*CounterY,
                                           10+40*CounterX+30, 10+40*CounterY+30,
                                            ndWndColorVector [4*CounterY+CounterX],
                                           COLOR_YELLOW,
                                              NDKEY_BORDER_SIZE (5) | ND_ROUNDED | ND_VOID, NORENDER);
        }
    }

    BaseScreenRender ();
}
```

And this is the result:



As you can see, the parameter that is related to color (ndWndColorVector) is ignored and the system limits itself to create void rectangles.

ndBS_DrawRtRectangle

Now we are ready to try a new function. We want to create some rhombi on screen. In order to do this we need a new routine: **ndBS_DrawRtRectangle**.

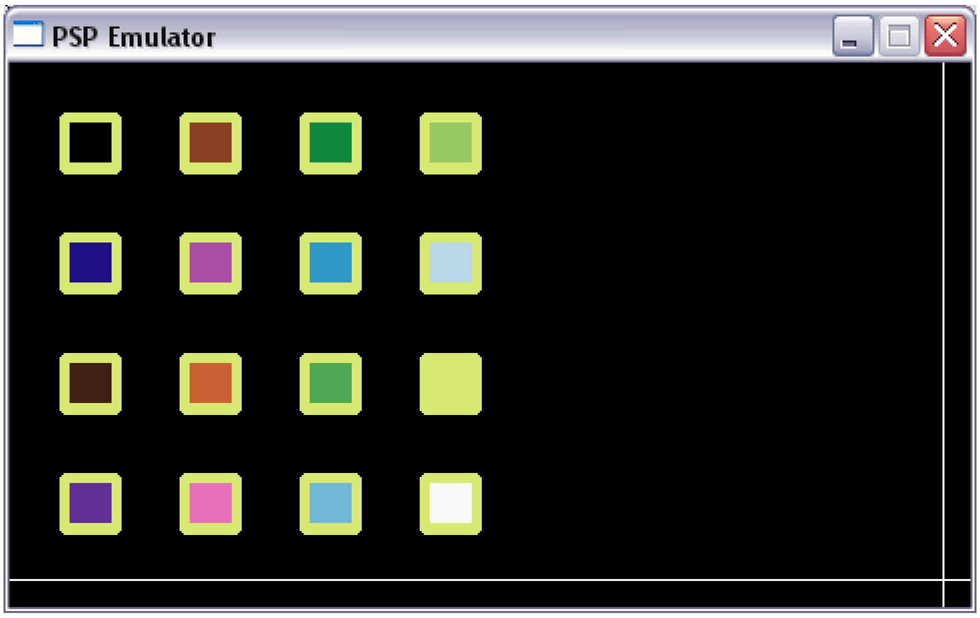
You must be careful about the fact that this routine is very different to the previous one.
The first 4 parameters do not define the coordinates in which you want the rectangle to be drawn, they have a different meaning.

The first 2 parameters of **ndBS_DrawRtRectangle** define the coordinates of the centre of the rectangle, while the third and fourth parameters define the X and Y dimensions of the rectangle (length and width). **ndBS_DrawRtRectangle** can be used like **DrawSpRectangle** providing that you convert the parameters accordingly.

We will try this program now:

```
#include <nanodesktop.h>

int ndMain (void)
{
    int CounterX, CounterY;

    ndInitSystem ();

    for (CounterY=0; CounterY<4; CounterY++)
    {
        for (CounterX=0; CounterX<4; CounterX++)
        {
            ndBS_DrawRtRectangle (40+60*CounterX, 40+60*CounterY, 30, 30,
                 ndWndColorVector [4*CounterY+CounterX], COLOR_YELLOW,
                 NDKEY_BORDER_SIZE (5) | ND_ROUNDED, NORENDER);
        }
    }

    BaseScreenRender ();
}
```

The result is similar to the previous one:



The only difference is that now I have changed some values (as in the distance between rectangles), and that I have done all this with the function that centres rectangles (remember that we are using **ndBS_DrawRtRectangle**).

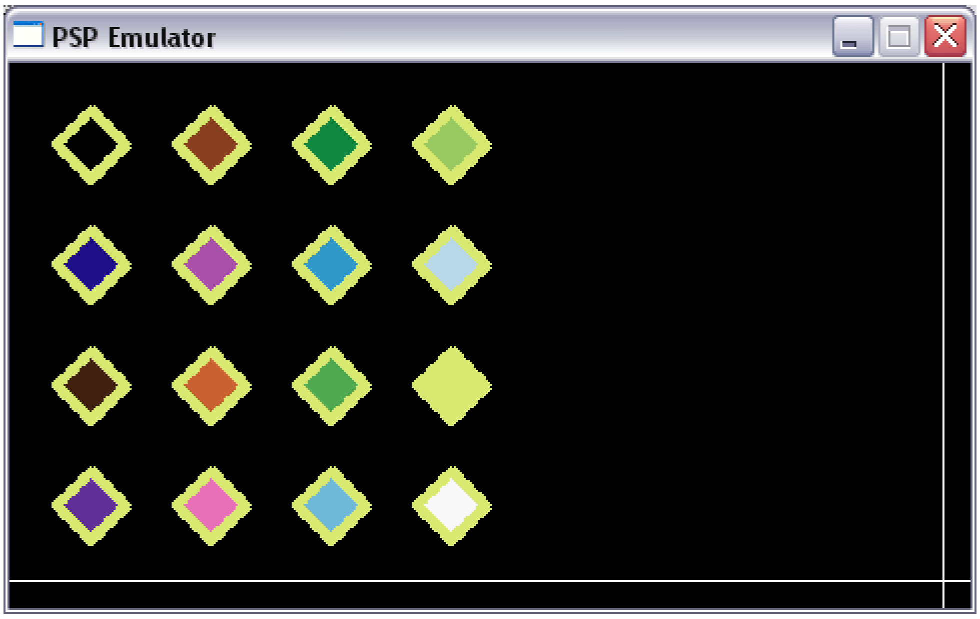
Note also that **ndBS_DrawRtRectangle** uses the same 64-bit keys that are accepted by **ndBS_DrawSpRectangle**, so you can set up the border width or rounded attribute in the same way.

The main difference of **ndBS_DrawRtRectangle** is that it accepts the key NDKEY_ROTATE(x). The x parameter of the key function is between -180 degrees and +180 degrees. Using this key, you can specify how many degrees of rotation you require before the element will be visualized on the screen.

Here is an example:

```
#include <nanodesktop.h>

int ndMain (void)
{
    int CounterX, CounterY;

    ndInitSystem ();

    for (CounterY=0; CounterY<4; CounterY++)
    {
        for (CounterX=0; CounterX<4; CounterX++)
        {
            ndBS_DrawRtRectangle (40+60*CounterX, 40+60*CounterY, 30, 30,
                 ndWndColorVector [4*CounterY+CounterX], COLOR_YELLOW,
                 NDKEY_BORDER_SIZE (5) | ND_ROUNDED | NDKEY_ROTATE (45),
                 NORENDER);
        }
    }

    BaseScreenRender ();
}
```

And here are the rhombi:



ndBS_DrawCircle and ndBS_DrawArcOfCircle

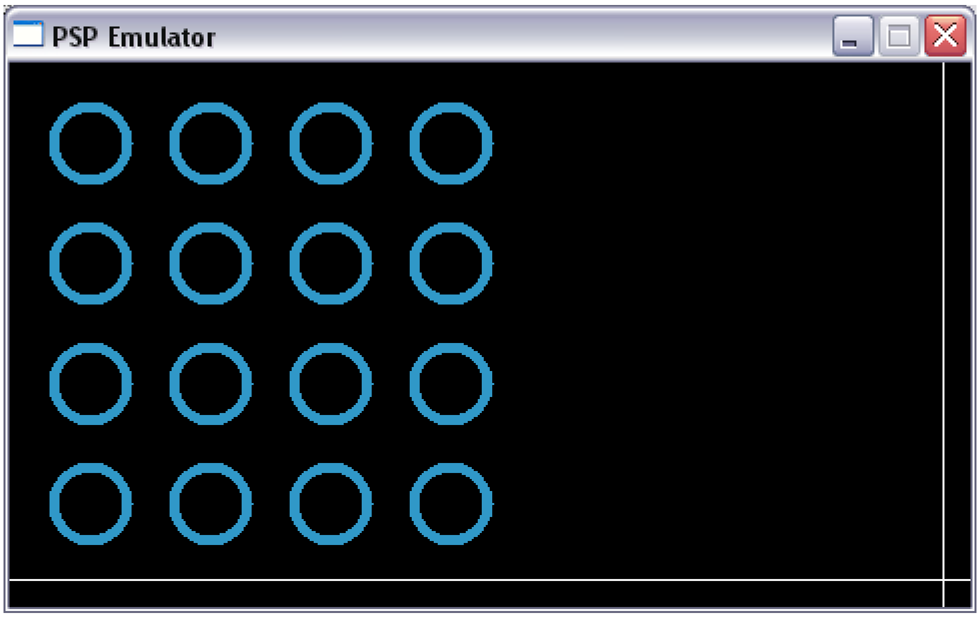
In the previous paragraphs we have learned to use DrawRectangle and other associated routines in order to draw rectangles on the screen.

In this lesson we will learn to draw circles and ellipses in the BaseScreen using the dedicated routines.

We will begin with this example:

```
#include <nanodesktop.h>


int ndMain (void)
{
    int CounterX, CounterY;

    ndInitSystem ();

    for (CounterY=0; CounterY<4; CounterY++)
    {
        for (CounterX=0; CounterX<4; CounterX++)
        {
            ndBS_DrawCircle       (40+60*CounterX, 40+60*CounterY,
                                   20, COLOR_BLACK, COLOR_LBLUE,
                                   NDKEY_BORDER_SIZE (2), NORENDER);
        }
    }

    BaseScreenRender ();
}
```

The result is this:



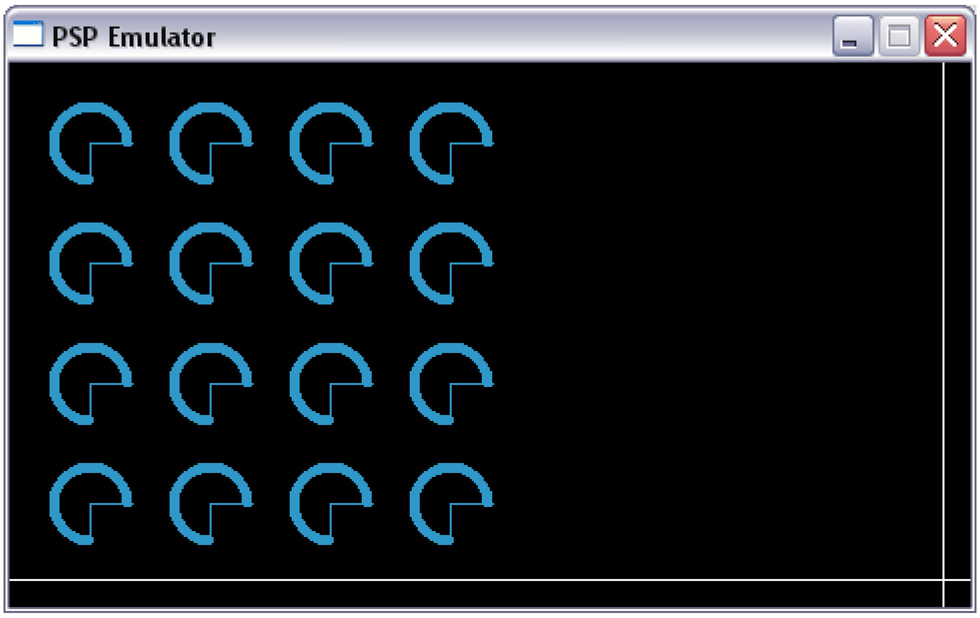
As you can see, a new routine is used: **ndBS_DrawCircle**. This routine works in the following way: the first parameters are the coordinates of the centre of the circle, the third is the radius. Two 16-bit values follow: they are the magic numbers related to the inner color of the circle and the edge. The following parameter is a 64-bit code that specifies the features of the graphical element: in this case we have simply set the border size to 2 pixels. At the end, there is the usual NORENDER (of which we already know the meaning)

Is it possible to create arcs of a circle ? The answer is yes: the function is called **ndBS_DrawArcOfCircle**

Try this example:

```
#include <nanodesktop.h>

int ndMain (void)
{
    int CounterX, CounterY;
    ndInitSystem ();

    for (CounterY=0; CounterY<4; CounterY++)
    {
        for (CounterX=0; CounterX<4; CounterX++)
        {
            ndBS_DrawArcOfCircle  (40+60*CounterX, 40+60*CounterY,
                                   20, 0, 270, COLOR_BLACK, COLOR_LBLUE,
                                   NDKEY_BORDER_SIZE (2) | ND_ROUNDED, NORENDER);
        }
    }

    BaseScreenRender ();
}
```

This is the result:



The parameters 0,270 specify the initial and final angular coordinates of the arc of the circle.

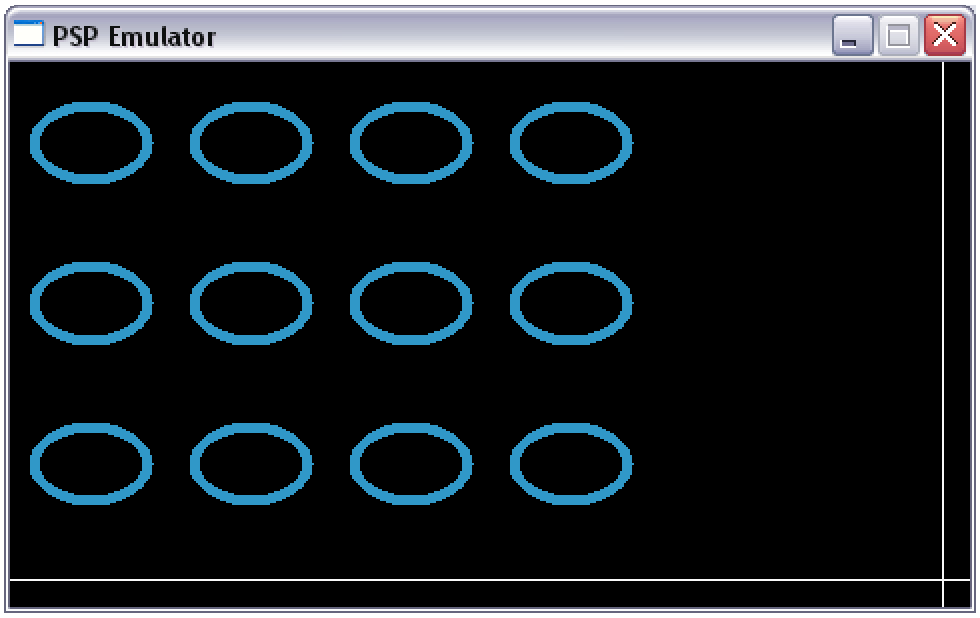
ndBS_DrawEllipse and ndBS_DrawArcOfEllipse

Nanodesktop also supports the drawing of arcs of ellipses or ellipses in the BaseScreen.

The dedicated functions are these: ndBS_DrawEllipse and ndBS_DrawArcOfEllipse
Try this code:

```
#include <nanodesktop.h>

int ndMain (void)
{
    int CounterX, CounterY;

    ndInitSystem ();

    for (CounterY=0; CounterY<4; CounterY++)
    {
        for (CounterX=0; CounterX<4; CounterX++)
        {
            ndBS_DrawEllipse      (40+80*CounterX, 40+80*CounterY,
                                   30, 20,  COLOR_BLACK, COLOR_LBLUE,
                                   NDKEY_BORDER_SIZE (2) | ND_VOID, NORENDER);
        }
    }

    BaseScreenRender ();
}
```

The result is shown on the following page:

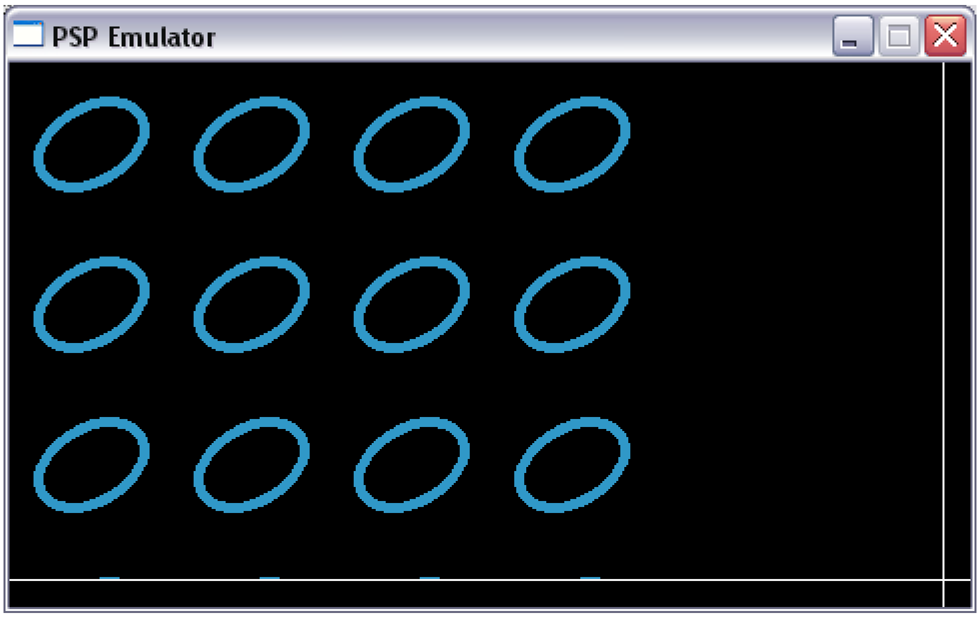


Obviously, you can use NDKEY_ROTATE(x) also with the DrawEllipse routine. So you can draw ellipses that have axis oriented in directions that aren't parallel to the screen axis.

Here is an example:

```
#include <nanodesktop.h>

int ndMain (void)
{
    int CounterX, CounterY;

    ndInitSystem ();

    for (CounterY=0; CounterY<4; CounterY++)
    {
        for (CounterX=0; CounterX<4; CounterX++)
        {
            ndBS_DrawEllipse      (40+80*CounterX, 40+80*CounterY,
                                   30, 20,  COLOR_BLACK, COLOR_LBLUE,
                                   NDKEY_BORDER_SIZE (2) | ND_VOID | NDKEY_ROTATE (30),
                                      NORENDER);
        }
    }

    BaseScreenRender ();
}
```

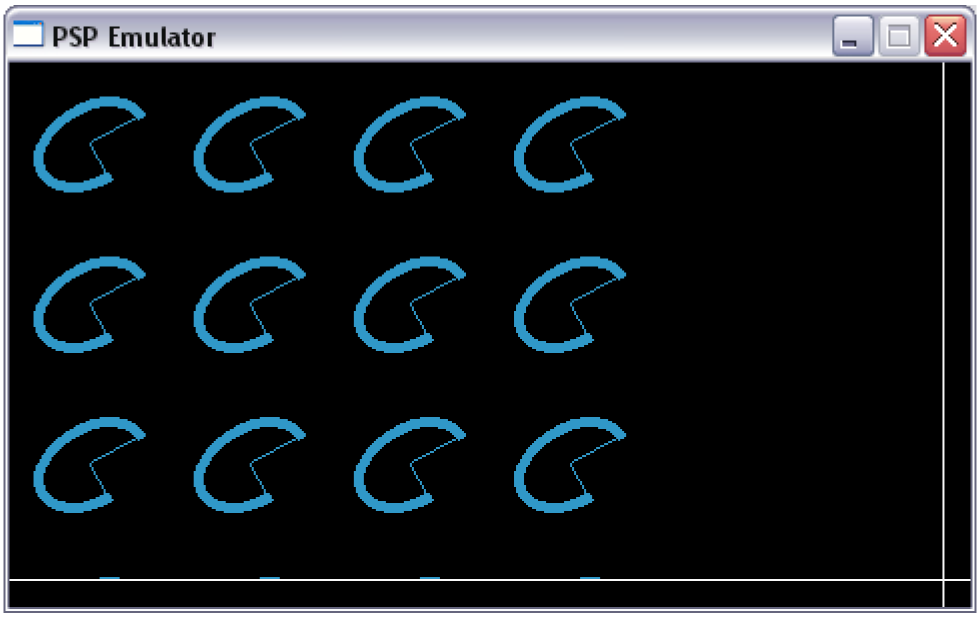
And this is the result:



Now we can try another example. Here we use ndBS_DrawArcOfEllipse:

```
#include <nanodesktop.h>

int ndMain (void)
{
    int CounterX, CounterY;

    ndInitSystem ();

    for (CounterY=0; CounterY<4; CounterY++)
    {
        for (CounterX=0; CounterX<4; CounterX++)
        {
            ndBS_DrawArcOfEllipse (40+80*CounterX, 40+80*CounterY,
                                   30, 20, 0, 270, COLOR_BLACK, COLOR_LBLUE,
                                   NDKEY_BORDER_SIZE (2) | ND_VOID | NDKEY_ROTATE (30), NORENDER);
        }
    }

    BaseScreenRender ();
}
```

The result is this:



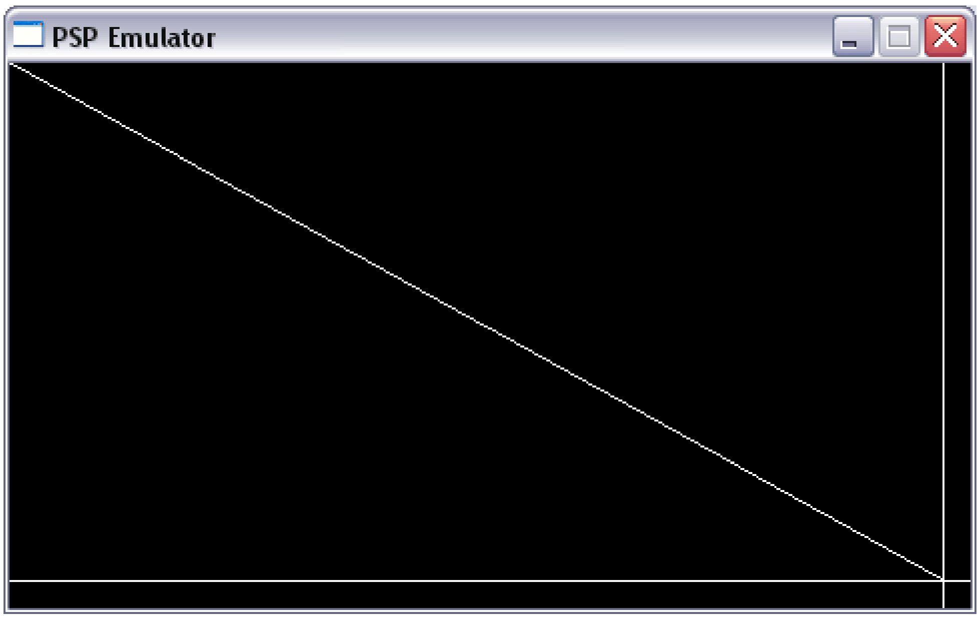
ndBS_DrawLine, ndBS_DrawPoly, ndBS_FillArea

Nanodesktop provides routines that are able to draw on screen lines, polygons and complex objects. The simplest of these routines is **ndBS_DrawLine**.

See this example:

```
#include <nanodesktop.h>

int ndMain (void)
{
    int CounterX, CounterY;

    ndInitSystem ();
    ndBS_DrawLine (0, 0, Windows_MaxScreenX, Windows_MaxScreenY, COLOR_WHITE, RENDER);
}
```

Here is the result:



The prototype of **ndBS_DrawLine** is very simple. If we want to create a broken line or a polygon, we could use numerous calls to **ndBS_DrawLine**. Fortunately this is not necessary.

In fact, Nanodesktop provides a dedicated routine that allows you to create a polygon. Its name is **ndBS_DrawPoly**. The prototype of this function is:

```
ndBS_DrawPoly ( RenderNow, Color, NrPixels, (x0, y0), (x1, y1), (x2, y2), .....)
```

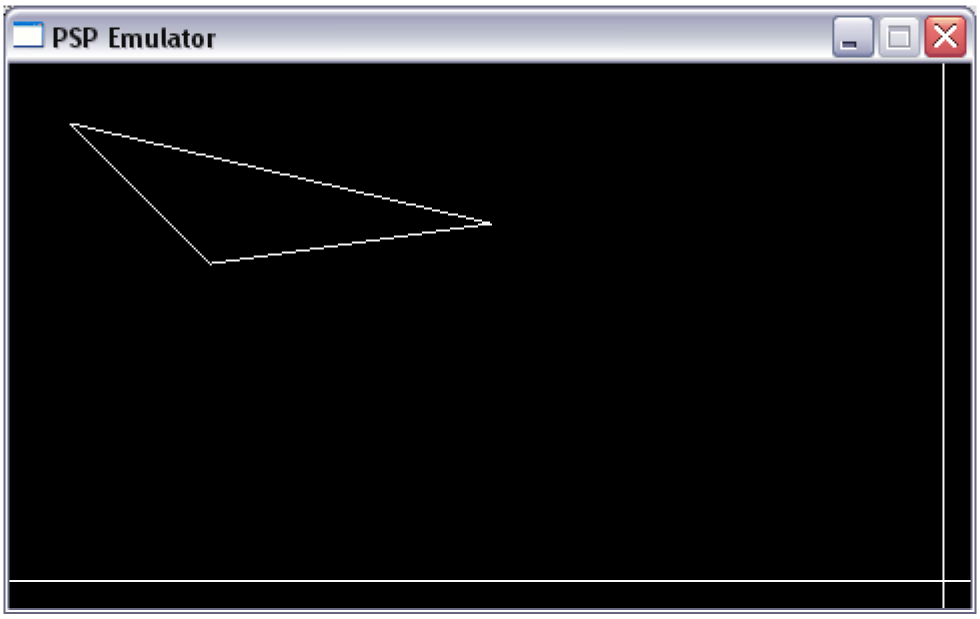
It is a variable parameters function. The first param is RENDER or NORENDER, the second is the chosen color, the third is the number of pixels, and the following parameters are the coordinates X-Y of each pixel.

Be careful that Nanodesktop has no way to check if the number of pixel coordinates entered is correct.

For example, the following program creates a polygon with vertex (30, 30), (100, 100), (240, 80), (30, 30). Note that the first pixel is identical to the last one: this means that the polygon that we want is a *closed polygon.*

```
#include <nanodesktop.h>

int ndMain (void)
{
    int CounterX, CounterY;

    ndInitSystem ();
    ndBS_DrawPoly (COLOR_WHITE, RENDER, 4, 30, 30, 100, 100, 240, 80, 30, 30);
}
```

Result:



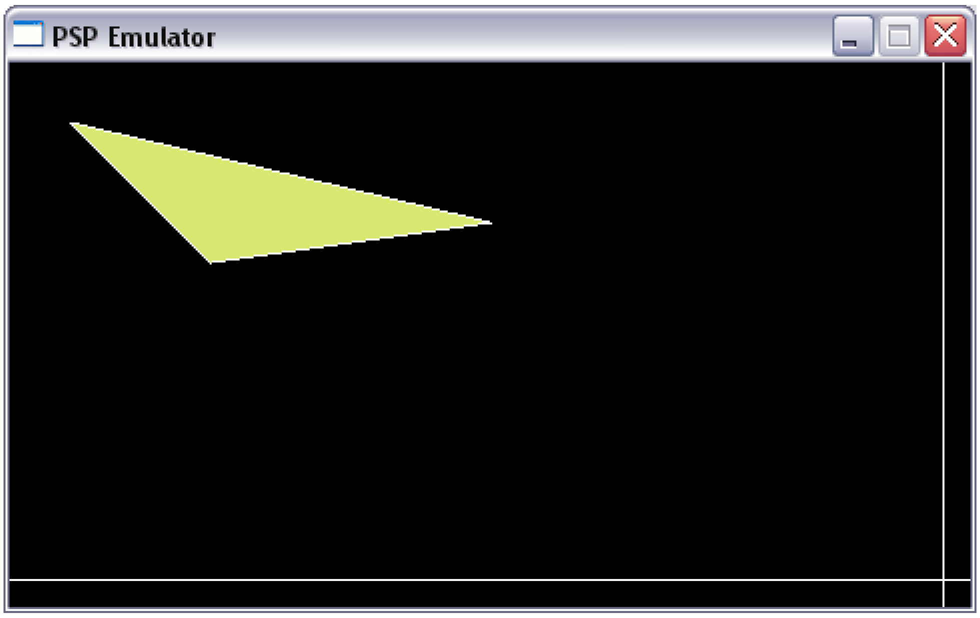
Now, we can fill the triangle. The function that we can use is **ndBS_FillArea**.

See this code:

```
#include <nanodesktop.h>

int ndMain (void)
{
    int CentroX, CentroY;

    ndInitSystem ();

    CentroX = (30+100+240)/3;
    CentroY = (30+100+80)/3;

    ndBS_DrawPoly (COLOR_WHITE, NORENDER, 4, 30, 30, 100, 100, 240, 80, 30, 30);
    ndBS_FillArea (CentroX, CentroY, COLOR_YELLOW, COLOR_WHITE, RENDER);
}
```

Note, first we calculate the centre of the triangle and afterwards we begin the filling operation from this pixel.

The result is shown on the following page.

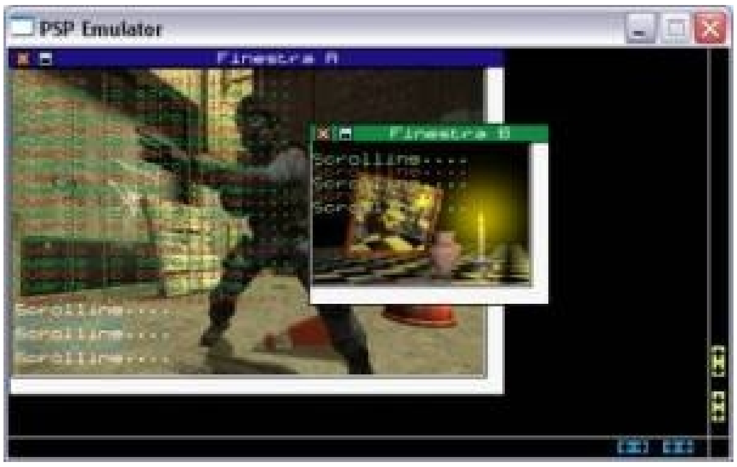


ndBS_GrWriteChar, ndBS_GrWriteLn, ndBS_GrPrintLn

Now we'll see the way for writing a string or a char in the BaseScreen.

Here, it is necessary for a short preamble. In Nanodesktop there are two classes for writing routines: the *textual* and the *graphical* writing routines.

The *textual routines* support the operations of writing in the *charmap*. This means that you can write in the charmap and the text will appear on the images that you have drawn in your graphical space. Examples of these routines are **ndWS_WriteChar, ndWS_WriteLn, ndWS_PrintLn**. The textual routines support overscan and text scrolling.



The textual routines write the chars to the *actual cursor position,* if you haven't specified differently.

These routines also supports functions such as **ndWS_GoCursor**, this moves the cursor pointer to a given position.

The *graphical writing routines* are quite different. Examples of these routines are **ndBS_GrWriteLn, ndBS_GrWriteChar, ndBS_GrPrintLn, ndWS_GrWriteLn, ndWS_GrWrite-Char, ndWS_GrPrintLn**.

These routines draw a string on the screen *as a simple graphical element, i.e. A group of pixels.* Graphical routines don't support cursor (the position must be specified in each call), don't support overscan, and don't support scrolling. However, they can draw strings using fonts that aren't large 8x8 pixels, and they can draw strings oriented with a given angle.

The BaseScreen supports only *graphical routines.* Textual routines are supported only in the window space.

Let's view this example:

```
#include <nanodesktop.h>

int ndMain (void)
{
    int Counter;

    ndInitSystem ();

    for (Counter=0; Counter<10; Counter++)
    {
        ndBS_GrWriteChar (30 + 10*Counter, 50, 'A'+Counter, COLOR_WHITE, COLOR_BLUE, NORENDER);
    }

    BaseScreenRender ();

    ndBS_GrWriteLn (100, 100, "Hello world", COLOR_WHITE, COLOR_RED, RENDER);
}
```

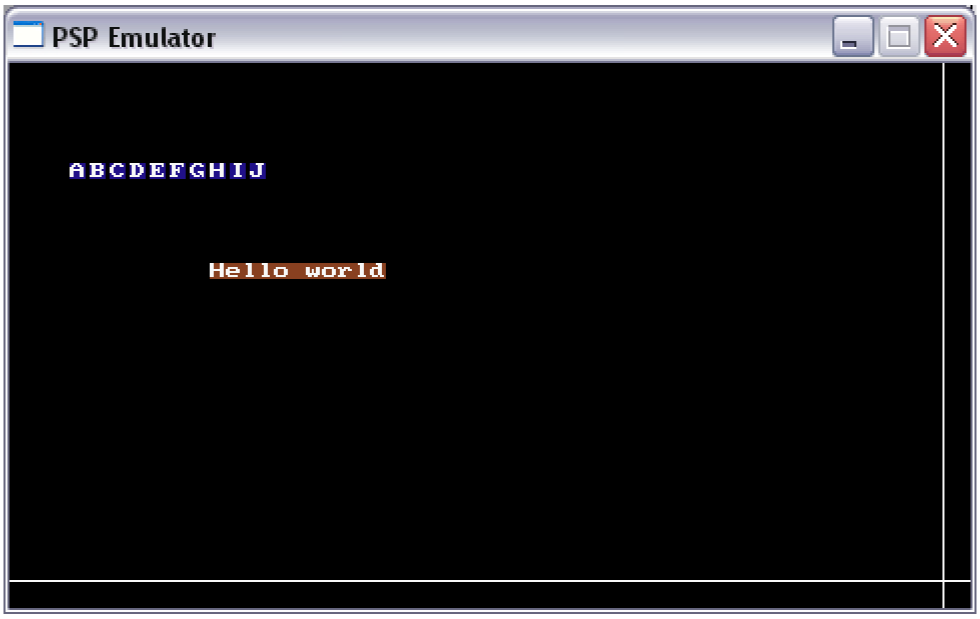
The program is very simple: the first *for* loop prints a single series of characters (A, B, C, D, E, F, G, H, I, J) on the screen.
The first 2 parameters indicate the position of the character (for being precise, of the first pixel up to the left of the character), the third parameter is a char that contains just the letter given to be visualized.

The fourth and fifth parameters are the color of the character and the color that will be assumed for the background behind the character.

The last parameter NORENDER has the usual meaning: the *for* loop executes the visualization of 10 characters in the background, and then **BaseScreenRender** makes them appear immediately.

Also **ndBS_GrWriteLn** has been used. This routine is similar to the previous **ndBS_GrWriteChar**, but with an important difference. The third parameter is a string constant. This is the string that will be visualized.

Here is the result:



```
ABCDEFGHIJ

Hello world
```

But now we'll make another hypothesis: suppose we want Nanodesktop to display a number, in a way similar to the printf function in C. ndBS_GrWriteLn isn’t useful in this case, because the function accepts a fixed number of parameters, so you cannot use it as you would use printf.

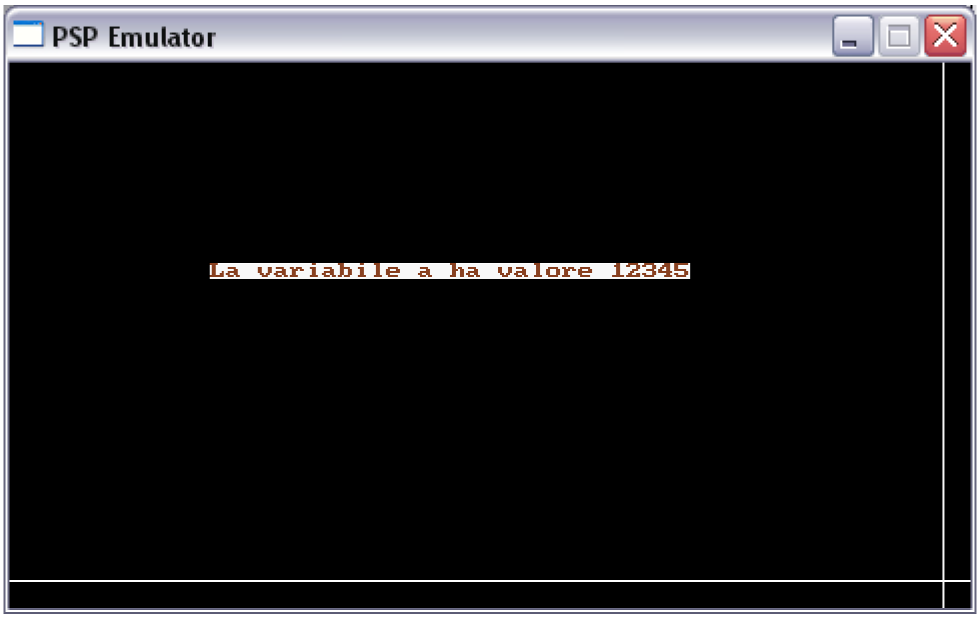
Fortunately, Nanodesktop comes to our aid with an appropriate routine: **ndBS_GrPrintLn**. The prototype of this function is this:

```
void ndBS_GrPrintLn (short unsigned int RRPosPixelX, short unsigned int RRPosPixelY, TypeColor
Color, TypeColor BGColor, ndint64 TextCode, char *FirstParam, ...);
```

Try this source example:

```
#include <nanodesktop.h>

int ndMain (void)
{
    int a = 12345;

    ndInitSystem ();
    ndBS_GrPrintLn (100, 100, COLOR_RED, COLOR_WHITE, RENDER, "La variabile a ha valore %d", a);
}
```

Here is the result:



```
La variabile a ha valore 12345
```

Fonts

Nanodesktop supports different *fonts.*

Under Nanodesktop, there are two types of fonts: *NTF Font (Nanodesktop Text Fonts) and TrueType Fonts.* Support for TrueType fonts will be added in Nanodesktop 0.4, so this section only addresses the use of NTF fonts.

NTF fonts are *bitmapped fonts,* or *array fonts,* of 8x8 pixels. A utility, in **<ndenv>\PSP\SDK\Tools\NTF Creator,** allows you to create a new .NTF font file with specific font data, starting from its array representation.

All fonts in nd, are assigned a number, called *font handler.* This number specifies which font you are using or you are requesting in a routine.

To load/free a NTF font, you can use the routines:

```
short int ndFONT_LoadFont (char *FntName);
char ndFONT_DestroyFont (char FntHandle)
```

That are defined in *<ndenv path>\PSP\SDK\Nanodesktop\src*

Fortunately, Nanodesktop pre-loads some *system fonts,* that are available for applications, without explicit loading by ndFONT_LoadFont.

In Nanodesktop 0.3, there are 4 system fonts (designated by handlers 1,2,3,4).

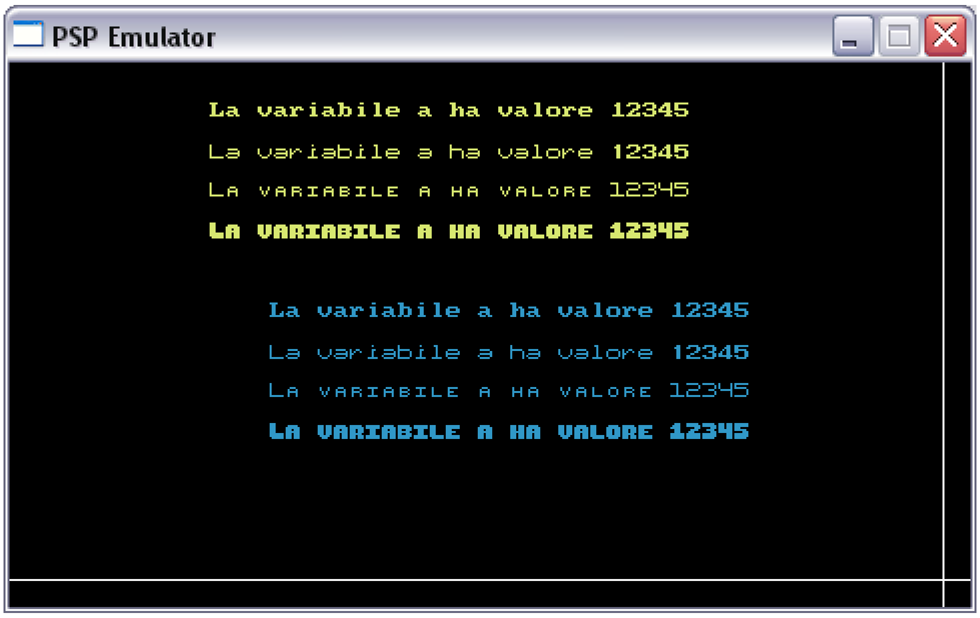
The user can select the font in two ways:

a) he can use a function, such as ndBS_SetFont, that changes in a permanent way the default BS font;
b) he can use the NDKEY_FONT routine that generates a 64 bit code that can be OR-ed to NORENDER or RENDER in ndBS_WriteLn or ndBS_PrintLn, changing the font in use only for that single element (the operation doesn't change the default font).

See this code:

```
#include <nanodesktop.h>

int ndMain (void)
{
    int a = 12345;

    ndInitSystem ();

    // Default font is set to 1. As if ndBS_SetFound(1) is here....
    ndBS_GrPrintLn (100, 20, COLOR_YELLOW, COLOR_BLACK, RENDER, "La variabile a ha valore %d", a);

    ndBS_SetFont (2);
    ndBS_GrPrintLn (100, 40, COLOR_YELLOW, COLOR_BLACK, RENDER, "La variabile a ha valore %d", a);

    ndBS_SetFont (3);
    ndBS_GrPrintLn (100, 60, COLOR_YELLOW, COLOR_BLACK, RENDER, "La variabile a ha valore %d", a);

    ndBS_SetFont (4);
    ndBS_GrPrintLn (100, 80, COLOR_YELLOW, COLOR_BLACK, RENDER, "La variabile a ha valore %d", a);

    // NDKEY Method
     ndBS_GrPrintLn (130, 120, COLOR_LBLUE, COLOR_BLACK, NDKEY_FONT (1) | RENDER, "La variabile a ha
valore %d", a);
     ndBS_GrPrintLn (130, 140, COLOR_LBLUE, COLOR_BLACK, NDKEY_FONT (2) | RENDER, "La variabile a ha
valore %d", a);
     ndBS_GrPrintLn (130, 160, COLOR_LBLUE, COLOR_BLACK, NDKEY_FONT (3) | RENDER, "La variabile a ha
valore %d", a);
     ndBS_GrPrintLn (130, 180, COLOR_LBLUE, COLOR_BLACK, NDKEY_FONT (4) | RENDER, "La variabile a ha
valore %d", a);

}
```

This source shows either method to change fonts:



```
La variabile a ha valore 12345
La variabile a ha valore 12345
LA VARIABILE A HA VALORE 12345
LA VARIABILE A HA VALORE 12345

La variabile a ha valore 12345
La variabile a ha valore 12345
LA VARIABILE A HA VALORE 12345
LA VARIABILE A HA VALORE 12345
```

Nanodesktop errors

How does Nanodesktop manage errors ?

For the larger part of the routines, there is a rule that you must remember:

a) if the routine returns *a positive value (or 0),* the operation has been terminated successfully, and the value returned is valid information;
b) if the routine returns *a negative value,* there is an error and the absolute value associated is an error code. The sources of nd, specifies the known error code for each subroutine.

The meaning of each error code can be found in the file

<ndenv path>\PSP\SDK\Nanodesktop\src\$$Errors.h

Chapter 9

Visualize an image in BS

Nanodesktop supports visualization of images either in the base screen or in windows space.

In order to load an image into memory, you must define a struct ndImage_Type

```
struct ndImage_Type
{
    unsigned short int LenX, LenY;

    char ColorFormat;
    char IsStatic;
    TypeColor *Data;                // Puntatore ai dati (MagicNumber) dell'immagine.
};
```

This struct contains information about the image that you want to manipulate. The first operation that must be done on this struct is the *initialization.* This is done by a dedicated function: **ndIMG_InitImage**.

```
Struct ndImage_Type MyImage;

ndIMG_InitImage (&MyImage);             // Initialize MyImage structure
```

Remember that initialization sets to zero the Data field in the structure. This is very important: if you use an uninitialized struct ndImage_Type in the following routines, your application will crash.

After the struct has been initialized, we must decide what we want to do with it.

Loading images from a file

We begin with *image loading*. Under nd, there are two ways to load a new image into ram: using **ndIMG_LoadImage** and **ndIMG_LoadImageFromNDFArray**

The function ndIMG_LoadImage has the following prototype:

```
ndIMG_LoadImage (struct ndImage_Type *MyImage, char *NameFile, char ColorFormat);
```

The first parameter is the address of struct ndImage_Type that will contain data about the image.

The second parameter is the address of a string that contains the path and filename from which the image will be read.

The third parameter is very particular: it can assume two values: NDMGKNB (*Nanodesktop magic number)* or NDRGB (*Nanodesktop RGB).*

When you choose NDRGB, the data is stored in ram using an 8-bit RGB format (8 bit for red channel, 8 bit for green channel and 8 bit for blue channel).

When you choose NDMGKNB, the data is stored in RGB555 pixel format: each pixel is represented by a 16-bit value, in which 5 bits are reserved for red component, 5 bits are reserved for green component and 5 bits are reserved for blue component (a bit is ignored).

As the PSP doesn’t support color depths better than RGB555, if you choose NDRGB or NDMGKNB, the results will be the same. If you choose NDRGB, the data will occupy 33% more space

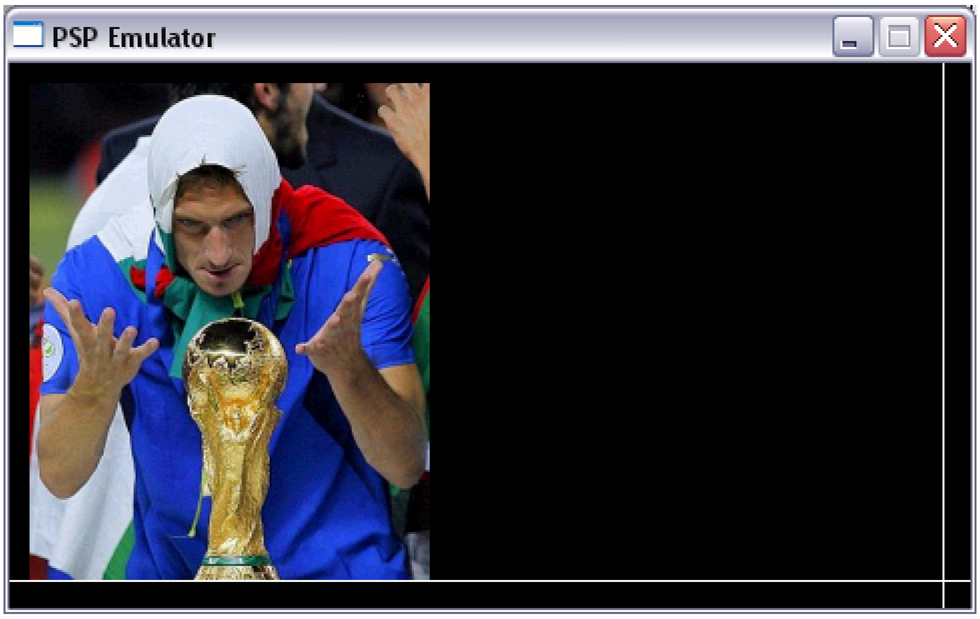
in ram, but when you'll try to visualize the image, nd will automatically scale the color depth and you won't have any advantage.
The main differences between NDRGB and NDMGKNB are appreciable only in these cases:

- when you save the image to disk. That's because the RGB images are visualized in RGB555 on screen, but in ram are managed as normal RGB images;
- when you convert images to *OpenCV format*

This program provides an example to load an image from a file and to visualize it on the screen:

```
#include <nanodesktop.h>

int ndMain (void)
{
    struct ndImage_Type MyImage;
    char ErrRep;

    ndInitSystem ();

    ndIMG_InitImage (&MyImage);
    ErrRep=ndIMG_LoadImage (&MyImage, "ms0:/prova.bmp", NDMGKNB);

    if (!ErrRep)
    {
       ndIMG_ShowImage_InBS (&MyImage, 10, 10, RENDER);
    }
    else
        printf ("Failed to load image. Error reported %d \n", -ErrRep);

    ndDelay (30);
}
```

Pay attention to the fact that this source opens a bitmap (.BMP). This is the result:



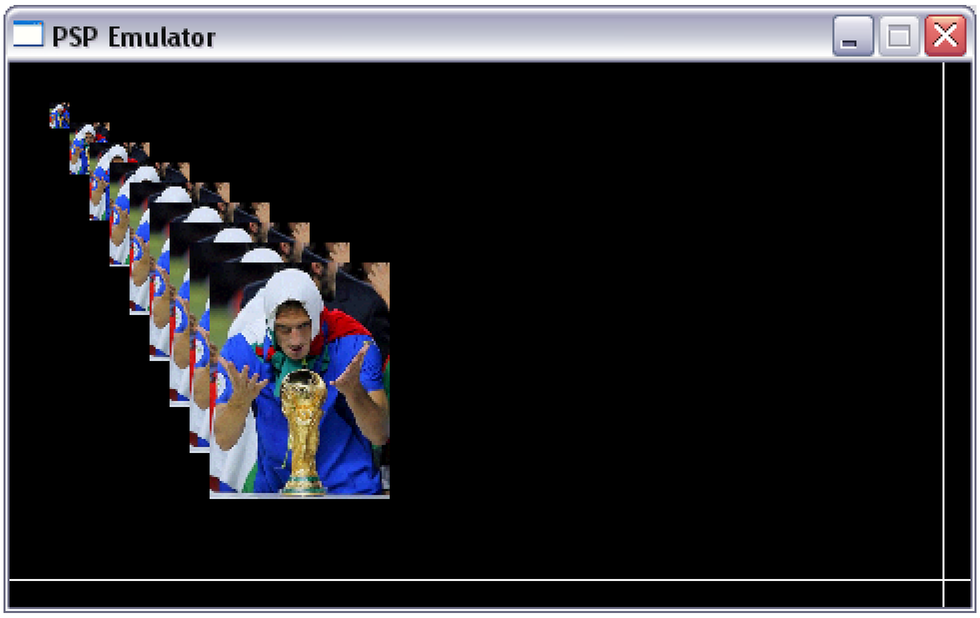
The routine **ndIMG_ShowImage_InBS,** visually places the image at position (10,10) in the BaseScreen. The part of the image that doesn't fit on the screen, will be cut.

Nanodesktop also provides a modified version of ndIMG_ShowImage_InBS, that allows you to scale the image before it is visualized on screen: **ndIMG_ShowImageScaled_InBS.**

View this source:

```
#include <nanodesktop.h>

int ndMain (void)
{
    struct ndImage_Type MyImage;
    char ErrRep;

    ndInitSystem ();
    ndIMG_InitImage (&MyImage);
    ErrRep=ndIMG_LoadImage (&MyImage, "ms0:/prova.bmp", NDMGKNB);

    if (!ErrRep)
    {
       int Counter;

       for (Counter=0; Counter<10; Counter++)
       {
              ndIMG_ShowImageScaled_InBS  (&MyImage,  10+10*Counter,  10+10*Counter,  0.05*Counter,
              0.05*Counter, RENDER);
        }
    }
    else
        printf ("Failed to load image. Error reported %d \n", -ErrRep);

    ndDelay (30);
}
```

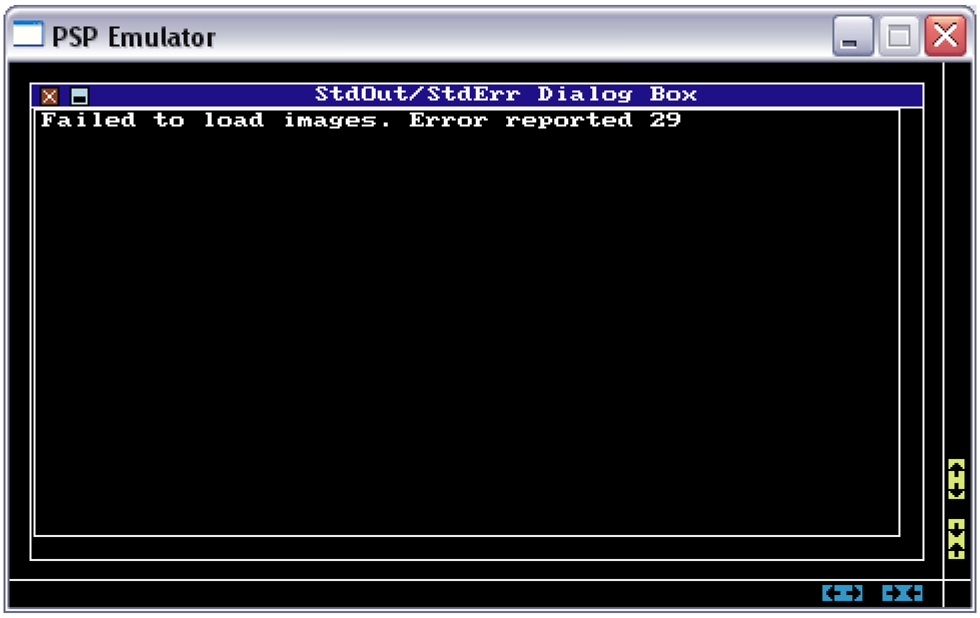
This example loads a bitmap from disk and then represents it on the screen with different scales (from 0.05 to 0.5 (i.e, from 5% to 50%).



Now try to replace the previous code, using provap.bmp, instead of prova.bmp.

```
#include <nanodesktop.h>

int ndMain (void)
{
    struct ndImage_Type MyImage;
    char ErrRep;

    ndInitSystem ();

    ndIMG_InitImage (&MyImage);
    ErrRep=ndIMG_LoadImage (&MyImage, "ms0:/provap.bmp", NDMGKNB);

    if (!ErrRep)
    {
       ndIMG_ShowImage_InBS (&MyImage, 10, 10, RENDER);
    }
    else
        printf ("Failed to load images. Error reported %d \n", -ErrRep);

    ndDelay (30);
}
```

The file doesn’t exist. Nd signals it returning an error code:



```
Failed to load images. Error reported 29
```

ErrorCode -29 means (see *<ndenv path>\PSP\SDK\Nanodesktop\src\$$Errors.h*) ERR_FILE_NOT_FOUND

There is another problem. If you try to load a *non-bitmap* image (such as .jpg or .png), you will have an error code: ERR_FORMAT_NOT_SUPPORTED (-30). This happens because the DEVIL support is still disabled.

Enable Dev-IL support

Nanodesktop supports a special version of the Dev-IL libraries by Delton Woods, that allow you to open/save different image formats with no problem.

The formats supported are: BMP, GIF, JPG, PNG, PNM, PGM, PSD, PCX, ICO, TGA, SGI, TIF, XPM.

The code that is needed for these operations is very large, so there are two versions of nd: the former doesn't support ndDevIL interface, and uses only internal routines to load and save bitmap images (and *only* bitmap images). This version of Nanodesktop should be used only when you want to create homebrews that are small, and if you don't need sophisticated image support.

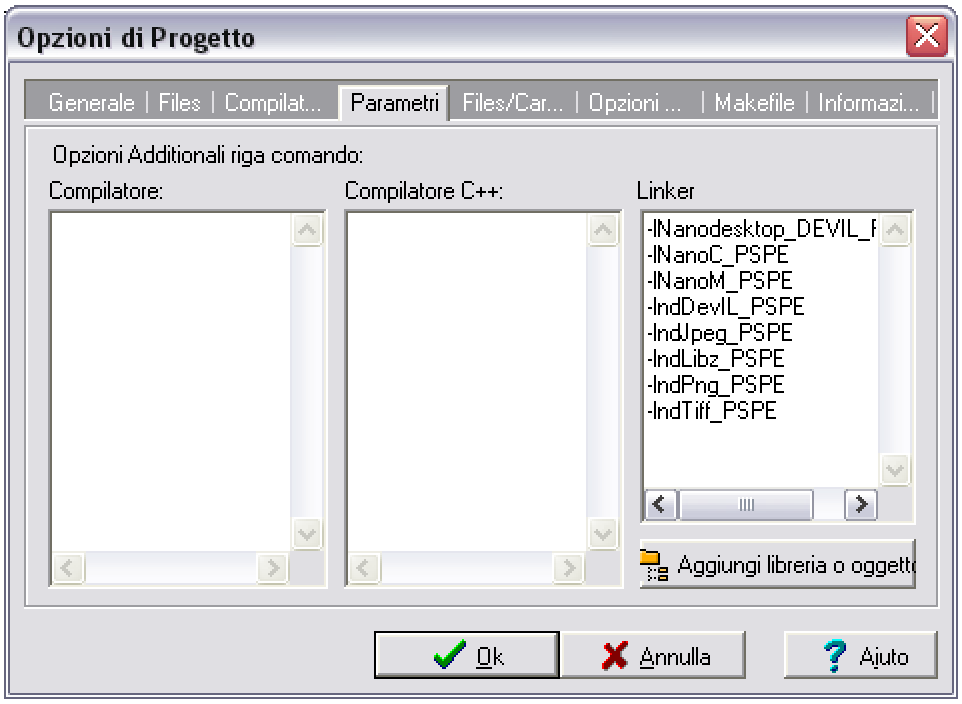
The latter, uses the ndDevIL interface and links other libraries, and should be used when you want larger homebrew, but with the capability to open different images formats.

If you want to enable ndDevIL support, do the following:

a) change the name of the library that is linked in Dev-CPP (replace **Nanodesktop_PSPE** with **Nanodesktop_DEVIL_PSPE** and replace **Nanodesktop_PSP** with **Nanodesktop_DEVIL_PSP**).

b) add to the list of the libraries that will be linked by psp-ld the following:

   for PSPE: **-lndDevIL_PSPE -lndJpeg_PSPE -lndLibz_PSPE -lndPng_PSPE -lndTiff_PSPE**
   for PSP: **-lndDevIL_PSP -lndJpeg_PSP -lndLibz_PSP -lndPng_PSP -lndTiff_PSP**


Opzioni di Progetto
Generale | Files | Compilat... | Parametri | Files/Car... | Opzioni ... | Makefile | Informazi...
Opzioni Additionali riga comando:
Compilatore:
Compilatore C++:
Linker
Aggiungi libreria o oggett
Ok
Annulla
Aiuto

```
-lNanodesktop_DEVIL_P
-lNanoC_PSPE
-lNanoM_PSPE
-lndDevIL_PSPE
-lndJpeg_PSPE
-lndLibz_PSPE
-lndPng_PSPE
-lndTiff_PSPE
```

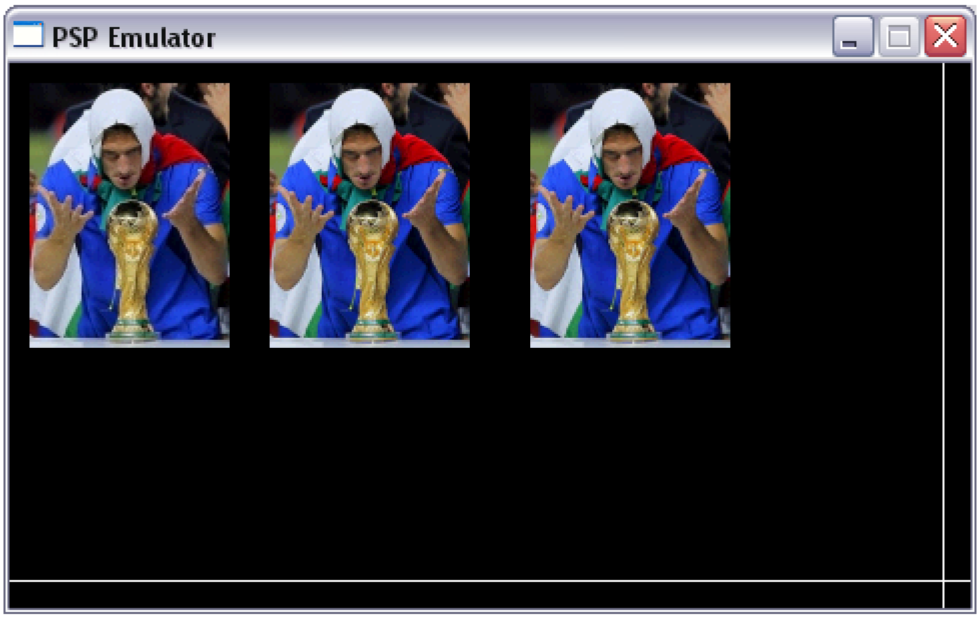
Be careful that –lNanodesktop_DEVIL_PSP means –ELLENanodesktop_DEVIL_PSP: the first char of the items is an –elle and not an –i)

Now Nanodesktop will open all image formats. Try this source code:

```
#include <nanodesktop.h>

int ndMain (void)
{
    struct ndImage_Type MyImage;
    char ErrRep;

    ndInitSystem ();

    ndIMG_InitImage (&MyImage);

    // Image 0

    ErrRep=ndIMG_LoadImage (&MyImage, "ms0:/prova0.jpg", NDMGKNB);

    if (!ErrRep)
       ndIMG_ShowImage_InBS (&MyImage, 10, 10, RENDER);
    else
        printf ("Failed to load image 0. Error reported %d \n", -ErrRep);

    // Image 1

    ErrRep=ndIMG_LoadImage (&MyImage, "ms0:/prova1.png", NDMGKNB);

    if (!ErrRep)
       ndIMG_ShowImage_InBS (&MyImage, 130, 10, RENDER);
    else
        printf ("Failed to load image 1. Error reported %d \n", -ErrRep);

    // Image 2

    ErrRep=ndIMG_LoadImage (&MyImage, "ms0:/prova2.tif", NDMGKNB);

    if (!ErrRep)
       ndIMG_ShowImage_InBS (&MyImage, 260, 10, RENDER);
    else
        printf ("Failed to load image 2. Error reported %d \n", -ErrRep);

    // End

    ndDelay (30);
}
```

Note: if you load an image in a struct ndImage_Type that has been associated with another image beforehand, the previous data relative to the previous image will be deallocated automatically.

The result of the previous code is this:



Loading images from a NDF array

Nanodesktop can also load an image from an NDF array. *An NDF array (Nanodesktop data format),* is simply an array that contains information about an image, declaring color pixel by pixel.

The values are always in RGB555 format. In *<ndenv>\PSP\SDK\Tools\M1pspc,* there is a utility that allows you to convert a .bmp image into .c source code that defines an array. This array must be converted into a Nanodesktop image, handled by a struct ndImage_Type.

The conversion can be executed using this routine

```
char ndIMG_LoadImageFromNDFArray (struct ndImage_Type *ndImgDest, int LenX, int LenY, short unsigned
int *PointerToFirstWord, char ColorFormat);
```

The first parameter is a pointer to a struct ndImage_Type that will contain image information, LenX and LenY are the width and the height of the image in pixels, the following parameter is a pointer to the first 16-bit word in the array NDF, and the char ColorFormat is the format that will be adopted in the target image (NDMGKNB or NDRGB).

Free memory

When you have used an image, you can free the memory that was utilized by the image, using the routine **ndIMG_ReleaseImage**

```
void ndIMG_ReleaseImage (struct ndImage_Type *MyImage);
```

Modify images

When an image has been assigned to a struct ndImage_Type, the user can modify the image in memory. This can be useful, for example, before of saving the modified image to disk.

For this task, nd provides the functions **ndIMG_GetPixelFromImage**, **ndIMG_GetPixelFromImage_RGB**, **ndIMG_PutPixelToImage**, **ndIMG_PutPixelToImage_RGB**.

Here, we report the prototypes of these functions:

```
char ndIMG_GetPixelFromImage (struct ndImage_Type *MyImage, short unsigned int PosX,
short unsigned int PosY, TypeColor *MagicNumber);
```

```
char ndIMG_GetPixelFromImage_RGB (struct ndImage_Type *MyImage, short unsigned int PosX,
short unsigned int PosY, unsigned char *ChannelR, unsigned char *ChannelG, unsigned char
*ChannelB);
```

```
char ndIMG_PutPixelToImage (struct ndImage_Type *MyImage, short unsigned int PosX, short
unsigned int PosY, TypeColor MagicNumber);
```

```
char ndIMG_PutPixelToImage_RGB (struct ndImage_Type *MyImage, short unsigned int PosX,
short unsigned int PosY, unsigned char ChannelR, unsigned char ChannelG, unsigned char
ChannelB);
```

The first and second functions allow you to read the content of each pixel in the image. The functions return the information as a MagicNumber, or in the form of R-G-B values.

The third and fourth functions allow you to write a single pixel to an image. Note that the user can write complex routines that use ndIMG_PutPixelToImage, obtaining the possibility to apply filters or transformation to the image (functions of this type, aren't included in Nanodesktop, but are available under the Nanodesktop environment, using ndOpenCV capabilities. ndOpenCV is a ported version of Intel's OpenCV and it is included in the Nanodesktop distribution).

Create a blank image

You can also create a blank image, and fill it with the pixels that you want (using ndIMG_PutPixelToImage). The function that allows you to do this is called **ndIMG_CreateImage**

```
char ndIMG_CreateImage (struct ndImage_Type *ndImgDest, int LenX, int LenY, char Color-
Format);
```

The user has only to specify image dimensions and color format.

Saving images

You can save an image from ram to disk using the function:

```
char ndIMG_SaveImage (struct ndImage_Type *MyImage, char *NameFile);
```

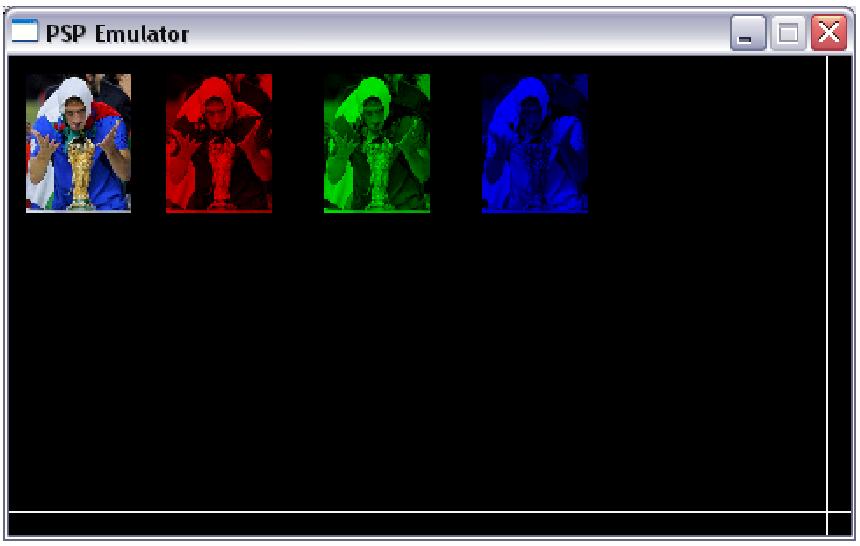
It returns 0 if there are no errors and <0 if there are errors. The extension NameFile indicates to nd the image destination and format that you wish to use.

An example

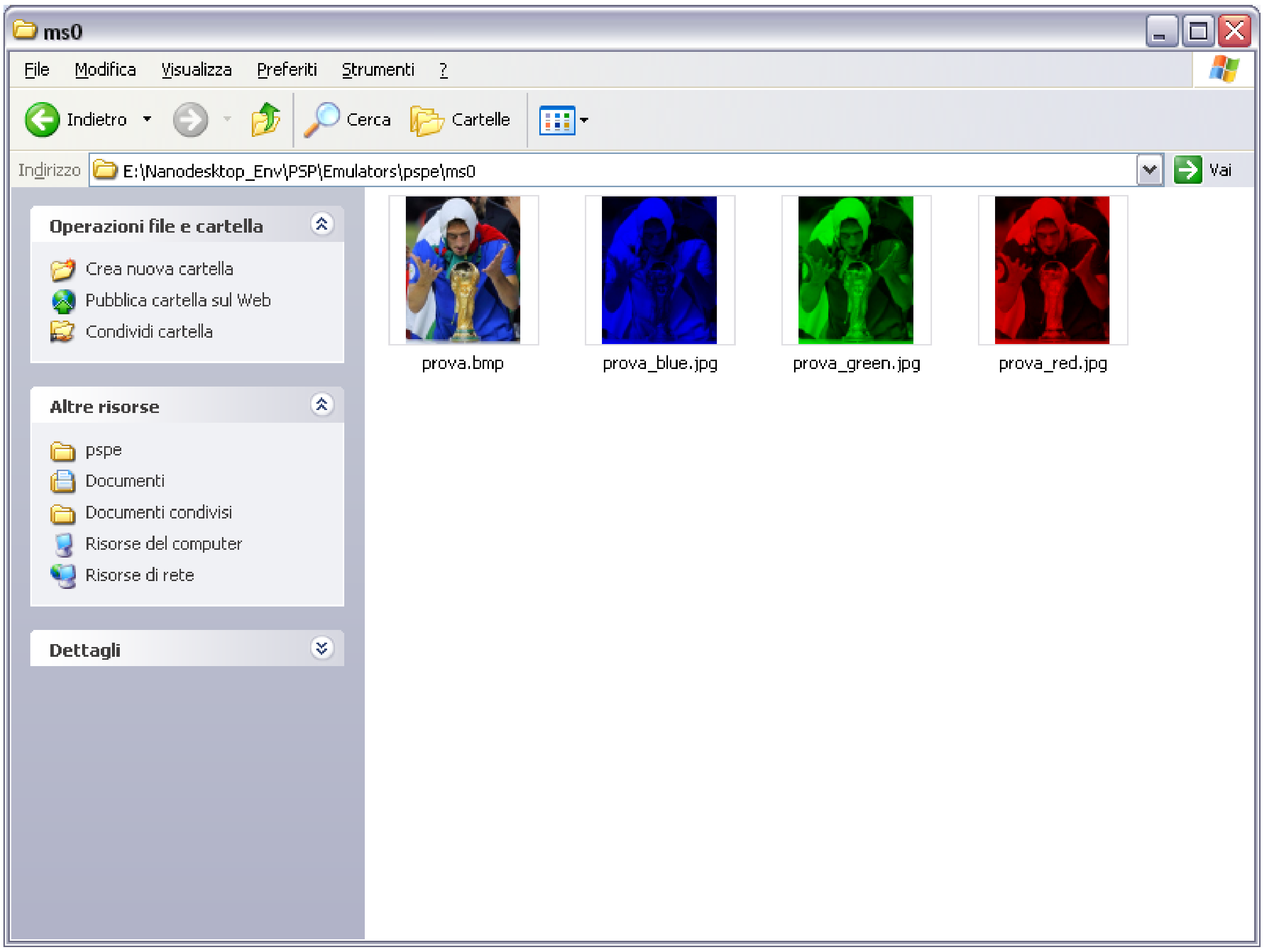
This is a short example using the previous routines. This program loads an image from disk, splits it into 3 images, one per channel, shows them and finally saves the three images to disk.

```
#include <nanodesktop.h>

int ndMain (void)
{
    struct ndImage_Type MyImage, MyImageRed, MyImageGreen, MyImageBlue;
    char ErrRep;

    ndInitSystem ();

    ndIMG_InitImage (&MyImage);

    ErrRep=ndIMG_LoadImage (&MyImage, "ms0:/prova.bmp", NDRGB);
    if (!ErrRep)
    {
        // It creates 3 void images, of identical dims to source image

        ndIMG_CreateImage (&MyImageRed,   MyImage.LenX, MyImage.LenY, NDRGB);
        ndIMG_CreateImage (&MyImageGreen, MyImage.LenX, MyImage.LenY, NDRGB);
        ndIMG_CreateImage (&MyImageBlue,  MyImage.LenX, MyImage.LenY, NDRGB);

        int  CounterX, CounterY;
        unsigned char Red, Green, Blue;

        for (CounterY=0; CounterY<MyImage.LenY; CounterY++)
        {
            for (CounterX=0; CounterX<MyImage.LenX; CounterX++)
            {
                // Extract channel informations from source image (MyImage)

                ndIMG_GetPixelFromImage_RGB (&MyImage, CounterX, CounterY, &Red, &Green, &Blue);

                // Put the informations to destination images

                ndIMG_PutPixelToImage_RGB (&MyImageRed,   CounterX, CounterY, Red, 0, 0);
                ndIMG_PutPixelToImage_RGB (&MyImageGreen, CounterX, CounterY, 0, Green, 0);
                ndIMG_PutPixelToImage_RGB (&MyImageBlue,  CounterX, CounterY, 0, 0, Blue);

            }
        }

       // Show source image

        ndIMG_ShowImageScaled_InBS (&MyImage, 10, 10, 0.3, 0.3, RENDER);

       // Show destination images

       ndIMG_ShowImageScaled_InBS (&MyImageRed,   90,  10, 0.3, 0.3, RENDER);
       ndIMG_ShowImageScaled_InBS (&MyImageGreen, 180, 10, 0.3, 0.3, RENDER);
       ndIMG_ShowImageScaled_InBS (&MyImageBlue,  270, 10, 0.3, 0.3, RENDER);

       // Save the images on memory stick

       ndIMG_SaveImage (&MyImageRed,   "ms0:/prova_red.jpg");
       ndIMG_SaveImage (&MyImageGreen, "ms0:/prova_green.jpg");
       ndIMG_SaveImage (&MyImageBlue,  "ms0:/prova_blue.jpg");

       // Free memory

       ndIMG_ReleaseImage (&MyImage);
       ndIMG_ReleaseImage (&MyImageRed);
       ndIMG_ReleaseImage (&MyImageGreen);
       ndIMG_ReleaseImage (&MyImageBlue);

    }
    else
        printf ("Failed to load image 0. Error reported %d \n", -ErrRep);


    ndDelay (30);
}
```

Here is the result on PSPE:



And this is the result stored on the memory stick (i.e. in PSPE emulated memory stick):



E:\Nanodesktop_Env\PSP\Emulators\pspe\ms0

prova.bmp
prova_blue.jpg
prova_green.jpg
prova_red.jpg

Chapter 10
Manage the windows

In the previous chapters, we have seen how to use BS space to create new graphical elements or to visualize a new image. Now we'll explain how to create a new window and how to manage it

ndLP_CreateWindow

Nanodesktop supports, for windows creation, the following routines:

```
ndLP_AllocateWindow
ndLP_AllocateWindow_MENU
ndLP_AllocateWindow_NORESIZE
```

and

```
ndLP_CreateWindow
```

As in Nanodesktop 0.3 the function we recommend you use is *ndLP_CreateWindow* (the previous functions AllocateWindow... come from nd 0.2 and though they are still supported in nd 0.3 this is only for backwards compatibility reasons), we'll refer, in the rest of this chapter, only to ndLP_CreateWindow.

The prototype of ndLP_CreateWindow is the following:

```
char ndLP_CreateWindow (unsigned short int _PosWindowX1, unsigned short int _PosWindowY1,
                        unsigned short int _PosWindowX2, unsigned short int _PosWindowY2,
                        char *_WindowTitle,
                        TypeColor _ColorTitle, TypeColor _ColorBGTitle,
                        TypeColor _ColorWindow, TypeColor _ColorBorderWindow,
                        ndint64 Attribute);
```

The first four parameters are the positions X1,Y1,X2,Y2 of the rectangle that will contain the window. The following parameter is the title of the window, followed by the colors of the title, of the title bar, of the window and of the border of the window.

The last parameter is very important: it is a 64-bit value that specifies the features of the window. Using the correct 64 bit keys, you can set, in a very powerful way, the behaviour of the window.

See this program: this is the simplest way to use ndLP_CreateWindow:

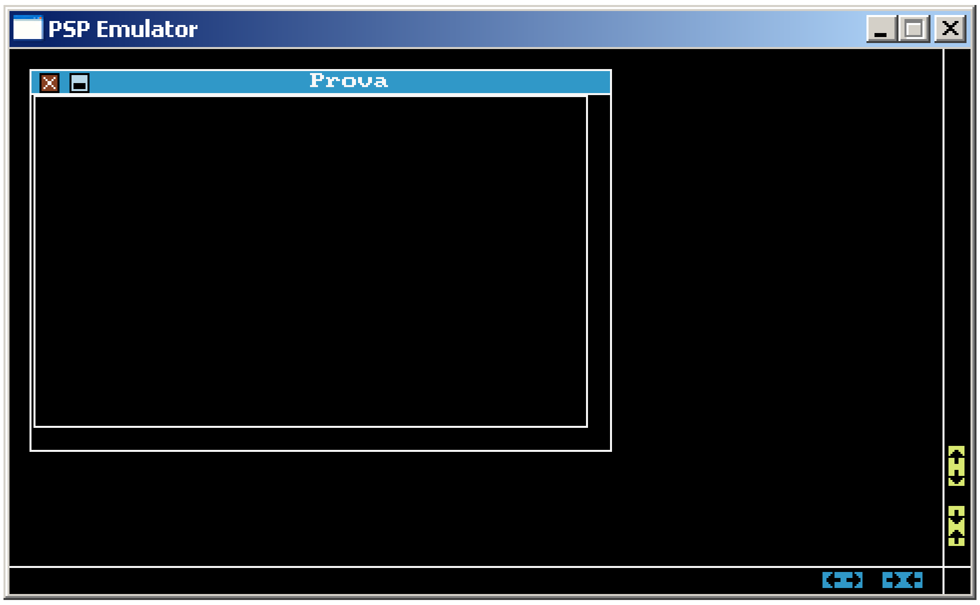
```
#include <nanodesktop.h>

int WndHandle;

int ndMain()
{
    int Counter;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (10, 10, 300, 200, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                 COLOR_BLACK, COLOR_WHITE, 0);

    if (WndHandle>=0)
    {
       ndLP_MaximizeWindow (WndHandle);

       // Put here the other commands
    }
}
```

The sequence you see here is extremely common in Nanodesktop code.

ndLP_CreateWindow returns the handle of the window. All operations that will be executed in this window in the future will have to contain this handle number.

The value is stored in WndHandle (this variable must be declared char or int). If the returned value is not negative (if it is negative, it means that there is an error in windows allocation), the program provides, as its first operation, to maximize the window (when a window has been created, it is initially minimized, so the source must expressly provide its maximized state).

Result:



ndLP_MaximizeWindow, ndLP_MinimizeWindow

These two routines are provided to maximize and minimize the window.

ndLP_PutWindowInFirstPlane

This function puts a window in the first plane

ndLP_MoveWindow

This function moves a window to a new position. The routine has the following prototype:

```
char ndLP_MoveWindow (short unsigned int NewPosX, short unsigned int NewPosY,
                      unsigned char WndHandle);
```

What is the window space ?

If you want to understand the following part of this chapter, you'll have to focus your attention on the concept of window space (WS).

Under Nanodesktop, each window is assigned a space that can be used for writing text or for showing images or other graphical elements. The important thing is that the dimensions of the window are not necessarily the same as its window space.

In fact, a window can show only a small part of its real window space. A typical case is when you reduce the width or the height of a window. The window dimensions have been reduced, but the content of the window is maintained.

The dimensions of a window are called WndLength and WndHeight. This dimensions can be changed by functions such as *ndLP_ResizeWindow*.

The dimensions of the window space are called WSWndLength and WSWndHeight: they are assigned at window creation and they cannot be changed until the window has been destroyed.

When you call *ndLP_CreateWindow* using 0 as an attribute parameter, nd automatically calculates the correct size of the window space. The dimensions of the WS are, in this case, almost the same as the window (not identical, because the window must also contain the space for scroll-bars: the difference on X and Y axis is called window overhead).

The user can expressly set the dimensions of the window space: in this way, you can obtain a window in which the window space is larger than the window dimensions.

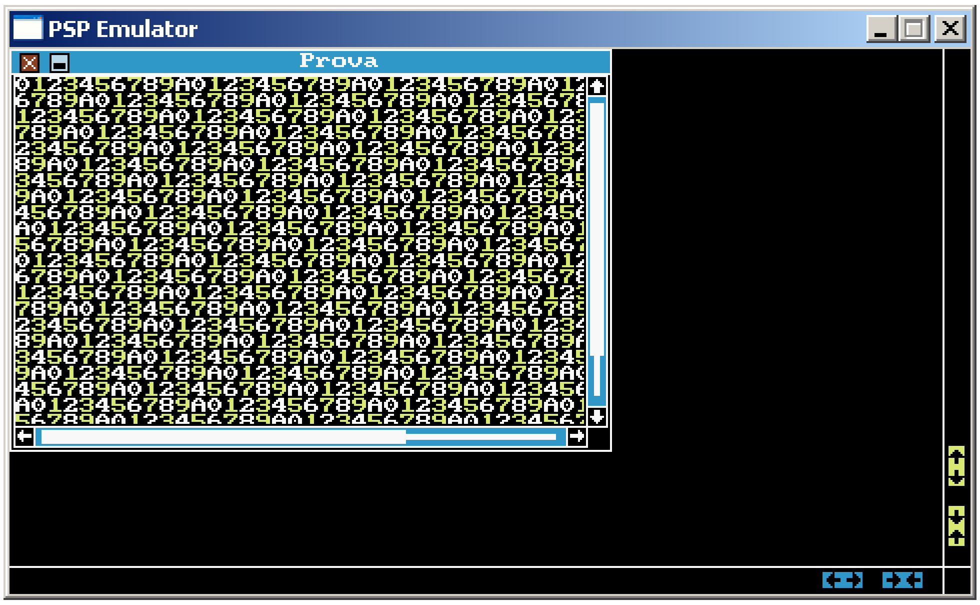
Let's see this source:

```
#include <nanodesktop.h>

int WndHandle;

int ndMain()
{
    int Counter;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (0, 0, 300, 200, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                COLOR_BLACK, COLOR_WHITE,
                                NDKEY_SETWSLENGTH (400) | NDKEY_SETWSHEIGHT (200));

    if (WndHandle>=0)
    {
       ndLP_MaximizeWindow (WndHandle);

       int Counter;

       for (Counter=0; Counter<100; Counter++)
       {
           ndWS_Write ("0", COLOR_WHITE,  WndHandle, NORENDER);
           ndWS_Write ("1", COLOR_YELLOW, WndHandle, NORENDER);
           ndWS_Write ("2", COLOR_WHITE,  WndHandle, NORENDER);
           ndWS_Write ("3", COLOR_YELLOW, WndHandle, NORENDER);
           ndWS_Write ("4", COLOR_WHITE,  WndHandle, NORENDER);
           ndWS_Write ("5", COLOR_YELLOW, WndHandle, NORENDER);
           ndWS_Write ("6", COLOR_WHITE,  WndHandle, NORENDER);
           ndWS_Write ("7", COLOR_YELLOW, WndHandle, NORENDER);
           ndWS_Write ("8", COLOR_WHITE,  WndHandle, NORENDER);
           ndWS_Write ("9", COLOR_YELLOW, WndHandle, NORENDER);
           ndWS_Write ("A", COLOR_WHITE,  WndHandle, NORENDER);

           XWindowRender (WndHandle);
       }

       for (Counter=0; Counter<=100; Counter+=10)
       {
           ndLP_ChangeWindowROI (Counter, 0, ROI_POSX | ROI_FRAC, WndHandle);
       }
    }
}
```

This code shows some interesting things.

The dimensions of the window space can be set using the 64-bit keys (NDKEY_SETWSLENGTH and NDKEY_SETWSHEIGHT). The value must be set between 0 and 2048. In this example, the window has dimensions of (X 300 Y 200). Window space has dimensions of (X 400 Y 200).

Note that the width of the window space is greater than the width of the window. You can observe this, looking at the result using the PSPE emulator. The code will visualize the sequence (0123456789A), repeatedly.

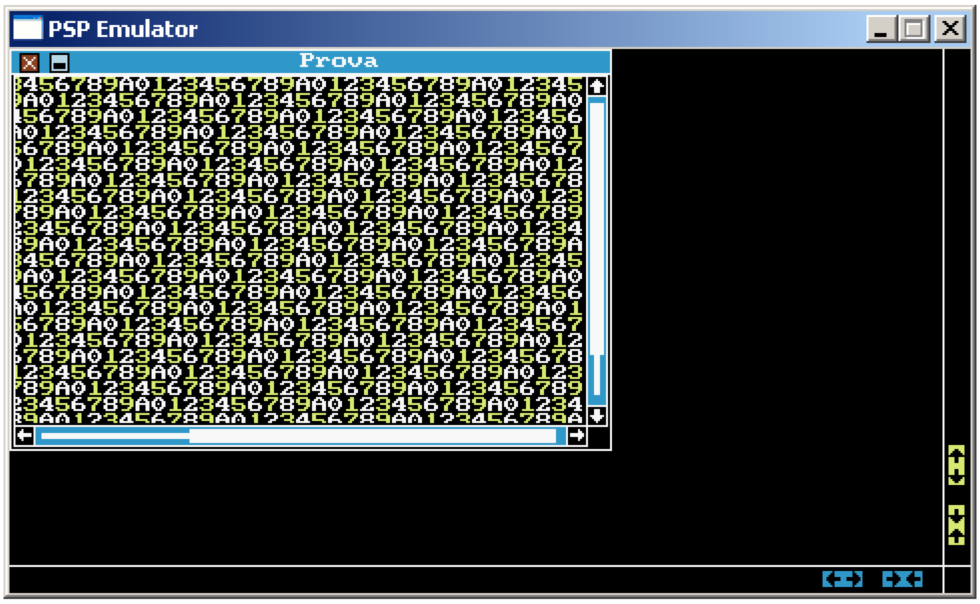


```
0123456789A0123456789A0123456789A01
6789A0123456789A0123456789A01234567
123456789A0123456789A0123456789A012
789A0123456789A0123456789A012345678
23456789A0123456789A0123456789A0123
89A0123456789A0123456789A0123456789
3456789A0123456789A0123456789A01234
9A0123456789A0123456789A0123456789A
456789A0123456789A0123456789A012345
A0123456789A0123456789A0123456789A0
56789A0123456789A0123456789A0123456
0123456789A0123456789A0123456789A01
6789A0123456789A0123456789A01234567
123456789A0123456789A0123456789A012
789A0123456789A0123456789A012345678
23456789A0123456789A0123456789A0123
89A0123456789A0123456789A0123456789
3456789A0123456789A0123456789A01234
9A0123456789A0123456789A0123456789A
456789A0123456789A0123456789A012345
A0123456789A0123456789A0123456789A0
```

And this is what the source does: but ndWS_Write writes in the window space, so part of the sequence is in a hidden area of the window space that it isn't visualized at this moment. The part of the WS space that is, in any specific moment, visualized in the area of a window, is called ROI (Region of Interest).

The routine XWindowRender (WndHandle) plays the same role as BaseScreenRender in BS space. It executes the rendering of the contents of a window. So the user will see the entire sequence 123456789A appear instantaneosly.

The last part of the code "moves" the position of the ROI for the window. This operation allows the user to see the entire window space: the user usually does this using the scroll bars, but it can be done by the programmer using the function *ndWS_ChangeWindowROI*.



```
456789A0123456789A0123456789A012345
A0123456789A0123456789A0123456789A0
56789A0123456789A0123456789A0123456
0123456789A0123456789A0123456789A01
6789A0123456789A0123456789A01234567
123456789A0123456789A0123456789A012
789A0123456789A0123456789A012345678
23456789A0123456789A0123456789A0123
89A0123456789A0123456789A0123456789
3456789A0123456789A0123456789A01234
9A0123456789A0123456789A0123456789A
456789A0123456789A0123456789A012345
A0123456789A0123456789A0123456789A0
56789A0123456789A0123456789A0123456
0123456789A0123456789A0123456789A01
6789A0123456789A0123456789A01234567
123456789A0123456789A0123456789A012
789A0123456789A0123456789A012345678
23456789A0123456789A0123456789A0123
89A0123456789A0123456789A0123456789
3456789A0123456789A0123456789A01234
```

When the first parameter is set to 0, the X-scroll is moved to the left. When the first parameter is set to 100, the X-scroll is moved to the right. The second parameter has the same effect, but it is referred to as the Y scroll (up/down). The real behaviour of the routine is set by the third parameter: the two explicit constants ROI_POSX and ROI_FRAC say that the routine here operates only on the x-coordinate (the second parameter has been put to zero because it is ignored here), and that the values are expressed in fractional numbers (0 indicates scroller at left/up, and 100 indicates scroller at right/down).

Remember that at any moment you can obtain the actual position of the ROI, using the routine *ndLP_GetWindowROIPosX* or the routine *ndLP_GetWindowROIPosY*

```
float ndLP_GetWindowROIPosX (char Frac, char WndHandle);
float ndLP_GetWindowROIPosY (char Frac, char WndHandle);
```

ndLP_ResizeWindow

The dimensions of WS space is determined at the time of the window creation/allocation and they cannot be modified. The dimensions of the window on screen can be modified by the mouse or using the routine *ndLP_ResizeWindow*

```
char ndLP_ResizeWindow (short unsigned int NewSizeX, short unsigned int NewSizeY, unsigned char
WndHandle);
```

This routine allows you to set the new dimensions of the window (dimensions of the window space will remain untouched)

Fonts

Now we'll see how to set the correct fonts for the title and for the chars in char map.

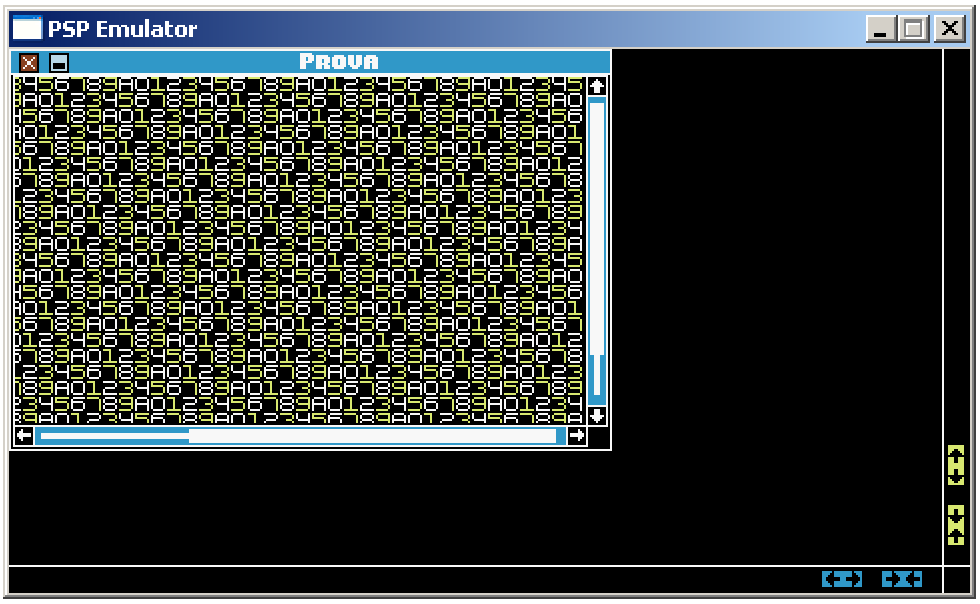
There are two ways. The former is to use an opportune 64-bit key in window creation.

Nanodesktop provides, for this task, the routines NDKEY_SETFONT (x) and NDKEY_SETTITLEFONT (x).

This is an example

```
#include <nanodesktop.h>

int WndHandle;

int ndMain()
{
    int Counter;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (0, 0, 300, 200, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                COLOR_BLACK, COLOR_WHITE,
                                NDKEY_SETWSLENGTH (400) | NDKEY_SETWSHEIGHT (200) |
                                NDKEY_SETFONT (3) | NDKEY_SETTITLEFONT (4) );

    if (WndHandle>=0)
    {
       ndLP_MaximizeWindow (WndHandle);

       int Counter;

       for (Counter=0; Counter<100; Counter++)
       {
           ndWS_Write ("0", COLOR_WHITE,  WndHandle, NORENDER);
           ndWS_Write ("1", COLOR_YELLOW, WndHandle, NORENDER);
           ndWS_Write ("2", COLOR_WHITE,  WndHandle, NORENDER);
           ndWS_Write ("3", COLOR_YELLOW, WndHandle, NORENDER);
           ndWS_Write ("4", COLOR_WHITE,  WndHandle, NORENDER);
           ndWS_Write ("5", COLOR_YELLOW, WndHandle, NORENDER);
           ndWS_Write ("6", COLOR_WHITE,  WndHandle, NORENDER);
           ndWS_Write ("7", COLOR_YELLOW, WndHandle, NORENDER);
           ndWS_Write ("8", COLOR_WHITE,  WndHandle, NORENDER);
           ndWS_Write ("9", COLOR_YELLOW, WndHandle, NORENDER);
           ndWS_Write ("A", COLOR_WHITE,  WndHandle, NORENDER);

           XWindowRender (WndHandle);
       }

       for (Counter=0; Counter<=100; Counter+=10)
       {
           ndLP_ChangeWindowROI (Counter, 0, ROI_POSX | ROI_FRAC, WndHandle);
       }
    }
}
```

As you can see, the fonts have been set in window allocation:



The second method is the following: you can use the routines *ndWS_SetFont, ndLP_SetFont, ndLP_ChangeWindowTitle*.

If you want to change the font used for the char map, you have to use ndWS_SetFont.

```
char ndWS_SetFont (unsigned char FntHandle, char WndHandle);
```

If you want to change the font used in the title, you have to use ndLP_SetFont, followed by ndLP_ChangeWndTitle:

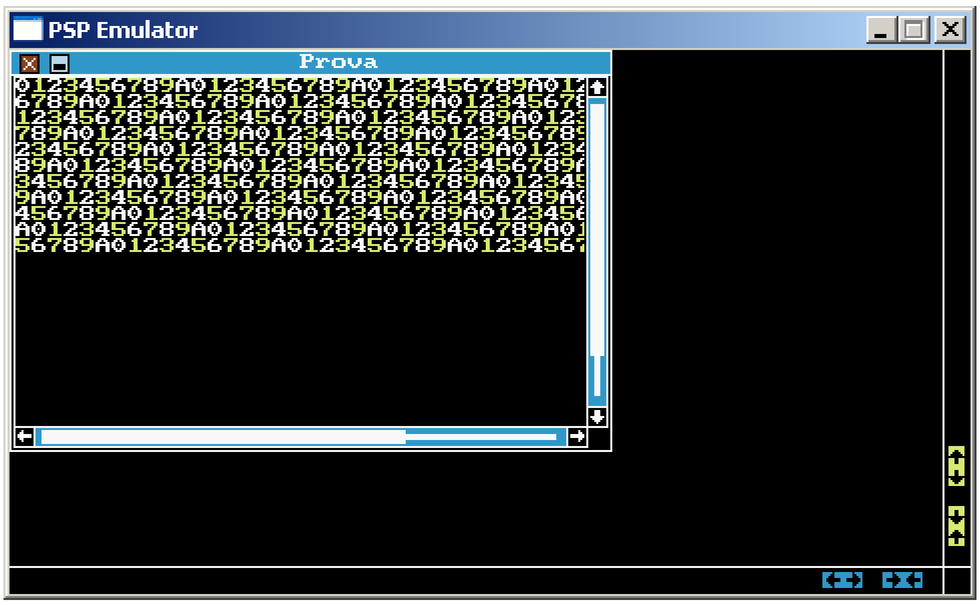
```
char  ndLP_ChangeWndTitle  (char  *NewTitle,  TypeColor  NewColorTitle,  TypeColor  NewBGColorTitle,
unsigned char WndHandle);
```

You can see the effect with this source:

```
#include <nanodesktop.h>

int WndHandle;

int ndMain()
{
    int Counter;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (0, 0, 300, 200, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                COLOR_BLACK, COLOR_WHITE,
                                NDKEY_SETWSLENGTH (400) | NDKEY_SETWSHEIGHT (200) );

    if (WndHandle>=0)
    {
       ndLP_MaximizeWindow (WndHandle);

       int Counter;

       for (Counter=0; Counter<100; Counter++)
       {
           ndWS_Write ("0", COLOR_WHITE,  WndHandle, NORENDER);
           ndWS_Write ("1", COLOR_YELLOW, WndHandle, NORENDER);
           ndWS_Write ("2", COLOR_WHITE,  WndHandle, NORENDER);
           ndWS_Write ("3", COLOR_YELLOW, WndHandle, NORENDER);
           ndWS_Write ("4", COLOR_WHITE,  WndHandle, NORENDER);
           ndWS_Write ("5", COLOR_YELLOW, WndHandle, NORENDER);
```

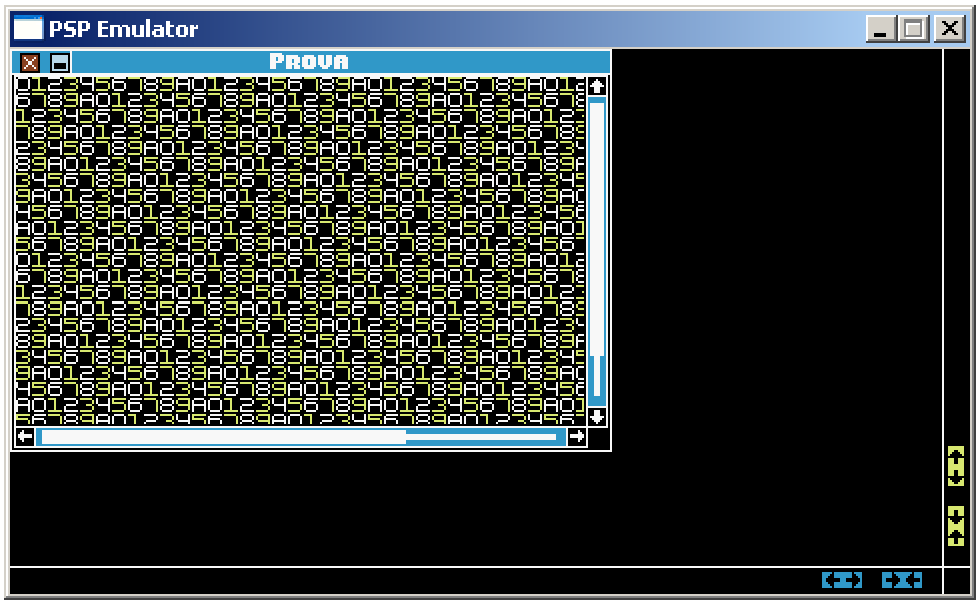
```
            ndWS_Write ("6", COLOR_WHITE,  WndHandle, NORENDER);
            ndWS_Write ("7", COLOR_YELLOW, WndHandle, NORENDER);
            ndWS_Write ("8", COLOR_WHITE,  WndHandle, NORENDER);
            ndWS_Write ("9", COLOR_YELLOW, WndHandle, NORENDER);
            ndWS_Write ("A", COLOR_WHITE,  WndHandle, NORENDER);

            XWindowRender (WndHandle);
        }

        ndDelay (4);

        ndWS_SetFont (3, WndHandle);

        ndLP_SetFont (4, WndHandle);
        ndLP_ChangeWndTitle ("Prova", COLOR_WHITE, COLOR_LBLUE, WndHandle);
    }
}
```

And this is the result:



```
0123456789A0123456789A0123456789A012
6789A0123456789A0123456789A01234567
123456789A0123456789A0123456789A012
789A0123456789A0123456789A012345678
23456789A0123456789A0123456789A0123
89A0123456789A0123456789A0123456789
3456789A0123456789A0123456789A01234
9A0123456789A0123456789A0123456789A
456789A0123456789A0123456789A012345
A0123456789A0123456789A0123456789A0
56789A0123456789A0123456789A0123456
```

Before



```
0123456789A0123456789A0123456789A012
6789A0123456789A0123456789A01234567
123456789A0123456789A0123456789A012
789A0123456789A0123456789A012345678
23456789A0123456789A0123456789A0123
89A0123456789A0123456789A0123456789
3456789A0123456789A0123456789A01234
9A0123456789A0123456789A0123456789A
456789A0123456789A0123456789A012345
A0123456789A0123456789A0123456789A0
56789A0123456789A0123456789A0123456
0123456789A0123456789A0123456789A012
6789A0123456789A0123456789A01234567
123456789A0123456789A0123456789A012
789A0123456789A0123456789A012345678
23456789A0123456789A0123456789A0123
89A0123456789A0123456789A0123456789
3456789A0123456789A0123456789A01234
9A0123456789A0123456789A0123456789A
456789A0123456789A0123456789A012345
A0123456789A0123456789A0123456789A0
```

After

Non-resizeable windows

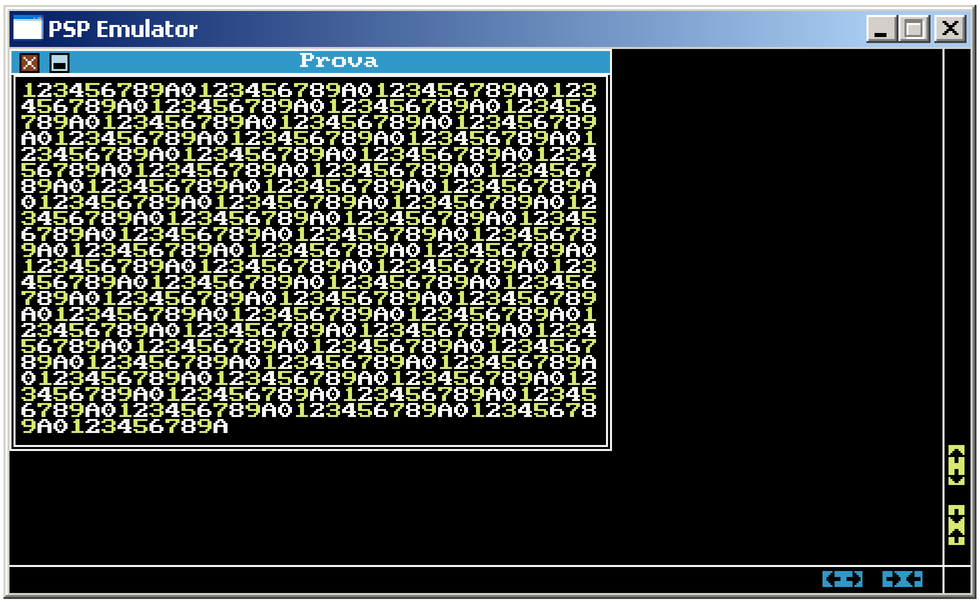
Under Nanodesktop you can also create windows that are not resizeable. These windows do not have the window overhead: so the space is better used.

You can create a non-resizeable window, using a particular option in ndLP_CreateWindow.

See this source:

```
#include <nanodesktop.h>

int WndHandle;

int ndMain()
{
    int Counter;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (0, 0, 300, 200, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                COLOR_BLACK, COLOR_WHITE,
                                NO_RESIZEABLE );

    if (WndHandle>=0)
    {
       ndLP_MaximizeWindow (WndHandle);

       int Counter;

       for (Counter=0; Counter<100; Counter++)
       {
           ndWS_Write ("0", COLOR_WHITE,  WndHandle, NORENDER);
           ndWS_Write ("1", COLOR_YELLOW, WndHandle, NORENDER);
           ndWS_Write ("2", COLOR_WHITE,  WndHandle, NORENDER);
           ndWS_Write ("3", COLOR_YELLOW, WndHandle, NORENDER);
           ndWS_Write ("4", COLOR_WHITE,  WndHandle, NORENDER);
           ndWS_Write ("5", COLOR_YELLOW, WndHandle, NORENDER);
           ndWS_Write ("6", COLOR_WHITE,  WndHandle, NORENDER);
           ndWS_Write ("7", COLOR_YELLOW, WndHandle, NORENDER);
           ndWS_Write ("8", COLOR_WHITE,  WndHandle, NORENDER);
           ndWS_Write ("9", COLOR_YELLOW, WndHandle, NORENDER);
           ndWS_Write ("A", COLOR_WHITE,  WndHandle, NORENDER);

           XWindowRender (WndHandle);
       }


    }
}
```

The attribute NO_RESIZEABLE indicates that we want a non-resizeable window. Note that if you choose this option, nd will ignore any option about WSWndLength or WSWndHeight. The dimensions of WS space will be calculated automatically



```
123456789A0123456789A0123456789A0123
456789A0123456789A0123456789A0123456
789A0123456789A0123456789A0123456789
A0123456789A0123456789A0123456789A01
23456789A0123456789A0123456789A01234
56789A0123456789A0123456789A01234567
89A0123456789A0123456789A0123456789A
0123456789A0123456789A0123456789A012
3456789A0123456789A0123456789A012345
6789A0123456789A0123456789A012345678
9A0123456789A0123456789A0123456789A0
123456789A0123456789A0123456789A0123
456789A0123456789A0123456789A0123456
789A0123456789A0123456789A0123456789
A0123456789A0123456789A0123456789A01
23456789A0123456789A0123456789A01234
56789A0123456789A0123456789A01234567
89A0123456789A0123456789A0123456789A
0123456789A0123456789A0123456789A012
3456789A0123456789A0123456789A012345
6789A0123456789A0123456789A012345678
9A0123456789A
```

ndLP_DestroyWindow

The windows that have been created can be destroyed using the routine ndLP_DestroyWindow.

```
char ndLP_DestroyWindow (unsigned char WndHandle);
```

ndLP_EnableTrasparency, ndLP_DisableTrasparency

This functions enable/disable the trasparency mode.

```
void ndLP_EnableTrasparency (void);
void ndLP_DisableTrasparency (void);
```

ndLP_LoadWallPaperFromFile, ndLP_LoadWallPaperFromNDImage

This function loads a wallpaper from a file on disk or from a struct ndImage_Type (see chapter 9):

```
char ndLP_LoadWallPaperFromFile (char *NameFile);
char ndLP_LoadWallPaperFromNDImage (struct ndImage_Type *MyImage);
```

ndLP_EnableWallpaper, ndLP_DisableWallpaper

This functions enable/disable the wallpaper.

```
void ndLP_EnableWallPaper (void);
void ndLP_DisableWallPaper (void);
```

Chapter 11

Draw in the window space (WS)

In the previous chapter we have learnt how to create, to manage and to destroy windows on the screen.

Now, we'll see how to draw in the window space. The first concept to understand is that for each drawing function that we have seen in the basescreen chapter there is an equivalent that draws or operates in window space.

The prototypes often are similar, except for the fact that ndWS drawing routines request the WndHandle number in order to specify in which window you want to operate.

This table shows the correspondence between basescreen routines that we have seen in chapters 8-9, and the window space routines

ndBS_GetPixel	ndWS_GetPixel
ndBS_PutPixel	ndWS_PutPixel
ndBS_DrawLine	ndWS_DrawLine
ndBS_DrawPoly	ndWS_DrawPoly
ndBS_FillArea	ndWS_FillArea
ndBS_DrawRectangle	ndWS_DrawRectangle
ndBS_DrawSpRectangle	ndWS_DrawSpRectangle
ndBS_DrawRtRectangle	ndWS_DrawRtRectangle
ndBS_DrawArcOfEllipse	ndWS_DrawArcOfEllipse
ndBS_DrawEllipse	ndWS_DrawEllipse
ndBS_DrawArcOfCircle	ndWS_DrawArcOfCircle
ndBS_DrawCircle	ndWS_DrawCircle
ndBS_ClrBaseScreen	ndWS_ClrALL
-	ndWS_ClrBufferSpace
ndBS_GrWriteChar	ndWS_GrWriteChar
ndBS_GrWriteLn	ndWS_GrWriteLn
ndBS_GrPrintLn	ndWS_GrPrintLn
-	ndWS_WriteChar
-	ndWS_WriteLetter
-	ndWS_Write
-	ndWS_WriteLn
-	ndWS_PrintLn
-	ndWS_Print
-	ndWS_CharScrolling
-	ndWS_CarriageReturn
-	ndWS_GoCursor
-	ndWS_ClrScr
BaseScreenRender	XWindowRender

These are the prototypes of the functions ws:

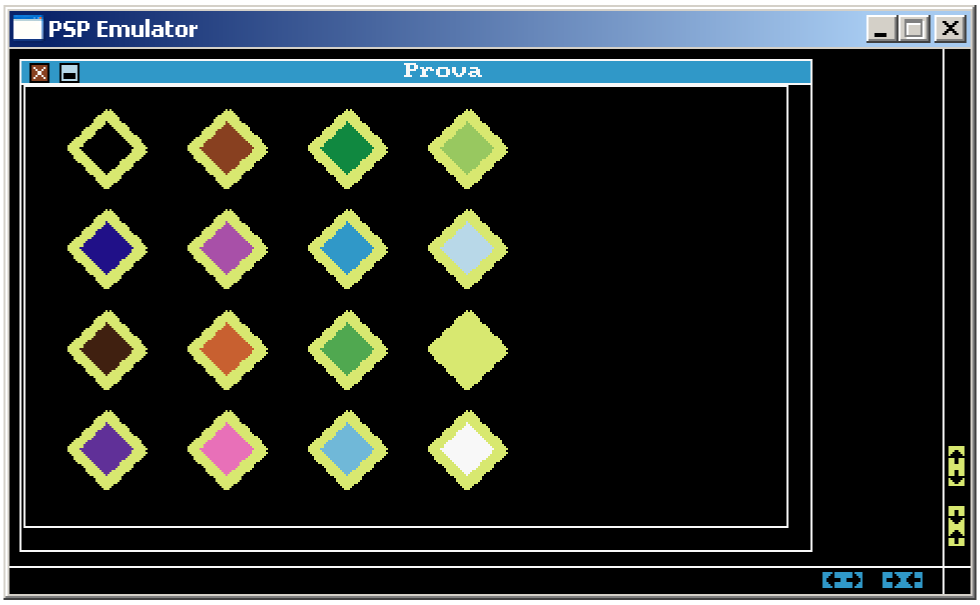
```
int  ndWS_GetPixel (short unsigned int RRPosPixelX, short unsigned int RRPosPixelY, unsigned char
WndHandle);

char ndWS_PutPixel (short unsigned int RRPosPixelX, short unsigned int RRPosPixelY, TypeColor Color,
unsigned char WndHandle, unsigned char RenderNow);

char ndWS_DrawLine (int RRPosX1, int RRPosY1, int RRPosX2, int RRPosY2, TypeColor Color, unsigned
char WndHandle, unsigned char RenderNow);

char ndWS_DrawPoly (int WndHandle, TypeColor Color, char RenderNow, int NrPixels, ...);

char ndWS_FillArea (int PosX, int PosY, TypeColor Color, TypeColor BorderColor, char WndHandle, char
RenderNow);

char ndWS_DrawRectangle (short unsigned int RRPosX1, short unsigned int RRPosY1, short unsigned int
RRPosX2, short unsigned int RRPosY2, TypeColor Color, TypeColor BorderColor, unsigned char
WndHandle, unsigned char RenderNow);

char ndWS_DrawRtRectangle (int CenterRectX, int CenterRectY, int UserDimX, int UserDimY, TypeColor
Color, TypeColor BorderColor, ndint64 CodeStyle, unsigned char WndHandle, unsigned char RenderNow);

char ndWS_DrawSpRectangle (short unsigned int RRPosX1, short unsigned int RRPosY1, short unsigned
int RRPosX2, short unsigned int RRPosY2, TypeColor Color, TypeColor BorderColor, ndint64 CodeStyle,
unsigned char WndHandle, unsigned char RenderNow);

char ndWS_DrawArcOfEllipse (short unsigned int CenterX, short unsigned int CenterY, float RayA,
float RayB, float DegreeA, float DegreeB, TypeColor Color, TypeColor ColorBorder, ndint64 CodeStyle,
char WndHandle, char RenderNow);

char ndWS_DrawEllipse (short unsigned int CenterX, short unsigned int CenterY, float RayA, float
RayB, TypeColor Color, TypeColor ColorBorder, ndint64 CodeStyle, char WndHandle, char RenderNow);

char ndWS_DrawArcOfCircle (short unsigned int CenterX, short unsigned int CenterY, float Ray, float
DegreeA, float DegreeB, TypeColor Color, TypeColor ColorBorder, ndint64 CodeStyle, char WndHandle,
char RenderNow);

char ndWS_DrawCircle (short unsigned int CenterX, short unsigned int CenterY, float Ray, TypeColor
Color, TypeColor ColorBorder, ndint64 CodeStyle, char WndHandle, char RenderNow);

char ndWS_GrWriteChar (short unsigned int RRPosPixelX, short unsigned int RRPosPixelY, unsigned char
Carattere, TypeColor Color, TypeColor BGColor, unsigned char WndHandle, ndint64 TextCode);

char ndWS_GrWriteLn (short unsigned int RRPosPixelX, short unsigned int RRPosPixelY, char *str,
TypeColor Color, TypeColor BGColor, unsigned char WndHandle, ndint64 TextCode);

char ndWS_GrPrintLn (short unsigned int RRPosPixelX, short unsigned int RRPosPixelY, TypeColor
Color, TypeColor BGColor, unsigned char WndHandle, ndint64 TextCode, char *FirstParam, ...);

char ndWS_SetFont (unsigned char FntHandle, char WndHandle);

char XWindowRender (unsigned char WndHandle);

char ndWS_WriteChar (unsigned int PosCursX, unsigned int PosCursY, char Carattere, TypeColor Colore,
unsigned char WndHandle, unsigned char RenderNow);

char ndWS_WriteLetter (char Carattere, TypeColor Colore, unsigned char WndHandle, unsigned char
RenderNow);

char ndWS_Write (char *str, TypeColor Colore, unsigned char WndHandle, unsigned char RenderNow);

char ndWS_WriteLn (char *str, TypeColor Colore, unsigned char WndHandle, unsigned char RenderNow);

char ndWS_PrintLn (char WndHandle, TypeColor Color, char RenderNow, char *FirstParam, ...);

char ndWS_Print (char WndHandle, TypeColor Color, char RenderNow, char *FirstParam, ...);

char ndWS_ClrBufferSpace (unsigned char WndHandle);

char ndWS_ClrALL (unsigned char WndHandle);

char ndWS_ChangeWndColor (TypeColor NewColor, short unsigned int WndHandle);

char ndWS_CharScrolling (unsigned char WndHandle);

char ndWS_CarriageReturn (unsigned char WndHandle);

char ndWS_GoCursor (short unsigned int PosCursX, short unsigned int PosCursY, unsigned char
WndHandle);

char ndWS_ClrScr (unsigned char WndHandle);
```

An example

See this source:

```
#include <nanodesktop.h>

int WndHandle;

int ndMain()
{
    int CounterX, CounterY;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (5, 5, 400, 250, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                COLOR_BLACK, COLOR_WHITE, 0);

    if (WndHandle>=0)
    {
        ndLP_MaximizeWindow (WndHandle);

        for (CounterY=0; CounterY<4; CounterY++)
        {
            for (CounterX=0; CounterX<4; CounterX++)
            {
                     ndWS_DrawRtRectangle (40+60*CounterX, 30+50*CounterY, 30, 30,
                     ndWndColorVector [4*CounterY+CounterX], COLOR_YELLOW,
                     NDKEY_BORDER_SIZE (5) | ND_ROUNDED | NDKEY_ROTATE (45),
                     WndHandle, NORENDER);
            }
        }

        XWindowRender (WndHandle);
    }
}
```

And the result ....



.... and compare the source with the program seen on page 55.

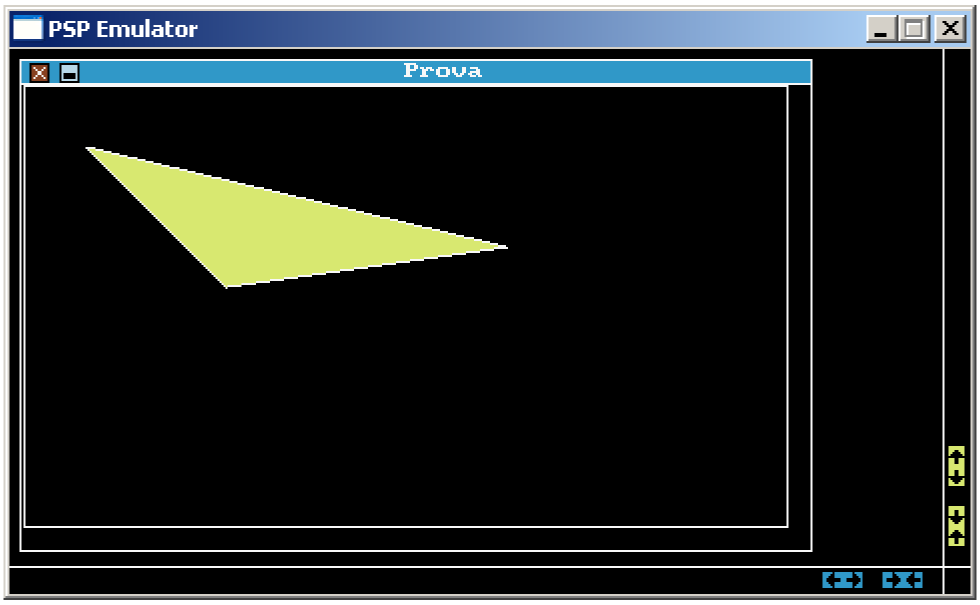
The most important difference is that here we are using ndWS_DrawRtRectangle and not ndBS_DrawRtRectangle. The behaviour of the two routines is identical, but ndWS_DrawRtRectangle operates on a window.

A second example

See this source:

```
#include <nanodesktop.h>

int WndHandle;

int ndMain()
{
    int CounterX, CounterY;
    int CentroX, CentroY;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (5, 5, 400, 250, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                COLOR_BLACK, COLOR_WHITE, 0);

    if (WndHandle>=0)
    {
        ndLP_MaximizeWindow (WndHandle);

        CentroX = (30+100+240)/3;
        CentroY = (30+100+80)/3;

        ndWS_DrawPoly (WndHandle, COLOR_WHITE, NORENDER, 4, 30, 30, 100, 100, 240, 80, 30, 30);
        ndWS_FillArea (CentroX, CentroY, COLOR_YELLOW, COLOR_WHITE, WndHandle, RENDER);

        XWindowRender (WndHandle);
    }
}
```

This is the result:



Compare it with the code seen on page 61

Image support in a window: ndIMG_ShowImage

You can also use the image support of nd in a window.

The functions for loading/saving/modifying images are the same seen in BaseScreen (*ndIMG_LoadImage, ndIMG_SaveImage* and the other).

For image visualization nd provides the routines *ndIMG_ShowImage* and *ndIMG_ShowImageScaled*.

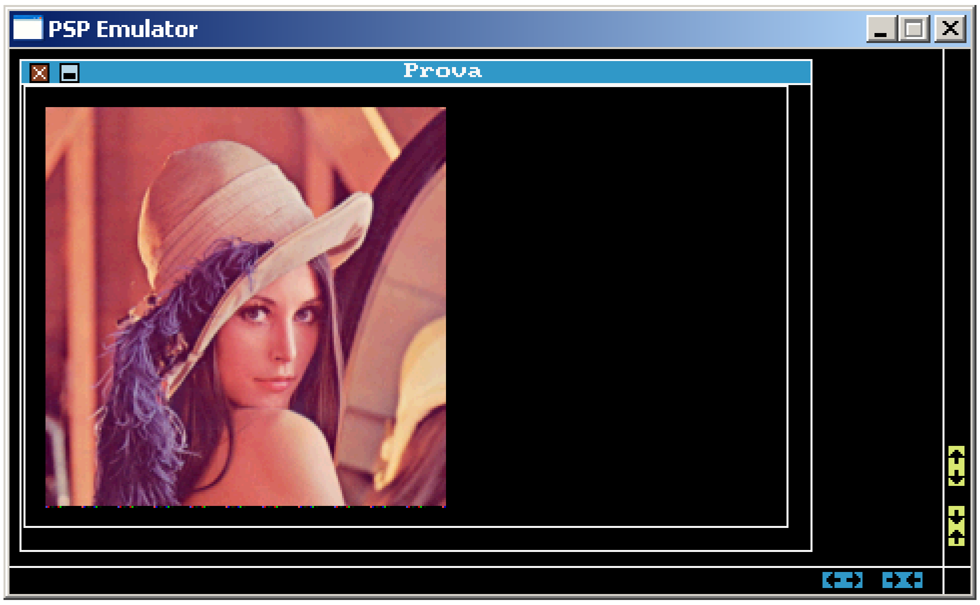
The prototypes of these two routines are the following:

```
char ndIMG_ShowImage (struct ndImage_Type *MyImage, short int RRPosX, short int RRPosY, unsigned
char WndHandle, char RenderNow);

char ndIMG_ShowImageScaled (struct ndImage_Type *MyImage, short int RRPosX, short int RRPosY, float
ScaleX, float ScaleY, unsigned char WndHandle, char RenderNow);
```

This is a simple example:

```
#include <nanodesktop.h>

int WndHandle;

int ndMain()
{
    struct ndImage_Type MyImage;
    int  CounterX, CounterY;
    char ErrRep;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (5, 5, 400, 250, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                COLOR_BLACK, COLOR_WHITE, 0);

    if (WndHandle>=0)
    {
        ndLP_MaximizeWindow (WndHandle);

        ndIMG_InitImage (&MyImage);
        ErrRep=ndIMG_LoadImage (&MyImage, "ms0:/lena.bmp", NDMGKNB);

        if (!ErrRep)
           ndIMG_ShowImage (&MyImage, 10, 10, WndHandle, RENDER);
        else
           printf ("Failed to load images. Error reported %d \n", -ErrRep);

        ndDelay (30);
    }
}
```

And this is the result: *ndIMG_ShowImage* shows the image in the window space



ndWS_GrWriteChar, ndWS_GrWriteLn, ndWS_GrPrintLn

These routines provide a way to write in the window space. It is important to understand that these routines are dedicated to *graphical writing:* when a string is written in the window space, the system replaces the pixels that constitute the letters one by one.

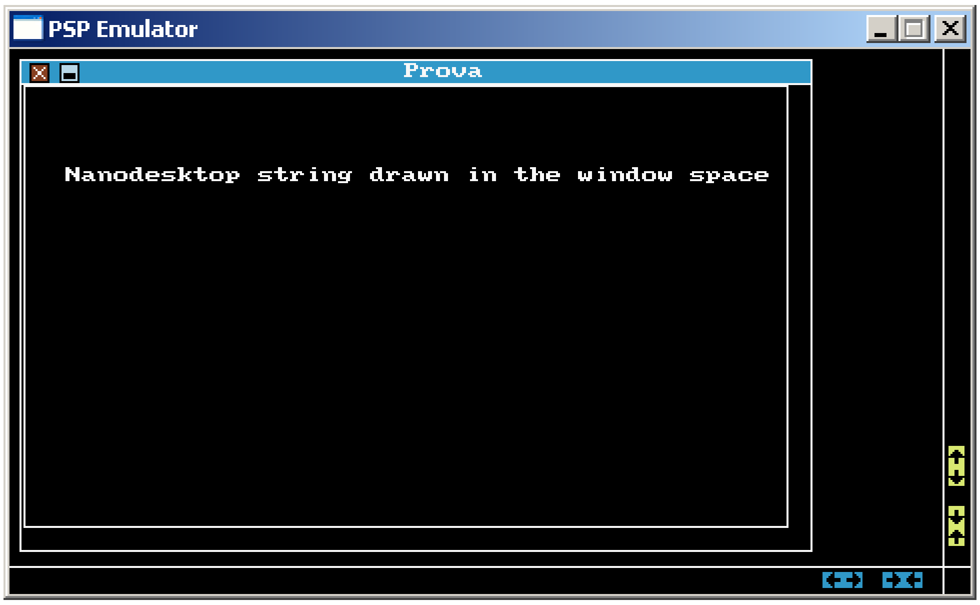
The graphical nature of routines such as *ndWS_GrWriteChar*, *ndWS_GrWriteLn*, *ndWS_GrPrintLn*, have two consequences:

a) These routines don't support a *cursor:* when you use them, you must declare, each time, the coordinates of the ws space at which the string will be drawn;
b) If you create a graphical element that overwrites the pixels of a previous string that had been already drawn, the previous string will disappear from the screen.

This is an example of this effect:

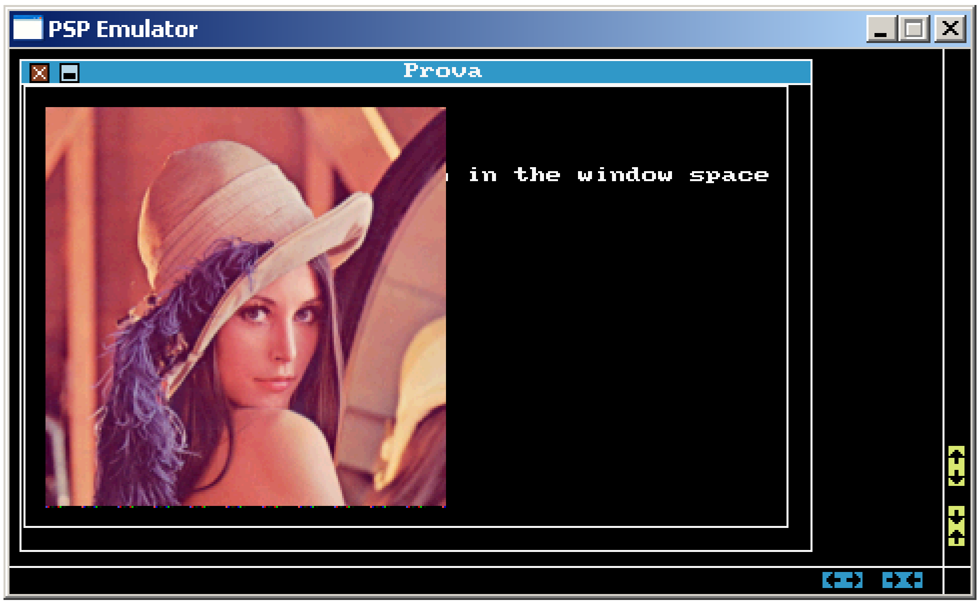
```
#include <nanodesktop.h>

int WndHandle;

int ndMain()
{
    struct ndImage_Type MyImage;
    int  CounterX, CounterY;
    char ErrRep;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (5, 5, 400, 250, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                COLOR_BLACK, COLOR_WHITE, 0);

    if (WndHandle>=0)
    {
        ndLP_MaximizeWindow (WndHandle);

        ndWS_GrWriteLn (20, 40, "Nanodesktop string drawn in the window space", COLOR_WHITE,
                               COLOR_BLACK, WndHandle, RENDER);
        ndDelay (10);

        ndIMG_InitImage (&MyImage);
        ErrRep=ndIMG_LoadImage (&MyImage, "ms0:/lena.bmp", NDMGKNB);

        if (!ErrRep)
           ndIMG_ShowImage (&MyImage, 10, 10, WndHandle, RENDER);
        else
           printf ("Failed to load images. Error reported %d \n", -ErrRep);

        ndDelay (30);
    }
}
```

This is the result: first



```
Nanodesktop string drawn in the window space
```

and after: the image has overwritten the graphical string.



Chapter 12
The char map of a window

Under nd, each window has a dedicated *char map.*

The chars in the char map are visualized over the window space: this allows you to create some interesting visual effects: for example, text that scrolls over a bitmap, is very simple to realize.

For writing in the char map, there are dedicated functions, as ndWS_WriteLn or ndWS_PrintLn.

These type of routines are specific to window space: there is no counterpart in basescreen, because it doesn't support a char map.

The nature of the char map routines has the following consequences:

a) each char map, one for any window, has its own cursor. The routines always write at the cursor position. The system provides dedicated routines that allow you to change the cursor position, to execute the scrolling text, to clear the char map without touching window space;
b) when you write a char in the map, the char is visualized over the window space, but the content of the window space is maintained separate and untouched.

What are the dimensions of the char map ? How many chars does it accept in x-dimension and how many in y-dimension ?

The number of chars in x and y dim is defined at the moment of window creation: it is obtained by dividing the width and the height of the window space by 8.

You can obtain the number of chars in x-dim and in y-dim reading these values directly from the struct WindowData

```
WindowData [WndHandle].WS_MaxCursX
WindowData [WndHandle].WS_MaxCursY
```

The struct WindowData is a *system struct:* it contains all the information about each window. The prototype of this structure is defined in *<ndenv path>\PSP\SDK\Nanodesktop\src\$$_Variables.h*

In Nanodesktop 0.3, the prototype is the following:

```
struct WindowDataType
{
   char *WindowTitle;                            // Puntatore al titolo della finestra
   char WindowTitleChar [__MAXCURSX];            // Array interno per salvataggio caratteri

   TypeColor ColorTitle, ColorBGTitle, ColorBorder, ColorBGWS;
                                                 // Memorizzano i colori della finestra

   unsigned short int WndLimitX1, WndLimitY1, WndLimitX2, WndLimitY2;
                                                 // Limiti delle finestre sullo schermo

   unsigned short int WndLength, WndHeight;       // Lunghezza e larghezza della finestra
   unsigned short int HwWndLength, HwWndHeight;   // Lunghezza e larghezza della finestra per
                                                 // l'acceleratore hardware

   unsigned short int WSWndLength, WSWndHeight;   // Lunghezza e larghezza del Window Space

   unsigned short int ROI_PosX, ROI_PosY;         // Posizione della region of interest del WS
                                                 // visualizzata

   unsigned short int ROI_LenX, ROI_LenY;         // Dimensione della region of interest

   unsigned short int ROI_PosX1, ROI_PosY1;
   unsigned short int ROI_PosX2, ROI_PosY2;

   unsigned short int AA1_X1, AA1_X2, AA1_Y1, AA1_Y2, AA1_LenX, AA1_LenY;
                                                 // Posizione e dimensioni dell'area attiva
                                                 // livello 1
```

```
    unsigned short int AA2_X1, AA2_X2, AA2_Y1, AA2_Y2, AA2_LenX, AA2_LenY;
                                            // Posizione e dimensioni dell'area attiva
                                            // livello 2

    unsigned short int AA3_X1, AA3_X2, AA3_Y1, AA3_Y2, AA3_LenX, AA3_LenY;
                                            // Posizione e dimensioni dell'area attiva livello 3

    unsigned short int OrizzBar_PosX1, OrizzBar_PosY1;   // Coordinate della barra orizzontale
    unsigned short int OrizzBar_PosX2, OrizzBar_PosY2;

    unsigned short int VertBar_PosX1, VertBar_PosY1;     // Coordinate della barra verticale
    unsigned short int VertBar_PosX2, VertBar_PosY2;

    TypeColor *_p;                                       // Puntatore ai dati della pagina logica
    TypeColor *_ws;                                      // Puntatore ai dati del window space
    TypeColor *_ss;                                      // Puntatore ai dati del service space
    unsigned char *_CharMap;                             // Puntatore alla mappa caratteri

    TypeColor *_ColorCharMap;                            // Puntatore alla color char map

    unsigned char WindowIsActive;                        // Posta a 1 quando la finestra è attiva
    unsigned char Minimize;                              // Posto a 1 quando la finestra viene
                                                        // minimizzata;

    unsigned short int WS_MaxCursX, WS_MaxCursY;         // Posizione massima del cursore nella
                                                        // finestra

    unsigned short int WS_PosCursX, WS_PosCursY;         // Posizione del cursore nella finestra

    unsigned short int MargineCharX, MargineCharY;       // Margini per la visualizzazione caratteri
    unsigned short int CharArea_PosX1, CharArea_PosY1;
    unsigned short int CharArea_PosX2, CharArea_PosY2;

    struct ButtonClass3_DataType ButtonClass3_Data[7];   // Array che contiene le informazioni
                                                        // inerenti ai pulsanti di classe 3

    char NrButtonsAllocated;

    struct ButtonClass4_DataType ButtonClass4_Data[__NRBUTTONSALLOCABLE];
                                                         // Array che contiene le informazioni
                                                         // inerenti ai pulsanti di classe 4

    unsigned char NoResizeable;

    unsigned char MenuSupported;

    unsigned short int MenuBar_PosX1, MenuBar_PosY1;
    unsigned short int MenuBar_PosX2, MenuBar_PosY2;
    unsigned short int MenuBar_LenX,  MenuBar_LenY;

    unsigned short int MenuBar_LA_PosX1, MenuBar_LA_PosY1;
    unsigned short int MenuBar_LA_PosX2, MenuBar_LA_PosY2;
    unsigned short int MenuBar_LA_LenX,  MenuBar_LA_LenY;

    char NrWMI1ButtonsAllocated;
    struct ButtonWMI1_DataType ButtonWMI1 [NRMAX_WMI1_BUTTON_ALLOCABLE];

    struct ButtonWMI2Serv_DataType ButtonWMI2Serv [2];

    char NrWMI2ButtonsAllocated;
    struct ButtonWMI2_DataType ButtonWMI2 [NRMAX_WMI2_BUTTON_ALLOCABLE];

    char NrTrackBarsAllocated;
    struct TrackBar_Type TrackBar [__NRTRACKBARSALLOCABLE];

    char NrTextBoxAllocated;
    struct TextBox_Type TextBox [__NRTEXTBOXALLOCABLE];

    char NrListBoxAllocated;
    struct ListBox_Type ListBox [__NRLISTBOXALLOCABLE];

    char UserArea [__USERAREASIZE];

    long int (*PntToCloseWndCallback)(char WndHandle, ndint64 WndInfoField);
    ndint64  WndInfoField;

    char NoNotifyToMouseControl;

    char ndWS_CurrentFntHandle;
    char ndMB_CurrentFntHandle;
    char ndLP_CurrentFntHandle;
};
```

If you want to know how many chars are allowed in the x-dim and in the y-dim in a window, you can write:

```
printf ("%d %d \n", WindowData [WndHandle].WS_MaxCursX, WindowData [WndHandle].WS_MaxCursY);
```

ndWS_WriteChar

Now we can see ndWS_WriteChar.

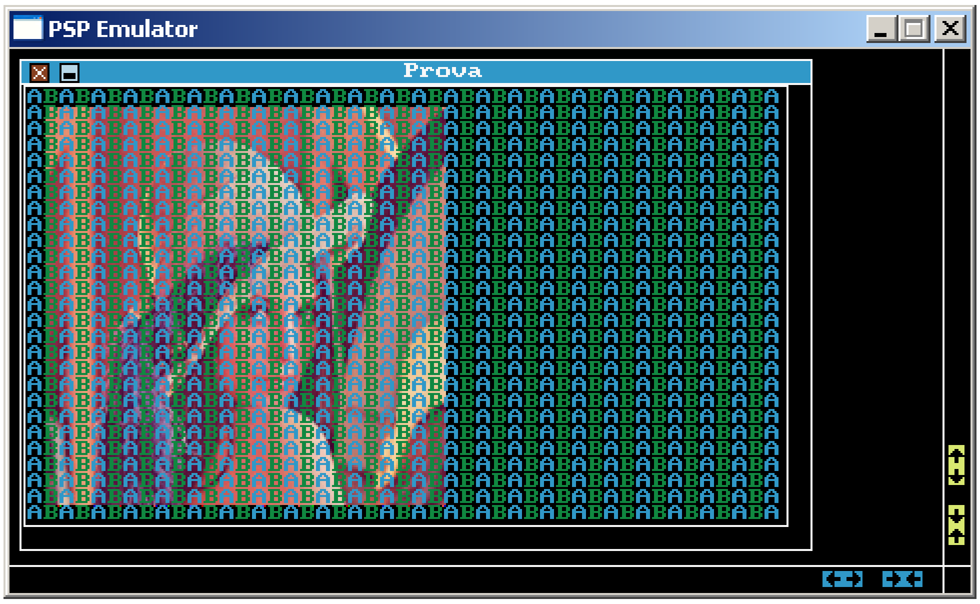
```
char ndWS_WriteChar (unsigned int PosCursX, unsigned int PosCursY, char Carattere, TypeColor Colore,
unsigned char WndHandle, unsigned char RenderNow);
```

This routine writes a single character to a specific position of the char map. See this source:

```
#include <nanodesktop.h>

int WndHandle;

int ndMain()
{
    struct ndImage_Type MyImage;
    int  CounterX, CounterY;
    char ErrRep;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (5, 5, 400, 250, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                COLOR_BLACK, COLOR_WHITE, 0);

    if (WndHandle>=0)
    {
        ndLP_MaximizeWindow (WndHandle);

        ndIMG_InitImage (&MyImage);
        ErrRep=ndIMG_LoadImage (&MyImage, "ms0:/lena.bmp", NDMGKNB);

        if (!ErrRep)
           ndIMG_ShowImage (&MyImage, 10, 10, WndHandle, RENDER);
        else
           printf ("Failed to load images. Error reported %d \n", -ErrRep);

        ndDelay (3);

        for (CounterY=0; CounterY<WindowData [WndHandle].WS_MaxCursY; CounterY++)
        {
            for (CounterX=0; CounterX<WindowData [WndHandle].WS_MaxCursX; CounterX=CounterX+2)
            {
                ndWS_WriteChar (CounterX+0, CounterY, 'A', COLOR_LBLUE, WndHandle, NORENDER);
                ndWS_WriteChar (CounterX+1, CounterY, 'B', COLOR_GREEN, WndHandle, NORENDER);
            }
        }

        XWindowRender (WndHandle);
    }
}
```

Be careful that ndWS_WriteChar accepts, as the third parameter, a *char* and not a *char* (i.e. a string address).* The loop changes CounterX and CounterY and *WriteChar* puts the requested char in the char map.

The result is shown on the following page. As you can see, the chars in the char map are visualized over window space: this behaviour is very different from that typical of ndWS_Gr.... routines.



ndWS_WriteLetter

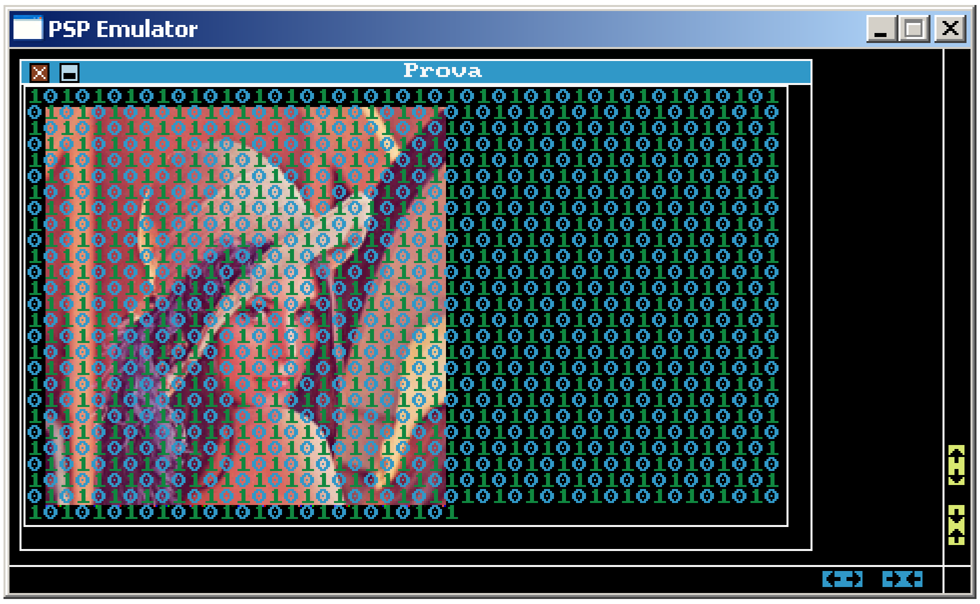
ndWS_WriteChar writes a char in a position of the char map that must be specified explicitly at each call of the routine. ndWS_WriteLetter, instead, writes a letter at the current position of the cursor.

```
char ndWS_WriteLetter (char Carattere, TypeColor Colore, unsigned char WndHandle, unsigned char
RenderNow);
```

See this source:

```
#include <nanodesktop.h>

int WndHandle;

int ndMain()
{
    struct ndImage_Type MyImage;
    int  CounterX, CounterY;
    char ErrRep;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (5, 5, 400, 250, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                COLOR_BLACK, COLOR_WHITE, 0);

    if (WndHandle>=0)
    {
        ndLP_MaximizeWindow (WndHandle);

        ndIMG_InitImage (&MyImage);
        ErrRep=ndIMG_LoadImage (&MyImage, "ms0:/lena.bmp", NDMGKNB);

        if (!ErrRep)
           ndIMG_ShowImage (&MyImage, 10, 10, WndHandle, RENDER);
        else
           printf ("Failed to load images. Error reported %d \n", -ErrRep);

        ndDelay (3);

        for (CounterY=0; CounterY<WindowData [WndHandle].WS_MaxCursY; CounterY++)
        {
            for (CounterX=0; CounterX<WindowData [WndHandle].WS_MaxCursX; CounterX=CounterX+2)
            {
                ndWS_WriteLetter ('0', COLOR_LBLUE, WndHandle, NORENDER);
                ndWS_WriteLetter ('1', COLOR_GREEN, WndHandle, NORENDER);
            }
        }

        XWindowRender (WndHandle);
    }
}
```

Result:



ndWS_GoCursor

The position of the cursor can be read from *struct WindowData*

```
WindowData [WndHandle].WS_PosCursX
WindowData [WndHandle].WS_PosCursY
```

You can change the position of the cursor, using the routine *ndWS_GoCursor*.

```
ndWS_GoCursor (short unsigned int PosCursX, short unsigned int PosCursY, unsigned char WndHandle);
```

ndWS_CarriageReturn

This routine moves the cursor to the next line

ndWS_Write, ndWS_WriteLn

If you want to write a string, you can use the routine *ndWS_WriteLn* or *ndWS_Write*.

```
char ndWS_Write (char *str, TypeColor Colore, unsigned char WndHandle, unsigned char RenderNow);
char ndWS_WriteLn (char *str, TypeColor Colore, unsigned char WndHandle, unsigned char RenderNow);
```

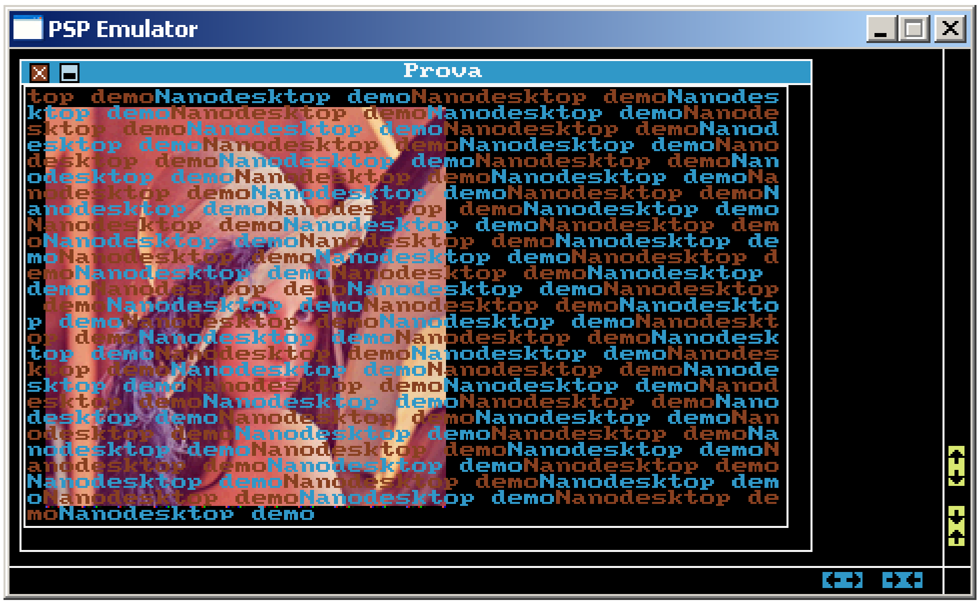
The difference between the former and the latter is that *ndWS_WriteLn* executes a carriage return after having written the string. *ndWS_Write* doesn't do this.

In this program we use ndWS_Write

```
#include <nanodesktop.h>

int WndHandle;

int ndMain()
{
    struct ndImage_Type MyImage;
    int  Counter;
    char ErrRep;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (5, 5, 400, 250, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                COLOR_BLACK, COLOR_WHITE, 0);

    if (WndHandle>=0)
    {
        ndLP_MaximizeWindow (WndHandle);

        ndIMG_InitImage (&MyImage);
        ErrRep=ndIMG_LoadImage (&MyImage, "ms0:/lena.bmp", NDMGKNB);

        if (!ErrRep)
           ndIMG_ShowImage (&MyImage, 10, 10, WndHandle, RENDER);
        else
           printf ("Failed to load images. Error reported %d \n", -ErrRep);

        ndDelay (3);

        for (Counter=0; Counter<100; Counter++)
        {
            ndWS_Write ("Nanodesktop demo", COLOR_LBLUE, WndHandle, RENDER);
            ndWS_Write ("Nanodesktop demo", COLOR_RED, WndHandle, RENDER);
        }
    }
}
```

Note that when the cursor goes out of the char map, nd executes automatically a *chars scrolling,* in a way similar to a terminal such as bash or command.com. The *chars scrolling,* can also be forced using a routine such as ndWS_CharScrolling ().

The result is this:



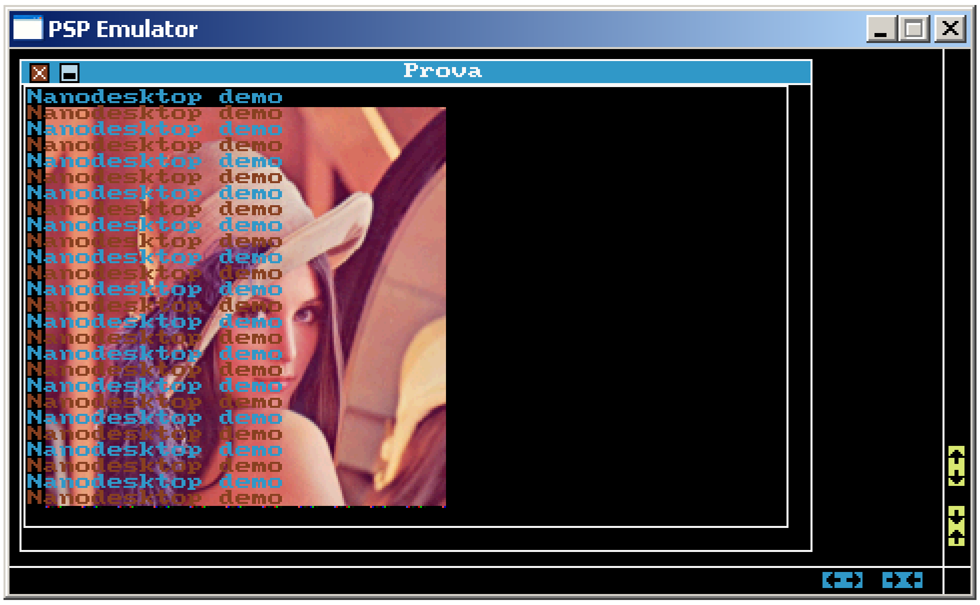
```
top demoNanodesktop demoNanodesktop demoNanodes
ktop demoNanodesktop demoNanodesktop demoNanode
sktop demoNanodesktop demoNanodesktop demoNanod
esktop demoNanodesktop demoNanodesktop demoNano
desktop demoNanodesktop demoNanodesktop demoNan
odesktop demoNanodesktop demoNanodesktop demoNa
nodesktop demoNanodesktop demoNanodesktop demoN
anodesktop demoNanodesktop demoNanodesktop demo
Nanodesktop demoNanodesktop demoNanodesktop dem
oNanodesktop demoNanodesktop demoNanodesktop de
moNanodesktop demoNanodesktop demoNanodesktop d
emoNanodesktop demoNanodesktop demoNanodesktop
demoNanodesktop demoNanodesktop demoNanodesktop
 demoNanodesktop demoNanodesktop demoNanodeskto
p demoNanodesktop demoNanodesktop demoNanodeskt
op demoNanodesktop demoNanodesktop demoNanodesk
top demoNanodesktop demoNanodesktop demoNanodes
ktop demoNanodesktop demoNanodesktop demoNanode
sktop demoNanodesktop demoNanodesktop demoNanod
esktop demoNanodesktop demoNanodesktop demoNano
desktop demoNanodesktop demoNanodesktop demoNan
odesktop demoNanodesktop demoNanodesktop demoNa
nodesktop demoNanodesktop demoNanodesktop demoN
anodesktop demoNanodesktop demoNanodesktop demo
Nanodesktop demoNanodesktop demoNanodesktop dem
oNanodesktop demoNanodesktop demoNanodesktop de
moNanodesktop demo
```

Here, we use ndWS_WriteLn:

```
#include <nanodesktop.h>

int WndHandle;

int ndMain()
{
    struct ndImage_Type MyImage;
    int  Counter;
    char ErrRep;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (5, 5, 400, 250, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                 COLOR_BLACK, COLOR_WHITE, 0);

    if (WndHandle>=0)
    {
        ndLP_MaximizeWindow (WndHandle);

        ndIMG_InitImage (&MyImage);
        ErrRep=ndIMG_LoadImage (&MyImage, "ms0:/lena.bmp", NDMGKNB);

        if (!ErrRep)
           ndIMG_ShowImage (&MyImage, 10, 10, WndHandle, RENDER);
        else
```

```
                printf ("Failed to load images. Error reported %d \n", -ErrRep);

            ndDelay (3);

            for (Counter=0; Counter<100; Counter++)
            {
                ndWS_WriteLn ("Nanodesktop demo", COLOR_LBLUE, WndHandle, RENDER);
                ndWS_WriteLn ("Nanodesktop demo", COLOR_RED, WndHandle, RENDER);
            }
        }
}
```

As you can see....



```
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
Nanodesktop demo
```

... the system executes a CarriageReturn after each operation of writing

ndWS_Print, ndWS_PrintLn

These routines are provided to print a complex string in the char map. The prototypes are:

```
char ndWS_PrintLn  (char WndHandle, TypeColor Color, char RenderNow, char *FirstParam, ...);   char
char ndWS_Print (char WndHandle, TypeColor Color, char RenderNow, char *FirstParam, ...);
```

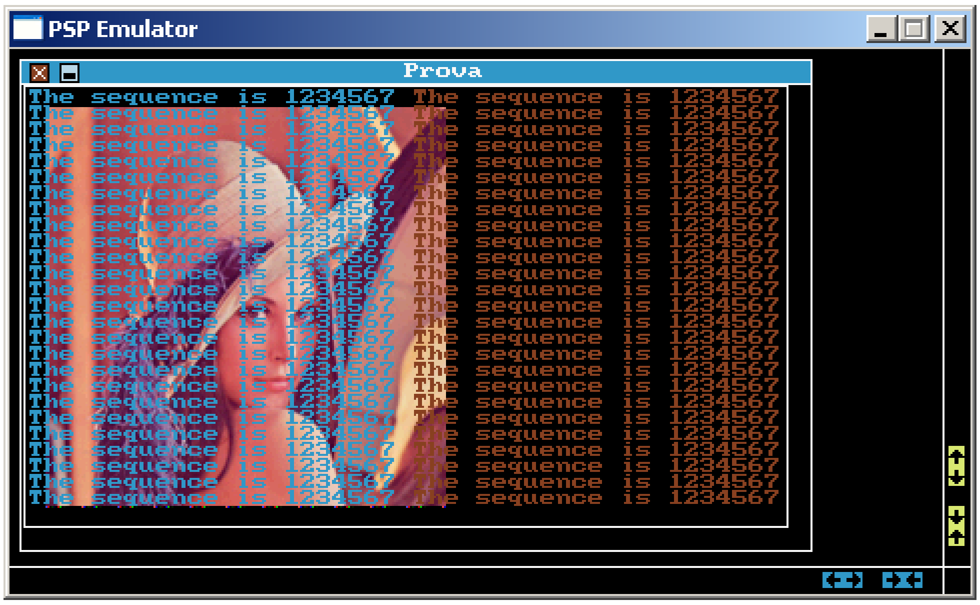
They allow you to print a string containing numbers, other strings, hexadecimals etc.

Remember that the '\n' char cannot be used in the complex string passed to these routines: you must use ndWS_PrintLn if you want the cursor to go to a newline after writing the string.

Example:

```
#include <nanodesktop.h>

int WndHandle;

int ndMain()
{
    struct ndImage_Type MyImage;
    int  Counter;
    char ErrRep;
    int  Sequence;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (5, 5, 400, 250, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                 COLOR_BLACK, COLOR_WHITE, 0);

    if (WndHandle>=0)
    {
        ndLP_MaximizeWindow (WndHandle);

        ndIMG_InitImage (&MyImage);
        ErrRep=ndIMG_LoadImage (&MyImage, "ms0:/lena.bmp", NDMGKNB);

        if (!ErrRep)
           ndIMG_ShowImage (&MyImage, 10, 10, WndHandle, RENDER);
        else
           printf ("Failed to load images. Error reported %d \n", -ErrRep);

        ndDelay (3);

        Sequence = 1234567;

        for (Counter=0; Counter<100; Counter++)
        {
            ndWS_Print (WndHandle, COLOR_LBLUE, RENDER, "The sequence is %d ", Sequence);
            ndWS_Print (WndHandle, COLOR_RED, RENDER, "The sequence is %d", Sequence);
        }
    }
}
```

The routine prints on screen the number 1234567



```
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
The sequence is 1234567 The sequence is 1234567
```

ndWS_ClrScr

The routine ndWS_ClrScr clears the char map. Remember that the window space remains untouched.

ndWS_ClrALL

This routine clears the char map *and* the window space. The result is a void window.

Chapter 13
Buttons

Now we'll see how to create buttons in a window. This step is fundamental to understanding how to create applications that can interact with the user.

You can create a new button using the routine *ndCTRL_CreateButton.*

Here, the situation is quite complex. There are 3 different routines for button creation: *ndCTRL_CreateButton, ndCTRL_CreateButtonSmart,* and *ndCTRL_CreateButtonComplex*.

We begin with *ndCTRL_CreateButton, which* is the simplest of them. Its prototype is:

```
char ndCTRL_CreateButton   ( short unsigned int RRPosX1, short unsigned int RRPosY1,
                             short unsigned int RRPosX2, short unsigned int RRPosY2,
                             char WndHandle, char *StringID,
                             char *LabelButton1, char *LabelButton2,
                             TypeColor ColorText,
                             TypeColor ColorButton,
                             TypeColor ColorBorderButton,
                             ndint64 ButtonStyle,
                             void* Callback,
                             ndint64 InfoField,
                             char RenderNow        );
```

Here's the meaning of each parameter:

- *RRPosX1, RRPosY1, RRPosX2, RRPosY2* are the coordinates of the button.
- *WndHandle* is the handle of the window in which the button will be created
- *StringID* is the address of a string, that uniquely identifies the button. This string will be passed to MouseControl, each time the button is pressed;
- *LabelButton1, LabelButton2* are two strings: they will be visualized in the button area
- *ColorText, ColorButton, ColorBorderButton*, are the colors that will be used in the button;
- *ButtonStyle* is a 64 bit parameter that specifies important options about the behaviour of the button;
- *Callback* is the address of button callback routine
- *InfoField* is a 64 bit value that will be passed to the callback, when the button will be pressed;
- *RenderNow* has the usual meaning: it can be RENDER or NORENDER

When the button is pressed, the system will recall a routine called *button callback.* This routine is specified by the user when the button is created.

In this first example, we'll set parameter ButtonStyle to 0. Moreover, we'll set parameter Callback to 0: this is equivalent to disable the callback for a button.

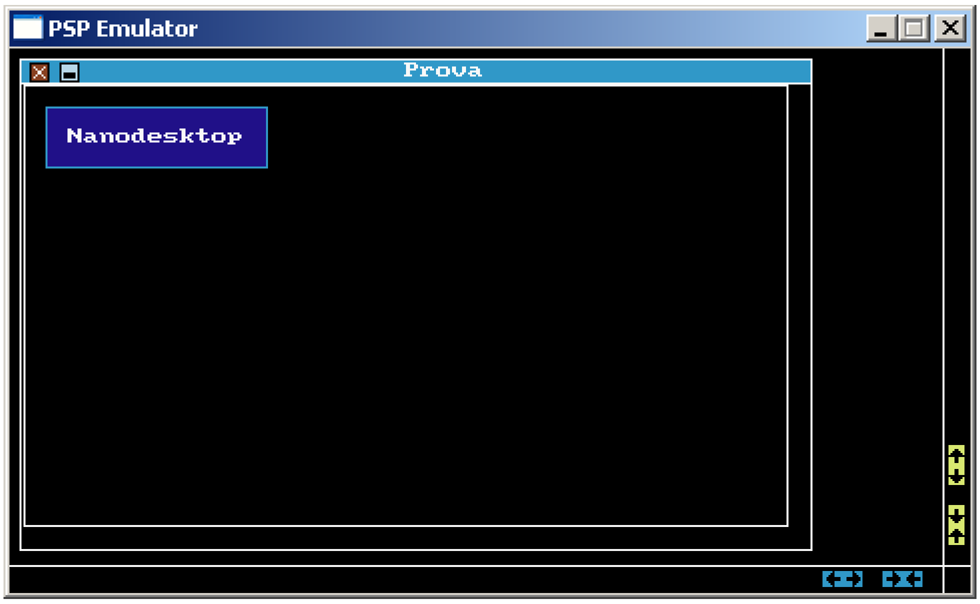
When the button has been successfully created, the routine returns a number that represents the *button handle.* The button handle is always a positive number; when CreateButton returns a negative value, an error has occurred.

In other words, each button is identified using two criteria: through a number called *button handle,* and through a string called *stringID.*

See this code:

```
#include <nanodesktop.h>

int WndHandle, BtnHandle;

int ndMain()
{
    ndInitSystem();

    WndHandle=ndLP_CreateWindow (5, 5, 400, 250, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                 COLOR_BLACK, COLOR_WHITE, 0);
    if (WndHandle>=0)
    {
        ndLP_MaximizeWindow (WndHandle);

        BtnHandle = ndCTRL_CreateButton          ( 10, 10, 120, 40,  WndHandle, "Button0",
                                                   "Nanodesktop", "",
                                                   COLOR_WHITE, COLOR_BLUE, COLOR_LBLUE,
                                                   0, 0, 12345678, RENDER        );
    }
}
```

View the result:

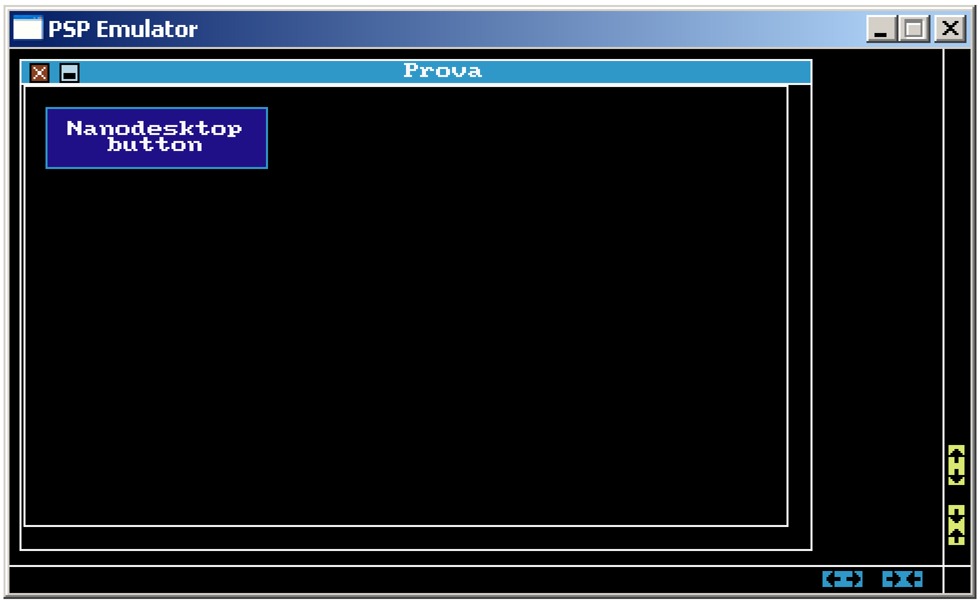


The stringID of this button is **Button0.** This string identifies the button. The labels (in blue) are drawn over button area: the system automatically centres the string to the centre of the button. If you don't want the label, it's sufficient to set to “” the second label string.

View this new source:

```
#include <nanodesktop.h>

int WndHandle, BtnHandle;

int ndMain()
{
    ndInitSystem();

    WndHandle=ndLP_CreateWindow (5, 5, 400, 250, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                 COLOR_BLACK, COLOR_WHITE, 0);

    if (WndHandle>=0)
    {
        ndLP_MaximizeWindow (WndHandle);

        BtnHandle = ndCTRL_CreateButton   ( 10, 10, 120, 40,  WndHandle, "Button0",
                                            "Nanodesktop", "button",
                                            COLOR_WHITE, COLOR_BLUE, COLOR_LBLUE,
                                            0, 0, 12345678, RENDER              );
    }
}
```

Here, there are two labels: “Nanodesktop” and “button”. The system centres them automatically as you can see:



Remember that you can't assign the same stringID to two different buttons that belong to the same window. The stringID is specific for each button.

To round the borders

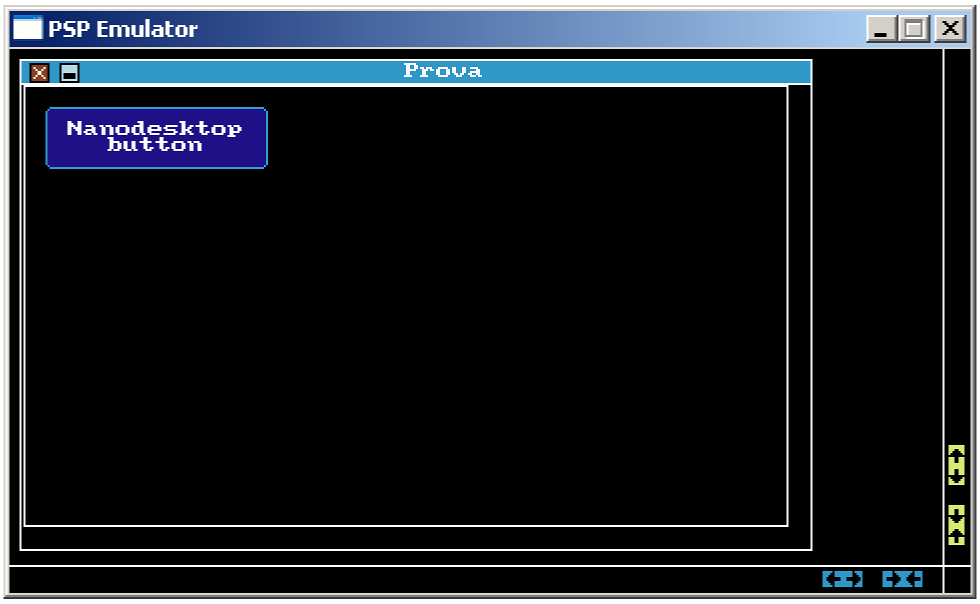
You can round the borders of a button using the option ND_BUTTON_ROUNDED in the 64-bit parameter *ButtonStyle*

See this source:

```
#include <nanodesktop.h>

int WndHandle, BtnHandle;

int ndMain()
{
    ndInitSystem();

    WndHandle=ndLP_CreateWindow (5, 5, 400, 250, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                COLOR_BLACK, COLOR_WHITE, 0);

    if (WndHandle>=0)
    {
        ndLP_MaximizeWindow (WndHandle);

        BtnHandle = ndCTRL_CreateButton   ( 10, 10, 120, 40,  WndHandle, "Button0",
                                            "Nanodesktop", "button",
                                            COLOR_WHITE, COLOR_BLUE, COLOR_LBLUE,
                                            ND_BUTTON_ROUNDED, 0, 12345678, RENDER  );
    }
}
```

The examples that we have seen before creates buttons without assigning a button callback (see the 0 in blue in the routine).

InfoField is set to 12345678.



Images in the button (ndCTRL_CreateButtonComplex)

When you create a new button, you can draw a small image or a small icon in its area. You can do this using the routine *ndCTRL_CreateButtonComplex*

```
char ndCTRL_CreateButtonComplex (    short unsigned int RRPosX1, short unsigned int RRPosY1,
                                     short unsigned int RRPosX2, short unsigned int RRPosY2,
                                     char WndHandle, char *StringID,
                                     char *LabelButton1, char *LabelButton2,
                                     TypeColor ColorText, TypeColor ColorButton,
                                     TypeColor ColorBorderButton,
                                     short unsigned int Q1PosX1, short unsigned int Q1PosY1,
                                     short unsigned int Q1PosX2, short unsigned int Q1PosY2,
                                     struct ndImage_Type *IconImage,
                                     short unsigned int Q2PosX1, short unsigned int Q2PosY1,
                                     short unsigned int Q2PosX2, short unsigned int Q2PosY2,
                                     ndint64 ButtonStyle, void* Callback, ndint64 InfoField,
                                     char RenderNow        );
```

It is a very complex routine that allows you to create a button with its own icon.

As you can see, there are 9 new parameters for this function that have been added, compared to ndCTRL_CreateButton.

Q1PosX1, Q1PosY1, Q1PosX2, Q1PosY2 represent the coordinates of the area in which will be created the labels associated to the button.

Q2PosX1, Q2PosY1, Q2PosX2, Q2PosY2 represent the coordinates of the area in which will be drawn the image of the icon. The image that will be visualized is pointed by *IconImage.

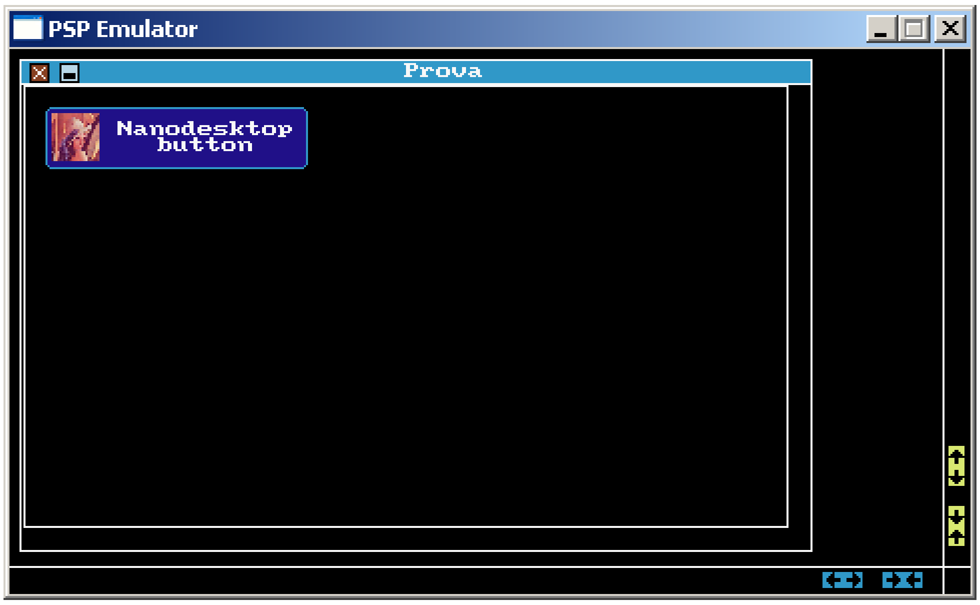
Also, this routine returns the handle assigned to the button. Keep in mind two things:

a) The "Q" coordinates are referred to the upper left angle of the button.
b) Nanodesktop resizes automatically the image described by *IconImage, so that the image fits all the Q1 area in the button;
c) The images used for ndCTRL_CreateButtonComplex can be taken from disk (loaded using ndIMG_LoadImage), but they can also be taken by an NDF array. If you use only NDF arrays, you can create an executable that contains the images of the icons and of the buttons with no need for external image loading.

See this source:

```
#include <nanodesktop.h>

int WndHandle, BtnHandle;

int ndMain()
{
    struct ndImage_Type MyImage;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (5, 5, 400, 250, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                COLOR_BLACK, COLOR_WHITE, 0);

    if (WndHandle>=0)
    {
        ndLP_MaximizeWindow (WndHandle);

        ndIMG_InitImage (&MyImage);
        ndIMG_LoadImage (&MyImage, "ms0:/lena.bmp", NDMGKNB);

        BtnHandle=ndCTRL_CreateButtonComplex   ( 10, 10, 10+130, 10+30,  WndHandle, "TheButton",
                                              "Nanodesktop", "button",
                                               COLOR_WHITE, COLOR_BLUE, COLOR_LBLUE,
                                               30, 0, 130, 30,
                                               &MyImage,
                                               3, 3, 27, 27,
                                               ND_BUTTON_ROUNDED, 0, 12345678, RENDER );
    }
}
```

The program loads the image “lena.bmp”, and visualizes it in the button. The button has width 130 and height 30: the Q2 area, in which will be visualized the icon, has (relative) coordinates (3, 3, 27, 27).



CreateButtonSmart

ndCTRL_CreateButtonComplex is a very complex routine and it requires many parameters. Nanodesktop helps the developer with a function called ndCTRL_CreateButtonSmart.

```
char ndCTRL_CreateButtonSmart (short unsigned int RRPosX1, short unsigned int RRPosY1,
                               char WndHandle,
                               char *StringID, char *LabelButton1, char *LabelButton2,
                               TypeColor ColorText, TypeColor ColorButton,
                               TypeColor ColorBorderButton,
                               struct ndImage_Type *ImageIcon,
                               ndint64 ButtonStyle, void* Callback, ndint64 InfoField,
                               char RenderNow);
```

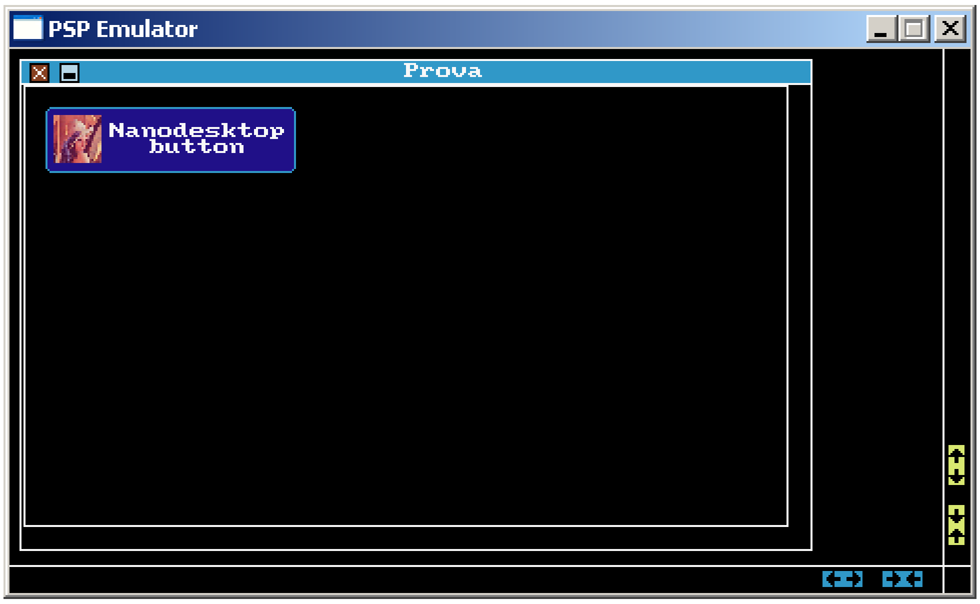
This routine calculates automatically the button coordinates and the “Q” coordinates.

The only information that this routine needs is the position of the upper left corner of the button.

This is an example:

```
#include <nanodesktop.h>

int WndHandle, BtnHandle;

int ndMain()
{
    struct ndImage_Type MyImage;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (5, 5, 400, 250, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                COLOR_BLACK, COLOR_WHITE, 0);

    if (WndHandle>=0)
    {
        ndLP_MaximizeWindow (WndHandle);

        ndIMG_InitImage (&MyImage);
        ndIMG_LoadImage (&MyImage, "ms0:/lena.bmp", NDMGKNB);

        BtnHandle = ndCTRL_CreateButtonSmart   (    10, 10,   WndHandle, "TheButton",
                                                    "Nanodesktop", "button",
                                                    COLOR_WHITE, COLOR_BLUE, COLOR_LBLUE,
                                                    &MyImage,
                                                    ND_BUTTON_ROUNDED, 0, 12345678,
                                                    RENDER         );
    }
}
```

The result is this:



Logical buttons

In some cases it can be useful to create an *invisible* button, i.e. a button that is present on the screen, but that has no graphical visualization.

This type of button is called "*logic mode*". You can create a button of this type, using the option ND_BUTTON_LOGIC_MODE in 64-bit parameter ButtonHandle.

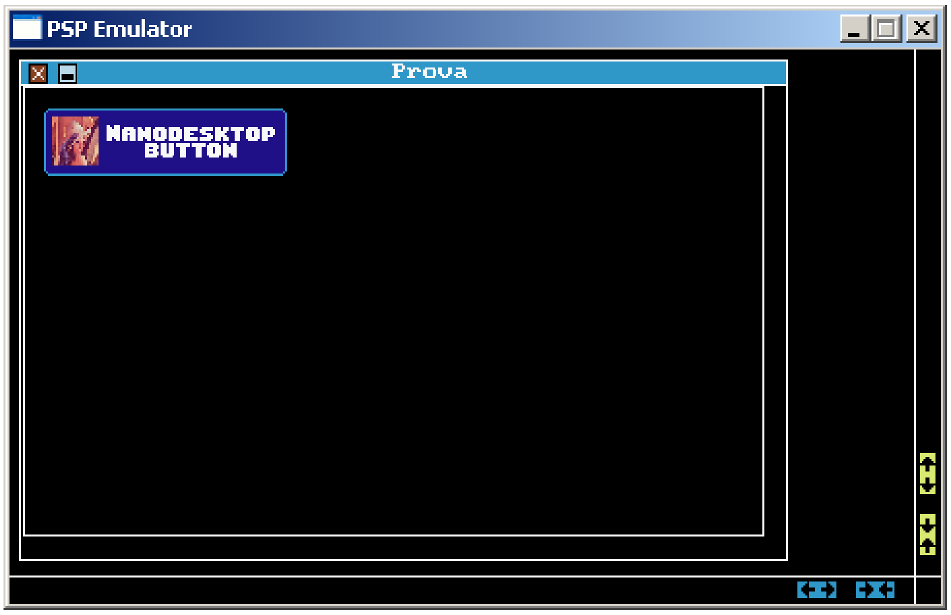
Fonts

You can set the font that will be used in button visualization, using the key NDKEY_SETFONT.

See this example:

```
#include <nanodesktop.h>

int WndHandle;

int ndMain()
{
    struct ndImage_Type MyImage;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (5, 5, 400, 250, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                COLOR_BLACK, COLOR_WHITE, 0);

    if (WndHandle>=0)
    {
        ndLP_MaximizeWindow (WndHandle);

        ndIMG_InitImage (&MyImage);
        ndIMG_LoadImage (&MyImage, "ms0:/lena.bmp", NDMGKNB);

        ndCTRL_CreateButtonSmart   ( 10, 10,        WndHandle, "TheButton",
                                                    "Nanodesktop", "button",
                                                    COLOR_WHITE, COLOR_BLUE, COLOR_LBLUE,
                                                    &MyImage,
                                                    ND_BUTTON_ROUNDED | NDKEY_FONT (4),
                                                    0, 12345678,
                                                    RENDER          );


    }
}
```

The result is this:



Chapter 14

Mouse control

In the previous chapter we have seen many examples about button creation; in all of these we never defined a *button callback.*

A button callback is a routine that is executed any time we press a button. These functions must have the following prototype:

```
ndint64 ButtonCallback (char *StringID, ndint64 InfoField, char WndHandle);
```

When we press a button, the system executes the button callback. This argument is quite complex, so we'll dedicate a lot of time to explaining it.

The first thing to know is this: there are different modes to control the mouse and to check when a button is pressed. The simplest mode is to use the *routines ndProvideMeTheMouse_Until and ndProvideMeTheMouse_Check.* These routines had been introduced in nd 0.3 to simplify things (the direct use of the MouseControl routine is a technical argument, quite complex and it will be studied later).

ndProvideMeTheMouse_Until

When a programmer wants to interact with the interface that he has drawn on the screen, he can use the routine ndProvideMeTheMouse_Until. The thread that contains a call to this routine will be suspended and a mouse pointer will appear on the screen.

The thread that contains the call will remain locked until an *exit condition* happens. When the exit condition happens, the thread that had called ndProvideMeTheMouse will be unlocked by the system and the execution of the thread will continue.

During the waiting (or locking...) time, the mouse can move freely on the screen and the user can press the buttons, activating the different *callbacks.*

Before calling the routine, the user must have declared a variable (it is called *control variable).* It is usual to use only one control variable. The control variable must be *global* and we suggest that it is declared as static. Furthermore, you MUST declare the control variable as *int (32 bit integer).*

The function of the control variable is to say to ndProvideMeTheMouse when the calling thread can be unlocked. The control variable is associated with the exit condition: in effect, the exit condition is a boolean condition that involves a control variable.

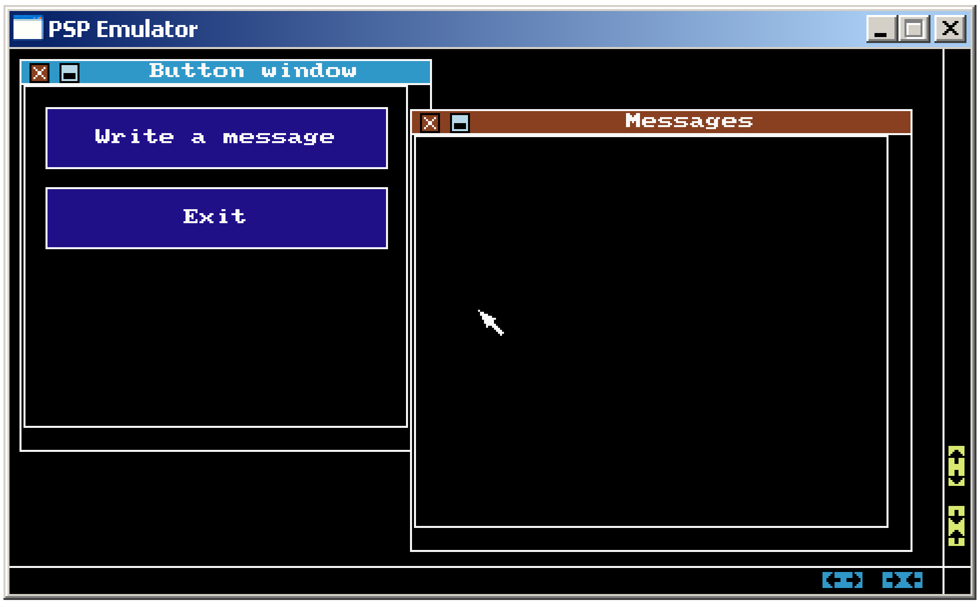
When the user presses a button (or a WMI button, i.e. a menu element, or a close window button), the system executes the callback and after it executes an *exit check.* This check verifies if the exit condition is realized. If it is true, the mouse cycle will be stopped and the thread continues its natural course (usually, the thread continues arresting the program, using functions as ndHAL_SystemHalt ()).

The work of the user must be based on the code of the callbacks. A button that determines the exit from the program, has to change the control variables, so that the exit condition becomes true.

Let's see this source:

```
#include <nanodesktop.h>

int WndHandle0, WndHandle1;
int Button0, Button1;

static int YouCanExit;   // Control variable

// Button callbacks

static ndint64 cbWriteMessage (char *StringID, ndint64 InfoField, char WndHandle)
{
   ndWS_WriteLn ("I have exec WriteMessage", COLOR_WHITE, WndHandle1, RENDER);
}

static ndint64 cbExit (char *StringID, ndint64 InfoField, char WndHandle)
{
   ndWS_WriteLn ("Now you can exit", COLOR_WHITE, WndHandle1, RENDER);
   ndDelay (3);

   YouCanExit=1;    // It makes that ndProvideMouse_Until will be unlocked
}


// Main source

int ndMain()
{
    ndInitSystem();

    WndHandle0=ndLP_CreateWindow (5, 5, 210, 200, "Button window", COLOR_WHITE, COLOR_LBLUE,
                                COLOR_BLACK, COLOR_WHITE, 0);

    WndHandle1=ndLP_CreateWindow (200, 30, 450, 250, "Messages", COLOR_WHITE, COLOR_RED,
                                COLOR_BLACK, COLOR_WHITE, 0);

    ndLP_MaximizeWindow (WndHandle0);
    ndLP_MaximizeWindow (WndHandle1);

    // Creates the buttons

    ndCTRL_CreateButton (10, 10, 180, 40, WndHandle0, "Btn0", "Write a message", "", COLOR_WHITE,
COLOR_BLUE, COLOR_WHITE, 0, &cbWriteMessage, 1234567, NORENDER);

    ndCTRL_CreateButton (10, 50, 180, 80, WndHandle0, "Btn1", "Exit", "", COLOR_WHITE, COLOR_BLUE,
COLOR_WHITE, 0, &cbExit, 0, NORENDER);

    XWindowRender (WndHandle0);

    // Begin a mouse control cycle

    YouCanExit=0;
    ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0); // << System will be locked here

    // The system will be here only after that the Exit button will be pressed

    printf ("The program is exiting....");
    ndDelay (3);
}
```

The part in black color creates two windows (“Button window” and “Messages”) and creates two buttons.



When the system finds a call to *ndProvideMeTheMouse_Until,* the current thread is suspended. The point at which the suspension happens is clearly indicated in the source code. Note that the following instruction would be *printf ("The program is exiting"),* but there is no printf window on this screen. This is because the execution of the current thread has been stopped at the indicated point, before the printf is processed.

The system will remain in this state until the check condition is satisfied. In this example, we have used

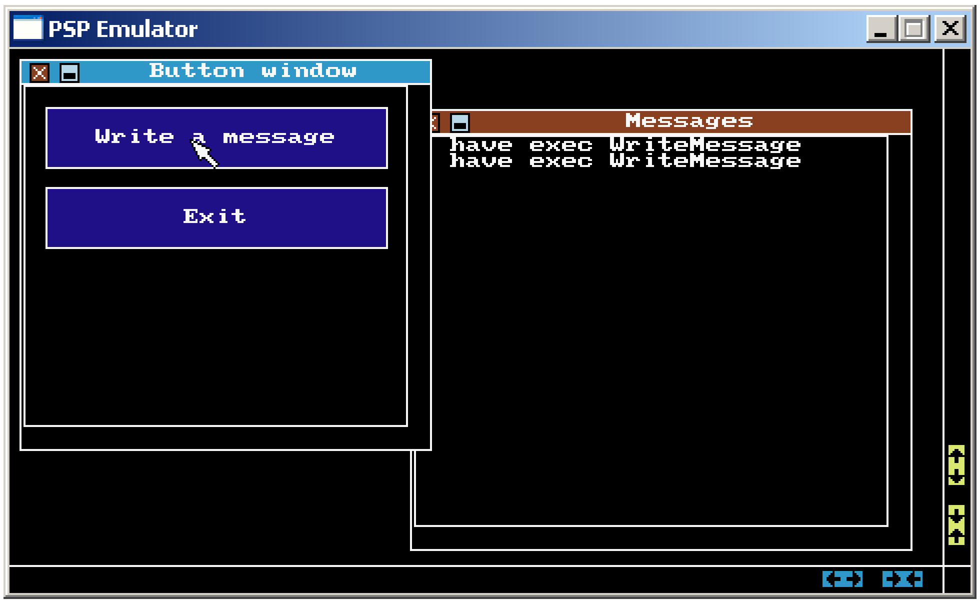
ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0);

Let's analyse this function call. The first parameter is the address of the control variable (that, as we said, must be *static* and *int*). The second and third parameters indicate the logical conditions that must be satisfied by the control variable YouCanExit to unlock the system. In this case, the second and third parameters indicate a logical condition that becomes true only when the control variable became equal to 1.

The fourth and fifth parameter, 0 and 0, normally indicate a second logical condition that must be verified by the control variable. But in this moment, we want to work only with a single logical condition (YouCanExit must be equal to 1 to exit), so the fourth and fifth parameters here are set to zero.

Note that the source has set to 0 the variable YouCanExit, before calling ndProvideMeTheMouse, so that the check condition initially is always *false,* and therefore the mouse pointer can appear.

During these moments, the mouse is on the screen and it is possible to execute button callbacks

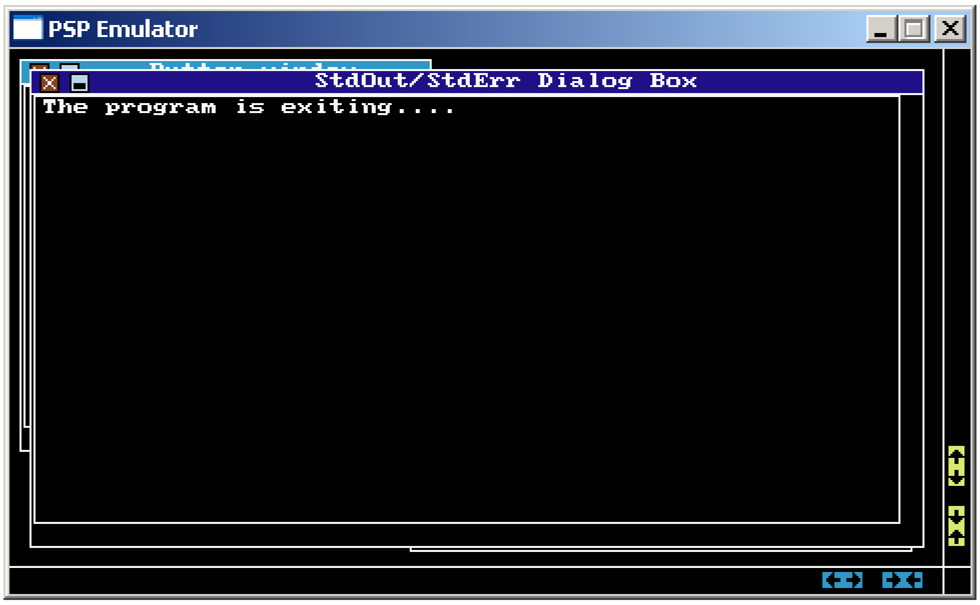


```
have exec WriteMessage
have exec WriteMessage
```

In this image we have pressed twice the button called “write a message”.

Note that the programmer can decide to resume the control of the suspended thread at any moment: it is sufficient that the relative button callback changes the value of a control variable YouCanExit (that is done by cbExit callback).

So, when the user presses the Exit button, the system executes cbExit callback, this sets to 1 the value of the YouCanExit variable, the system checks the exit condition stated by ndProvideMeTheMouse, and afterwards it unlocks the thread, because now the exit condition is true. The following instruction that will be executed by the thread will be `printf ("The program is exiting....");`



```
The program is exiting....
```

The exit condition

Now let's see something about the exit condition. The call to *ndProvideMeTheMouse_Until* was this:

ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0);

The exit condition can involve only an *int* variable. This control variable can be submitted to one or two logical conditions. When you specify two logical conditions, the exit condition is realized only when they are verified together (the two conditions are *and*-ed). The two parameters in red indicate the first logical condition and the two parameters in blue indicate the second logical condition (when there is only one logical condition the parameters in blue must be set to 0;0).

These are the codes that indicate the different logical conditions that can be used:

ND_IS_EQUAL	**1**
ND_IS_DIFFERENT	**2**
ND_IS_GREATER	**3**
ND_IS_GREATER_OR_EQUAL	**4**
ND_IS_SMALLER	**5**
ND_IS_SMALLER_OR_EQUAL	**6**
ND_NOT	**128**

These examples will better explain the use of these codes:

ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0);

Exit condition happens when YouCanExit is equal to 1

ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_DIFFERENT, 3, 0, 0);

Exit condition happens when YouCanExit is different than 3

ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_GREATER, 0, ND_IS_SMALLER, 5);

Exit condition happens when YouCanExit is greater than 0 and is smaller than 5 (0<x<5)

Note also that *ndProvideMeTheMouse_Until* locks only the thread that has called it. The other threads remain unlocked, so it is usual to create a single thread that manages the graphical interface (and that will be locked by this function) while the other threads manage the other functions of the program.

The InfoField parameter

You could think that: if an application needs n different buttons in the interface, the programmer must create n different callbacks, one for each button. This is not always true.

Nanodesktop allows you to pass to a callback a 64-bit value, called InfoField. The InfoField can contain different informations for the callback (contained in a bit-mapped field). The callback doesn't know which routine called it: using the InfoField you can pass to the callback this information. The InfoField allows you also to use a single callback for different buttons: it is sufficient that different buttons call the same callback with different InfoField codes.

The InfoField code that will be passed to the callback must be specified when the button is created.
Let's see this example:

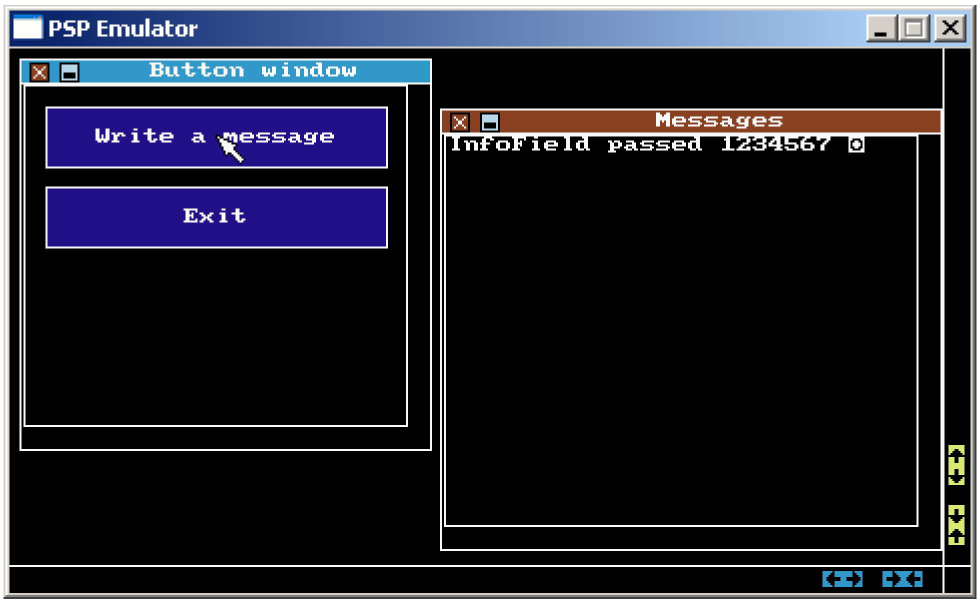
```
#include <nanodesktop.h>

int WndHandle0, WndHandle1;
int Button0, Button1;

static int YouCanExit;   // Control variable

// Button callbacks

static ndint64 cbWriteMessage (char *StringID, ndint64 InfoField, char WndHandle)
{
   ndWS_PrintLn (WndHandle1, COLOR_WHITE, RENDER, "InfoField passed %d \n", (int)(InfoField) );
}

static ndint64 cbExit (char *StringID, ndint64 InfoField, char WndHandle)
{
   ndWS_WriteLn ("Now you can exit", COLOR_WHITE, WndHandle1, RENDER);
   ndDelay (3);

   YouCanExit=1;    // It makes that ndProvideMouse_Until will be unlocked
}
```

```
// Main source

int ndMain()
{

    ndInitSystem();

    WndHandle0=ndLP_CreateWindow (5, 5, 210, 200, "Button window", COLOR_WHITE, COLOR_LBLUE,
                                COLOR_BLACK, COLOR_WHITE, 0);

    WndHandle1=ndLP_CreateWindow (200, 30, 450, 250, "Messages", COLOR_WHITE, COLOR_RED,
                                COLOR_BLACK, COLOR_WHITE, 0);

    ndLP_MaximizeWindow (WndHandle0);
    ndLP_MaximizeWindow (WndHandle1);

    // Creates the buttons

     ndCTRL_CreateButton (10, 10, 180, 40, WndHandle0, "Btn0", "Write a message", "", COLOR_WHITE,
COLOR_BLUE, COLOR_WHITE, 0, &cbWriteMessage, 1234567, NORENDER);
     ndCTRL_CreateButton (10, 50, 180, 80, WndHandle0, "Btn1", "Exit", "", COLOR_WHITE, COLOR_BLUE,
COLOR_WHITE, 0, &cbExit, 0, NORENDER);
    XWindowRender (WndHandle0);

    // Begin a mouse control cycle

    YouCanExit=0;
    ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0);

    // The system will be here only after that the Exit button will be pressed

    printf ("The program is exiting....");
    ndDelay (3);
}
```

This program is identical to the previous one, except for the callback cbWriteMessage. The programmer creates the button “Write a message” and specifies the InfoField 1234567:

```
ndCTRL_CreateButton (10, 10, 180, 40, WndHandle0, "Btn0", "Write a message", "", COLOR_WHITE,
COLOR_BLUE, COLOR_WHITE, 0, &cbWriteMessage, 1234567, NORENDER);
```

This value is passed to the callback cbWriteMessage that prints the message `InfoField passed 1234567`.



```
InfoField passed 1234567
```

More about callbacks

We have seen that when using the InfoField parameter the callback can receive information about the button that has called it. But there are other methods, that allow us to obtain the same result and that frees the InfoField bits for other usages.

As I've said, the prototype of a button callback is this:

```
ndint64 ButtonCallback (char *StringID, ndint64 InfoField, char WndHandle);
```

The StringID parameter is the string that we have specified when we defined the button (see page 100).

WndHandle is the handle of the window that contains the button that has called the callback. Using the pair (StringID:WndHandle), the callback can recognize which button has called it.

Intra-context and extra-context callback

This is complicated, but is absolutely essential to understand how to use the callbacks. It's sometimes necessary that the code contained in the callback recalls, itself, another routine that uses internally MouseControl.

For example, *ndProvideMeTheMouse_Until* and *ndProvideMeTheMouse_Check* internally use MouseControl.

See this source:

```
#include <nanodesktop.h>

int WndHandle0, WndHandle1;
int Button0, Button1;

static int YouCanExit;   // Control variable

// Button callbacks

static ndint64 cbCheckExtraContext (char *StringID, ndint64 InfoField, char WndHandle)
{
   ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0);
}

static ndint64 cbExit (char *StringID, ndint64 InfoField, char WndHandle)
{
   ndWS_WriteLn ("Now you can exit", COLOR_WHITE, WndHandle1, RENDER);
   ndDelay (3);

   YouCanExit=1;    // It makes that ndProvideMouse_Until will be unlocked
}




// Main source

int ndMain()
{

    ndInitSystem();

    WndHandle0=ndLP_CreateWindow (5, 5, 210, 200, "Button window", COLOR_WHITE, COLOR_LBLUE,
                                COLOR_BLACK, COLOR_WHITE, 0);

    WndHandle1=ndLP_CreateWindow (200, 30, 450, 250, "Messages", COLOR_WHITE, COLOR_RED,
                                COLOR_BLACK, COLOR_WHITE, 0);

    ndLP_MaximizeWindow (WndHandle0);
    ndLP_MaximizeWindow (WndHandle1);

    // Creates the buttons

      ndCTRL_CreateButton (10, 10, 180, 40, WndHandle0, "Btn0", "Extra context", "", COLOR_WHITE,
COLOR_BLUE, COLOR_WHITE, 0, &cbCheckExtraContext, 0, NORENDER);
     ndCTRL_CreateButton (10, 50, 180, 80, WndHandle0, "Btn1", "Exit", "", COLOR_WHITE, COLOR_BLUE,
COLOR_WHITE, 0, &cbExit, 0, NORENDER);
    XWindowRender (WndHandle0);

    // Begin a mouse control cycle

    YouCanExit=0;
    ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0);

    // The system will be here only after that the Exit button will be pressed

    printf ("The program is exiting....");
    ndDelay (3);
}
```

Note that there is a callback (cbCheckExtraContext), that calls a routine like ndProvideMeTheMouse_Until, that is one of the routines that internally use MouseControl. While under PSPE, apparently, there is no trouble, under the PSP the behaviour of the application will be wrong. It can seem like a bug: instead, it is normal and it is caused by the Nanodesktop architecture.

When the system tries to execute cbCheckExtraContext callback, the mouse on the screen is halted, and even exiting from the system becomes impossible.

The cause of this behaviour is the following: when a Nanodesktop application is started, a hidden thread, called *Phoenix Mouse thread,* is created and started. The Phoenix Mouse thread executes the small code of the callback, but there is a problem.

If the code of a callback contains a recursive call to MouseControl, or to a routine that uses MouseControl internally (like FileManager or ndProvideMeTheMouse_Until), the code of MouseControl would lock the current thread until the exit condition could be verified. But this current thread would be, in this case Phoenix Mouse thread, so the mouse pointer and after the entire system would be halted.

This technical condition is called by us *MouseControl in MouseControl condition (MCinMC trouble).*

What is the solution ? Fortunately, there is a very simple solution. The user can ask Nanodesktop that the callback is executed in a different thread, created on the fly, when the callback has to be managed. This is possible through the option ND_CALLBACK_IN_NEW_CONTEXT.

See this source:

```
#include <nanodesktop.h>

int WndHandle0, WndHandle1;
int Button0, Button1;

static int YouCanExit;   // Control variable

// Button callbacks

static ndint64 cbCheckExtraContext (char *StringID, ndint64 InfoField, char WndHandle)
{
   ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0);
}

static ndint64 cbExit (char *StringID, ndint64 InfoField, char WndHandle)
{
   ndWS_WriteLn ("Now you can exit", COLOR_WHITE, WndHandle1, RENDER);
   ndDelay (3);

   YouCanExit=1;    // It makes that ndProvideMouse_Until will be unlocked
}




// Main source

int ndMain()
{

    ndInitSystem();

    WndHandle0=ndLP_CreateWindow (5, 5, 210, 200, "Button window", COLOR_WHITE, COLOR_LBLUE,
                                COLOR_BLACK, COLOR_WHITE, 0);

    WndHandle1=ndLP_CreateWindow (200, 30, 450, 250, "Messages", COLOR_WHITE, COLOR_RED,
                                COLOR_BLACK, COLOR_WHITE, 0);

    ndLP_MaximizeWindow (WndHandle0);
    ndLP_MaximizeWindow (WndHandle1);



// Creates the buttons
```

```
     ndCTRL_CreateButton (10, 10, 180, 40, WndHandle0, "Btn0", "Extra context", "", COLOR_WHITE,
COLOR_BLUE, COLOR_WHITE, ND_CALLBACK_IN_NEW_CONTEXT, &cbCheckExtraContext, 0, NORENDER);

     ndCTRL_CreateButton (10, 50, 180, 80, WndHandle0, "Btn1", "Exit", "", COLOR_WHITE, COLOR_BLUE,
COLOR_WHITE, 0, &cbExit, 0, NORENDER);

    XWindowRender (WndHandle0);

    // Begin a mouse control cycle

    YouCanExit=0;
    ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0);

    // The system will be here only after that the Exit button will be pressed

    printf ("The program is exiting....");
    ndDelay (3);
}
```

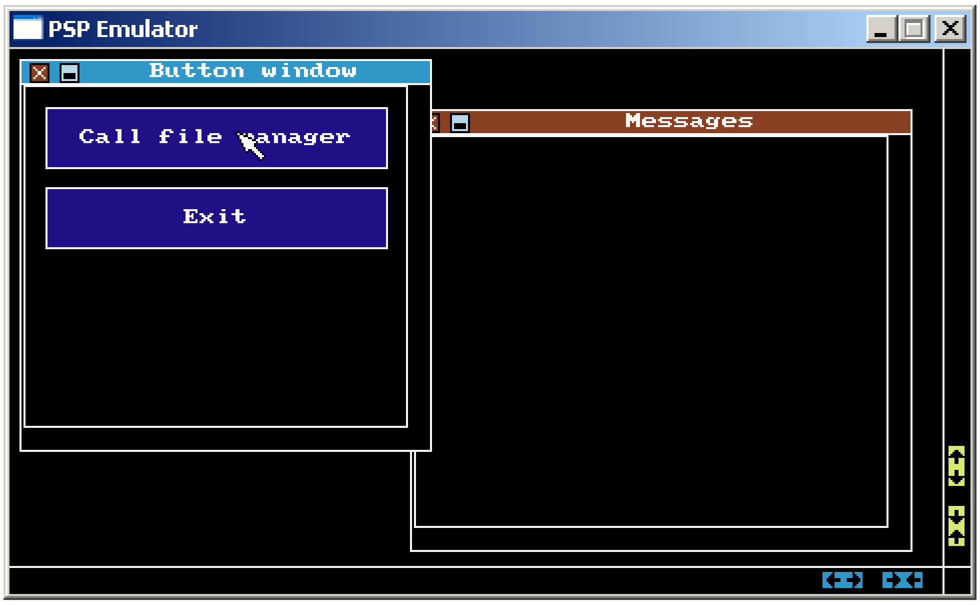
The part in red shows how it is possible to define an *extra-context callback*, i.e. a callback that will be executed by a dedicated thread. If you execute the source now, the previous hang will be removed.

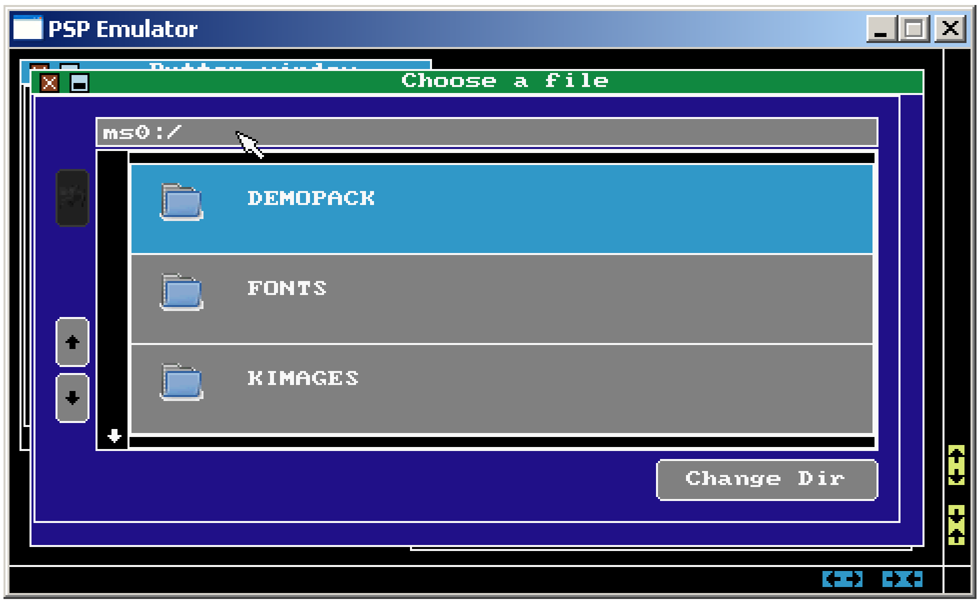
You perhaps think that this is not a very important problem, but it is fundamental: the programmer must be careful that the code executed by a callback doesn't call a function that internally uses a MouseControl routine. If necessary the callback must be executed in a new context.

This source shows another example:

```
#include <nanodesktop.h>

int WndHandle0, WndHandle1;
int Button0, Button1;
char FileChoosen [255];

static int YouCanExit;   // Control variable

// Button callbacks

static ndint64 cbFileManager (char *StringID, ndint64 InfoField, char WndHandle)
{
   FileManager ("Choose a file", 0, 0, &FileChoosen);
}

static ndint64 cbExit (char *StringID, ndint64 InfoField, char WndHandle)
{
   ndWS_WriteLn ("Now you can exit", COLOR_WHITE, WndHandle1, RENDER);
   ndDelay (3);

   YouCanExit=1;    // It makes that ndProvideMouse_Until will be unlocked
}


// Main source

int ndMain()
{

    ndInitSystem();

    WndHandle0=ndLP_CreateWindow (5, 5, 210, 200, "Button window", COLOR_WHITE, COLOR_LBLUE,
                                COLOR_BLACK, COLOR_WHITE, 0);

    WndHandle1=ndLP_CreateWindow (200, 30, 450, 250, "Messages", COLOR_WHITE, COLOR_RED,
                                COLOR_BLACK, COLOR_WHITE, 0);

    ndLP_MaximizeWindow (WndHandle0);
    ndLP_MaximizeWindow (WndHandle1);

    // Creates the buttons

    ndCTRL_CreateButton (10, 10, 180, 40, WndHandle0, "Btn0", "Call file manager", "", COLOR_WHITE,
COLOR_BLUE, COLOR_WHITE, 0, &cbFileManager, 0, NORENDER);
    ndCTRL_CreateButton (10, 50, 180, 80, WndHandle0, "Btn1", "Exit", "", COLOR_WHITE, COLOR_BLUE,
COLOR_WHITE, 0, &cbExit, 0, NORENDER);
    XWindowRender (WndHandle0);

    // Begin a mouse control cycle

    YouCanExit=0;
    ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0);
    // The system will be here only after that the Exit button will be pressed

    printf ("The program is exiting....");
    ndDelay (3);
}
```

This program creates two buttons: the first calls the Nanodesktop FileManager





Under PSPE there is no problem (because the emulator doesn't support multiple threads and so the Phoenix Subsystem is disabled). Try executing the same program under the PSP: the file manager is called but immediately after the mouse pointer is blocked.

To resolve this situation you can define the callback in a new context.

```
#include <nanodesktop.h>

int WndHandle0, WndHandle1;
int Button0, Button1;
char FileChoosen [255];



static int YouCanExit;   // Control variable

// Button callbacks

static ndint64 cbFileManager (char *StringID, ndint64 InfoField, char WndHandle)
{
   FileManager ("Choose a file", 0, 0, &FileChoosen);
}

static ndint64 cbExit (char *StringID, ndint64 InfoField, char WndHandle)
{
   ndWS_WriteLn ("Now you can exit", COLOR_WHITE, WndHandle1, RENDER);
   ndDelay (3);

   YouCanExit=1;    // It makes that ndProvideMouse_Until will be unlocked
}


// Main source

int ndMain()
{

    ndInitSystem();

    WndHandle0=ndLP_CreateWindow (5, 5, 210, 200, "Button window", COLOR_WHITE, COLOR_LBLUE,
                                  COLOR_BLACK, COLOR_WHITE, 0);

    WndHandle1=ndLP_CreateWindow (200, 30, 450, 250, "Messages", COLOR_WHITE, COLOR_RED,
                                  COLOR_BLACK, COLOR_WHITE, 0);

    ndLP_MaximizeWindow (WndHandle0);
    ndLP_MaximizeWindow (WndHandle1);

    // Creates the buttons

    ndCTRL_CreateButton (10, 10, 180, 40, WndHandle0, "Btn0", "Call file manager", "", COLOR_WHITE,
COLOR_BLUE, COLOR_WHITE, ND_CALLBACK_IN_NEW_CONTEXT, &cbFileManager, 0, NORENDER);
    ndCTRL_CreateButton (10, 50, 180, 80, WndHandle0, "Btn1", "Exit", "", COLOR_WHITE, COLOR_BLUE,
COLOR_WHITE, 0, &cbExit, 0, NORENDER);
    XWindowRender (WndHandle0);

    // Begin a mouse control cycle

    YouCanExit=0;
    ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0);

    // The system will be here only after that the Exit button will be pressed

    printf ("The program is exiting....");
    ndDelay (3);
}
```

Routines that use MouseControl internally

From the things that we have said, we can state the following, general rule: when a callback (or a subroutine that is called by the code contained in the callback) contains a call to a *routine that uses internally MouseControl* , the callback **must be defined extra-context**. Otherwise the callback can be defined intra-context (no bitflag required in the button options parameter).

So it is important to know which routines call MouseControl internally. These routines are called *MouseControl based.*

In this version, the only routines that are MouseControl based, are:

MouseControl, ndProvideMeTheMouse_Until, ndProvideMeTheMouse_Check, FileManager.

Note that MouseControl is considered MC-based. So, if a callback contains a direct call to MouseControl, it must be defined extra-context.

Change callback or options

If you are asking yourself is it possible to change the callback assigned to a button that has been already created, the answer is yes. Nanodesktop provides a dedicated function for this, called

```
char ndCTRL_ChangeCallBack (int Callback, ndint64 InfoField, char ButtonHandle, char WndHandle,
char cbOptions);
```

This function allows you to change the callback assigned to a button defined by the pair (ButtonHandle: WndHandle).

For example:

```
ndCTRL_ChangeCallBack (&NewCallback, 0, MyBtnHandle, MyWndHandle, 0);
```

This call changes the callback assigned to a button identified by MyBtnHandle:MyWndHandle.

```
ndCTRL_ChangeCallBack (&NewCallback, 0, MyBtnHandle, MyWndHandle, ND_CALLBACK_IN_NEW_CONTEXT);
```

This is the same as the previous one, but the new callback is defined as an external callback.

There also exists a different version of ChangeCallback, that is able to recognize the button from the pair (StringID:WndHandle) instead of from the pair (BtnHandle:WndHandle). This routine is called

```
char ndCTRL_ChangeCallBackUsingStringID (int Callback, ndint64 InfoField, char *StringID, char
WndHandle, char cbOptions)
```

Return value of the callback

Button callback is a 64-bit function. So, the callback will return a 64-bit value.

```
ndint64 ButtonCallback (char *StringID, ndint64 InfoField, char WndHandle);
```

You may be asking yourself what is the meaning of the return value of the callback. For now, you can ignore this aspect and you can simply put the return value of the callback to 0 (i.e. return from your callback with a simple **return 0**). In fact, if you don't use directly MouseControl routines (you use only indirect routines as *ndProvideMeTheMouse_Until* or *ndProvideMeTheMouse_Check*), the return value of the callback is always ignored.

ndProvideMeTheMouse_Check

As we have seen before, ndProvideMeTheMouse_Until allows you to define an exit condition that can be given by one or two logical conditions that must be verified together. Exit conditions that are more complex, involving two or more control variables, are not as manageable using this function.

Fortunately, in this case Nanodesktop provides a function that can solve the problem: it is called ndProvideMeTheMouse_Check.

```
void ndProvideMeTheMouse_Check (void *CheckCallbackAdr, ndint64 CodeCheck);
```

This routine needs two parameters: the former is the address of a callback, called *exitcheck callback,* and the last is a 64-bit variable, called *CodeCheck.* This last parameter is available for the programmer:

it will be passed to the exitcheck callback when it is executed. It hasn't a precise function: you can use it, for example, to use the *same exitcheck callback* with different *instances of ndProvideMeTheMouse_Check*. If you don't know how to use this value, you can simply set it to 0.

Our attention now has to be focused on exitcheck callback. An exitcheck callback is a routine with the prototype:

```
char CheckCallback (ndint64 CodeCheck, char TypeEvent, char WindowID, char ButtonID, char *StringID,
                    char CallbackExecuted, ndint64 CallbackAnswer, char ZString)
```

This routine has numorous parameters:

- *CodeCheck:* the 64-bit value that has been defined in ndProvideMeTheMouse_Check;
- *TypeEvent:* a code that informs the callback about the type of event that has been realized. The codes are:

  - *SPECIAL_KEY_PRESSED*
    This is used in very rare cases: it means that a special key (like the START button under the PSP), has been pressed. Usually is it is used for internal use only(by ndHighGUI for example). I suggest you program your code to ignore this;

  - *CLOSEWNDCALLBACK_HAS_BEEN_EXEC*
    A close window callback has been executed. WindowID specifies the handle of the window that is relative to the close window button. ButtonID is set to 0

  - *A_WINDOW_WANTS_TO_BE_CLOSED*
    The user has pressed the close window button. The programmer didn't define a close window callback, so the system limits itself to a signal to MouseControl, and this passes the code to the exitcheck callback. In WindowID the specified handle of the window that is relative to the close window button. ButtonID is set to 0

  - *CUSTOM_BUTTON_PRESSED*
    A button has been pressed. It is this case that is used most frequently. WindowID contains the handle of the window that has been pressed, ButtonID contains the handle of the button, StringID contains the address of the StringID that identifies the button. CallbackExecuted here is a boolean value: it is set to 1 if the button has associated a callback and it has been executed. In this case CallbackAnswer is the 64-bit value that has been returned by the callback (we have talked about it on page 115: in this case you cannot ignore it). Zstring is set to void string.

  - *L1WMI_BUTTON_PRESSED*
    An L1 WinMenu element has been selected: in this case WindowID contains the handle of the window that has been selected, ButtonID is set to 0, StringID is set to a void string. CallbackExecuted here is a boolean value: it is set to 1 if the L1WMI has associated a callback and it has been executed. In this case CallbackAnswer is the 64-bit value that has been returned by the callback. Zstring contains the zstring relative to the menu element.

  - *L2WMI_BUTTON_PRESSED*
    An L2 WinMenu element has been selected: in this case WindowID contains the handle of the window that has been selected, ButtonID is set to 0, StringID is set to a void string. CallbackExecuted here is a boolean value: it is set to 1 if the L2WMI has associated a callback and it has been executed. In this case CallbackAnswer is the 64-bit value that has been returned by the callback. Zstring contains the zstring relative to the menu element.

From the information that we have seen, we understand that the exitcheck callback has all the information about each event that happens in the system. Every time that an event happens, Nanodesktop executes the relative callback (if it has been foreseen by the user), and after executes the exitcheck callback to know if the current thread has to be unlocked.

If the exitcheck callback returns a value different than 0, the MouseControl loop will be interrupted, the thread that had called ndProvideMeTheMouse_Check will be unlocked, and the execution of the thread will continue.

The CloseWindow callback

In the windows environment, the selection of a particular button closes a window. This closing operation must be communicated to the program: for example, the program must know if a window is open or closed to avoid that particular window appearing twice on the screen.

Nanodesktop is able to manage the closewindow callbacks. A closewindow callback is a callback that is executed when the closing button of the window is pressed.

The prototype of a closewindow callback is the following:

```
ndint64 CloseWndCallback (char WndHandle, ndint64 WndInfoField);
```

The closewindow callback must be defined by the user. After defining it, the user has to use a particular function to assign a closewindow callback to an open window:

```
char ndLP_SetupCloseWndCallback (void *CloseWndCallback, ndint64 WndInfoField, char
NoNotifyToMouseControl, char WndHandle)
```

The first parameter is the address of the assigned callback, the second parameter is a 64-bit value that is passed to the callback each time it is executed. As usual, this 64-bit code is available for the programmer for various uses: for example, a programmer can use this value to manage different windows with the same callback.

The third parameter is a boolean value. When NoNotifyToMouseControl is set to 1, the system executes the close window callback, but in transparent mode. The event of closing the window will seem completely invisible to the upper layer of the software. Normally, after executing the callback associated with the event of closing a window, Nanodesktop would pass a message to MouseControl or ndProvideMeTheMouse, so that these last routines can verify if it is necessary to exit from the mousecontrol loop. The parameter NoNotifyToMouseControl set to 1 allows us to avoid this check.

The parameter WndHandle defines the window to which the callback has been assigned.

A programming method that is often used is the following. The user defines a variable that keeps track of the fact that a window is open or closed.

When a window is open, the variable is set to 1. So, if the user tries to press a second time the button that opens this window, and the window is already open, nothing happens.

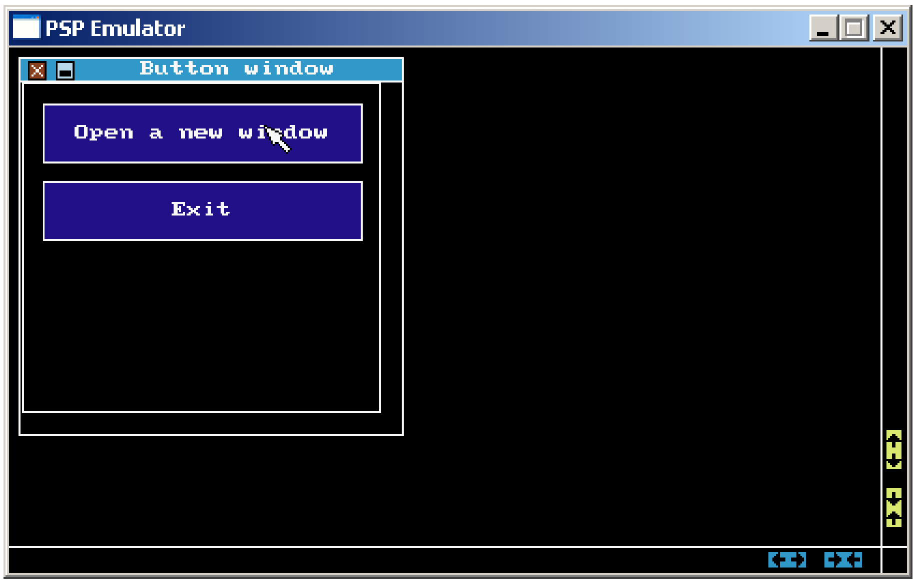
After creating the new window, a call to ndLP_SetupCloseWndCallback assigns to it a dedicated callback. This closes the window and resets the value of the variable to 0.

Let's see a simple example:

```
#include <nanodesktop.h>

int WndHandle0;
int Button0, Button1;

int MyNewWindowHandle;
int MyNewWindowIsOpen = 0;

static int YouCanExit;   // Control variable



// CloseWnd callback

static ndint64 cbCloseNewWindow (char WndHandle, ndint64 WndInfoField)
{
    ndLP_DestroyWindow (WndHandle);
    MyNewWindowIsOpen = 0;          // Now the window can be reopened
}
```

```
// Button callbacks

static ndint64 cbOpenANewWindow (char *StringID, ndint64 InfoField, char WndHandle)
{
    if (!MyNewWindowIsOpen)
    {
        MyNewWindowHandle = ndLP_CreateWindow (280, 100, 450, 200, "My window", COLOR_WHITE,
COLOR_BLUE,
                                    COLOR_GRAY, COLOR_GRAY, 0);

        if (MyNewWindowHandle>=0)   // Creation of the new window successful
        {
            ndLP_MaximizeWindow (MyNewWindowHandle);
            ndLP_SetupCloseWndCallback (&cbCloseNewWindow, 0, 0, MyNewWindowHandle);

            MyNewWindowIsOpen=1;    // Avoid creation of the same window until this isn't closed
        }
    }
}

static ndint64 cbExit (char *StringID, ndint64 InfoField, char WndHandle)
{
   YouCanExit=1;    // It makes that ndProvideMouse_Until will be unlocked
}


// Main source

int ndMain()
{

    ndInitSystem();

    WndHandle0=ndLP_CreateWindow (5, 5, 210, 200, "Button window", COLOR_WHITE, COLOR_LBLUE,
                                COLOR_BLACK, COLOR_WHITE, 0);

    ndLP_MaximizeWindow (WndHandle0);

    // Creates the buttons

    ndCTRL_CreateButton (10, 10, 180, 40, WndHandle0, "Btn0", "Open a new window", "", COLOR_WHITE,
COLOR_BLUE, COLOR_WHITE, 0, &cbOpenANewWindow, 0, NORENDER);
    ndCTRL_CreateButton (10, 50, 180, 80, WndHandle0, "Btn1", "Exit", "", COLOR_WHITE, COLOR_BLUE,
COLOR_WHITE, 0, &cbExit, 0, NORENDER);
    XWindowRender (WndHandle0);

    // Begin a mouse control cycle

    YouCanExit=0;
    ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0);

    // The system will be here only after that the Exit button will be pressed

    printf ("The program is exiting....");
    ndDelay (3);
}
```

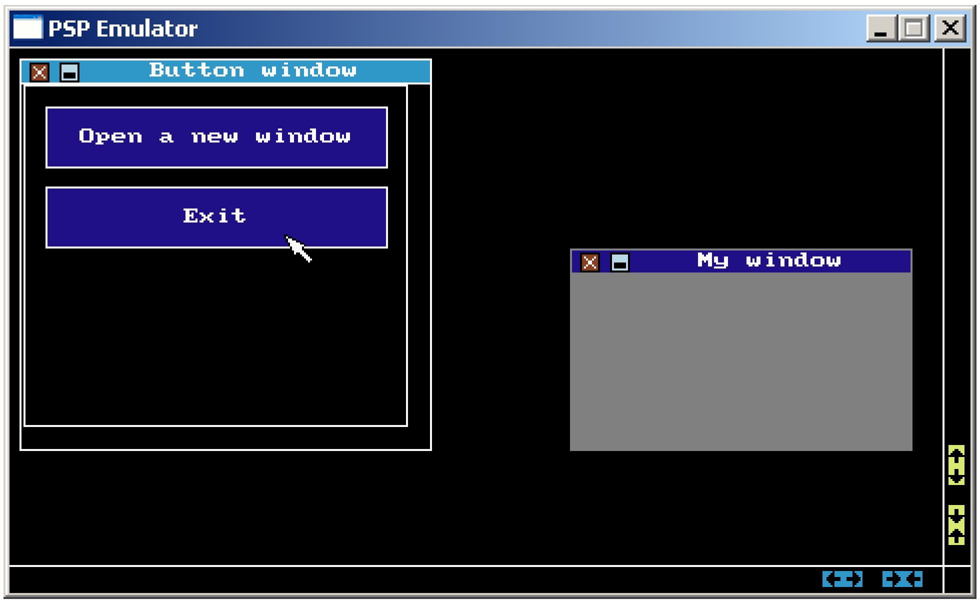
When we start this program, the following appears on the screen:



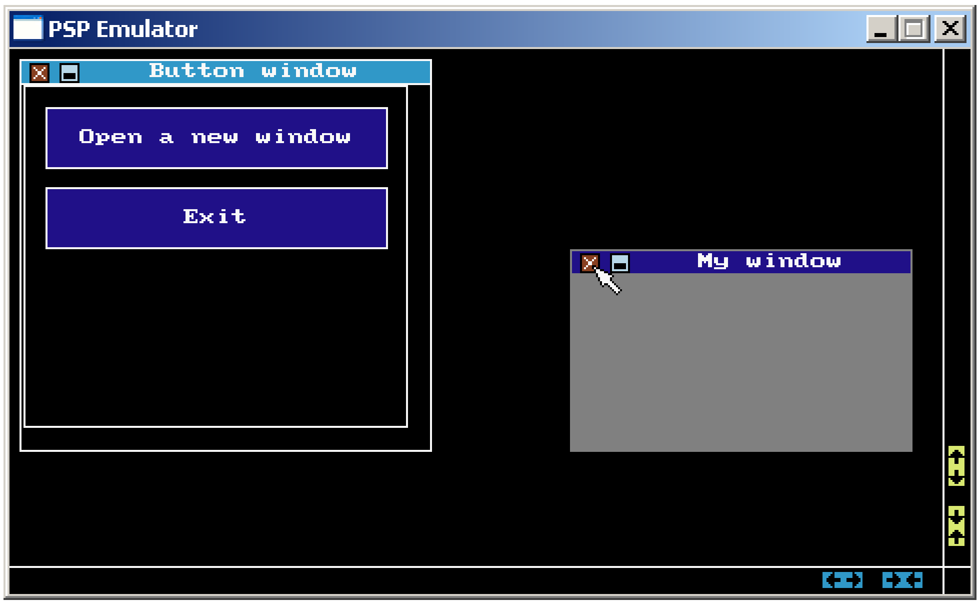
At the beginning of the execution, the variable MyNewWindowIsOpen is set to 0. When we select the first button, the system executes the callback cbOpenANewWindow.

```
static ndint64 cbOpenANewWindow (char *StringID, ndint64 InfoField, char WndHandle)
{
    if (!MyNewWindowIsOpen)
    {
        MyNewWindowHandle = ndLP_CreateWindow (280, 100, 450, 200, "My window", COLOR_WHITE,
COLOR_BLUE, COLOR_GRAY, COLOR_GRAY, 0);

        if (MyNewWindowHandle>=0)    // Creation of the new window successful
        {
            ndLP_MaximizeWindow (MyNewWindowHandle);
            ndLP_SetupCloseWndCallback (&cbCloseNewWindow, 0, 0, MyNewWindowHandle);

            MyNewWindowIsOpen=1;    // Avoid creation of the same window until this isn't closed
        }
    }
}
```

Note that the callback is effective only if the variable MyNewWindowIsOpen is set to 0. The callback sets to 1 this variable after the window is opened, so if you try to select the button a second time it won't have any effect.



The callback also sets the closewnd callback. So, when you press the close button....



the system will execute this callback:

```
// CloseWnd callback

static ndint64 cbCloseNewWindow (char WndHandle, ndint64 WndInfoField)
{
    ndLP_DestroyWindow (WndHandle);
    MyNewWindowIsOpen = 0;           // Now the window can be reopened
}
```

This callback closes the window and resets to 0 the variable MyWindowIsOpen, so that the button can reopen the window if it is pressed again.

Chapter 15

Window menu

Nanodesktop allows you to manage *window menus*. This allows you to create applications that are similar to those that are common under other os-es, like Microsoft Windows or like Linux (for example, under the KDE or Gnome Window Managers).

To create a menu, you have to create a window that accepts menu commands.

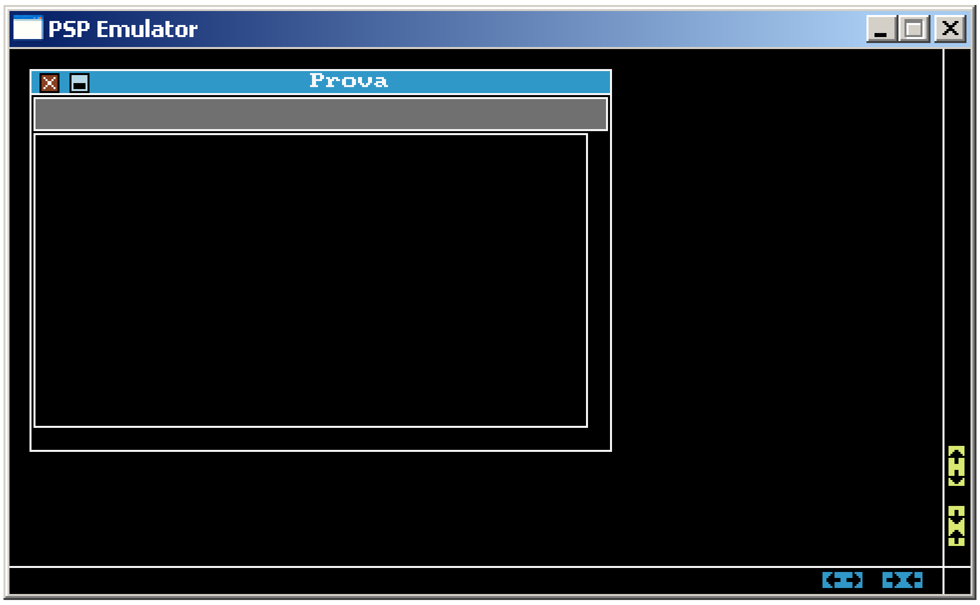
This is simple: we have seen that the prototype of ndLP_CreateWindow is this

```
char ndLP_CreateWindow (unsigned short int _PosWindowX1, unsigned short int _PosWindowY1,
                        unsigned short int _PosWindowX2, unsigned short int _PosWindowY2,
                        char *_WindowTitle,
                        TypeColor _ColorTitle, TypeColor _ColorBGTitle,
                        TypeColor _ColorWindow, TypeColor _ColorBorderWindow,
                        ndint64 Attribute);
```

In chapter 10, we have seen that the 64-bit parameter attribute is able to modify the behaviour of the window. Menu support in a window can be enabled using an appropriate 64-bit key: *MENUSUPPORTED*.

```
#include <nanodesktop.h>

int WndHandle;

int ndMain()
{
    int Counter;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (10, 10, 300, 200, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                 COLOR_BLACK, COLOR_WHITE, MENUSUPPORTED);

    if (WndHandle>=0)
    {
       ndLP_MaximizeWindow (WndHandle);

       // Put here the other commands
    }
}
```

The result is this:



Now we have to define the various *Window Menu Items (WMI in the following).* The first thing to say is that there exist two types of WMI, called WMI1 and WMI2.

WMI1 and WMI2

The WMI1 (WinMenu Item Type 1), is a type of menu that is shown in the menu bar. Each WMI1 can have a submenu, composed by WMI2 (WinMenu Item Type 2).

A question that is asked frequently: can the WMI2 element have submenus comprising of a second, third, fourth level ? The answer, unfortunately, is *no.* This design choice greatly reduces the complexity of Nanodesktop code. In any case, Nanodesktop has been designed for small devices with small screens, so you won't miss the third level WMI2 submenus.

Now, we will try to define 3 WMI1 items. The WMI items (either Type 1 or Type 2) are defined using the function

```
ndWMENU_DefineMenuItem (unsigned char WndHandle, char *ZString, char *NameItem, char Features, void
*CallbackFunction, unsigned char RenderNow);
```

The first parameter is the handle of the window, the second and third parameters are two strings, the *Zstring* and the *NameItem*s, the following parameter is an 8-bit parameter that allows us to set the features of the element. *CallbackFunction* is a pointer to the WMI callback, *RenderNow* has the usual meaning.

The most important parameters are Zstring and NameItem.

Zstring is a string that is associated with the structure of the winmenu. The WMI1 items can have a WMI1 submenu of a second, third, forth, fifth level, in a hierarchical structure. Remember: a WMI2 element cannot have a submenu, but WMI1 elements can.

The Zstring represents this hierarchical structure. For example:

```
FirstBranch

FirstBranch/File

FirstBranch/About

SecondBranch

SecondBranch/Compile

SecondBranch/Options
```

This succession of Zstring defines this hierarchical structure:

```
       FirstBranch(WMI1)

              File(WMI1)

              About(WMI1)

       SecondBranch(WMI1)

              Compile(WMI1)

              Options(WMI1)
```

From Zstring, Nanodesktop can understand the structure that the user wants.

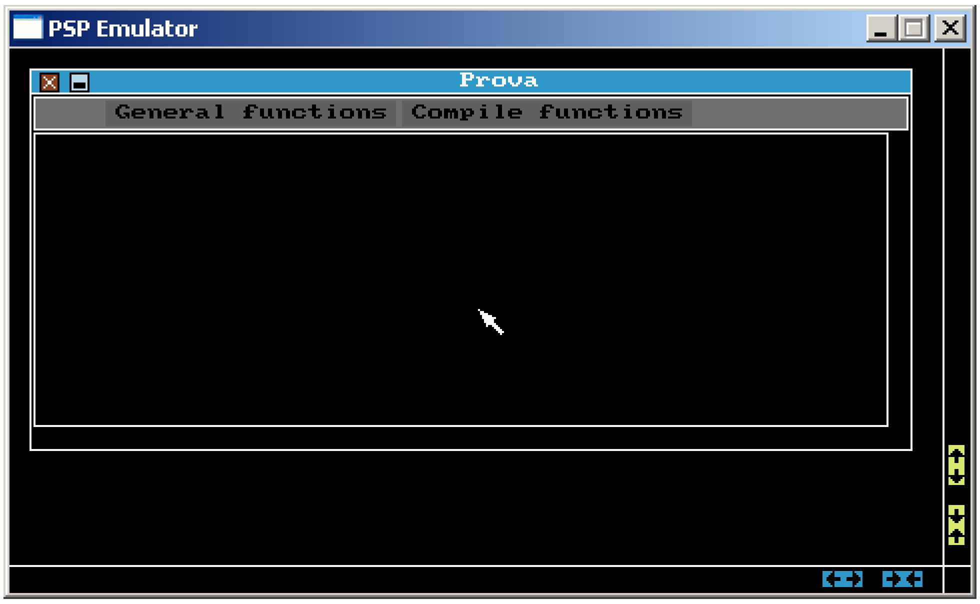
So, if you have already defined a WMI1 element, you can always define a new WMI1 element that is its child, simply using the “/” character.

Note that the Zstring has no connection with the string that will be visualized on the screen: the string that will be visualized is defined by the string NameItem.

See this program:

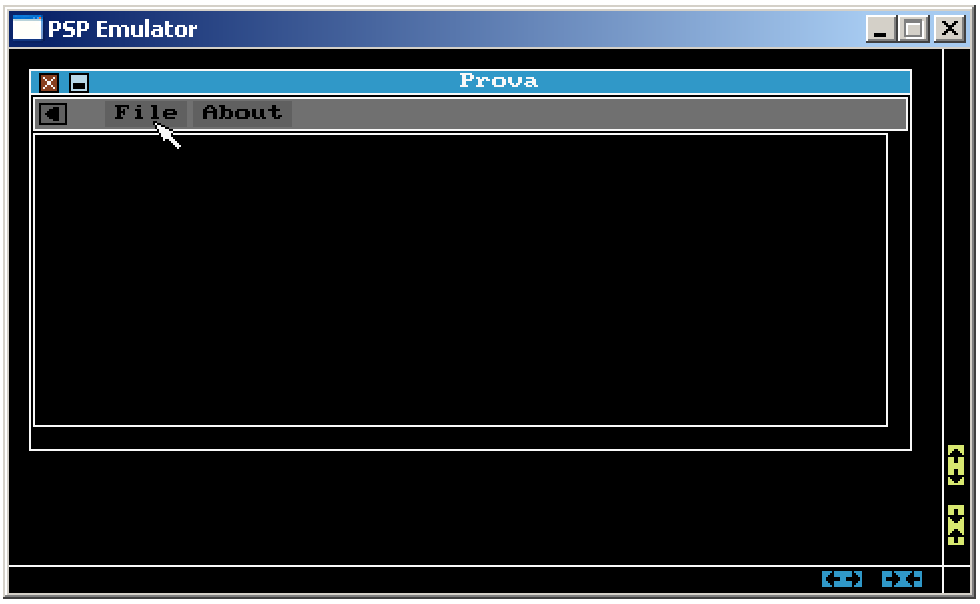
```
#include <nanodesktop.h>

int  WndHandle;
char YouCanExit;

int ndMain()
{
    int Counter;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (10, 10, 450, 200, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                 COLOR_BLACK, COLOR_WHITE, MENUSUPPORTED);

    if (WndHandle>=0)
    {
       ndLP_MaximizeWindow (WndHandle);

       ndWMENU_DefineMenuItem (WndHandle, "FirstBranch",           "General functions", 0, 0,
NORENDER);
       ndWMENU_DefineMenuItem (WndHandle, "FirstBranch/File",      "File", 0, 0, NORENDER);
       ndWMENU_DefineMenuItem (WndHandle, "FirstBranch/About",     "About", 0, 0, NORENDER);

       ndWMENU_DefineMenuItem (WndHandle, "SecondBranch",          "Compile functions", 0, 0,
NORENDER);
       ndWMENU_DefineMenuItem (WndHandle, "SecondBranch/Compile",  "Compile", 0, 0, NORENDER);
       ndWMENU_DefineMenuItem (WndHandle, "SecondBranch/Build",    "Build", 0, 0, NORENDER);

       XWindowRender (WndHandle);

       YouCanExit=0;
       ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0);
    }
}
```

At the higher level, the source defines two branches: FirstBranch and SecondBranch.



The FirstBranch is named *General functions*, the SecondBranch is named *Compile functions.*

When we select the first item, the system switches to the WMI1 submenu that has been defined. For example, in this image we have selected the *General functions* item.



The arrow at the left allows us to return to the higher level menu.

Now, we have to define some WMI2 items. Remember that a WMI1 item can have a WMI1 sub-menu or a WMI2 sub-menu. WMI2 items, instead, cannot use there own sub-menus.

For defining a WMI2 item you use the point "." character. So, the Zstring

```
FirstBranch/File.New

FirstBranch/File.Open

FirstBranch/File.Save
```

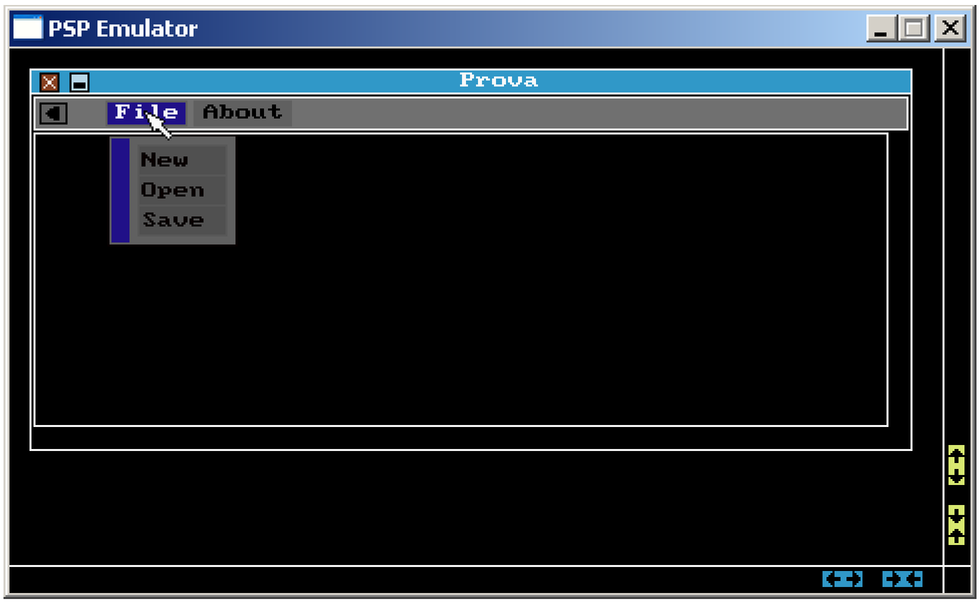
would change the previous hierarchy in the following way:

```
FirstBranch(WMI1)

               File(WMI1)

                       New  (WMI2)

                       Open  (WMI2)

                       Save  (WMI2)

               About(WMI1)

SecondBranch(WMI1)

               Compile(WMI1)

               Options(WMI1)
```

Let's modify the previous program:

```
#include <nanodesktop.h>

int  WndHandle;
char YouCanExit;

int ndMain()
{
    int Counter;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (10, 10, 450, 200, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                 COLOR_BLACK, COLOR_WHITE, MENUSUPPORTED);

    if (WndHandle>=0)
    {
       ndLP_MaximizeWindow (WndHandle);
       ndWMENU_DefineMenuItem (WndHandle, "FirstBranch",           "General functions", 0, 0,
NORENDER);
       ndWMENU_DefineMenuItem (WndHandle, "FirstBranch/File",      "File", 0, 0, NORENDER);

       ndWMENU_DefineMenuItem (WndHandle, "FirstBranch/File.New",       "New", 0, 0, NORENDER);
       ndWMENU_DefineMenuItem (WndHandle, "FirstBranch/File.Open",      "Open", 0, 0, NORENDER);
       ndWMENU_DefineMenuItem (WndHandle, "FirstBranch/File.Save",      "Save", 0, 0, NORENDER);

       ndWMENU_DefineMenuItem (WndHandle, "FirstBranch/About",     "About", 0, 0, NORENDER);

       ndWMENU_DefineMenuItem (WndHandle, "SecondBranch",          "Compile functions", 0, 0,
NORENDER);
       ndWMENU_DefineMenuItem (WndHandle, "SecondBranch/Compile",  "Compile", 0, 0, NORENDER);
       ndWMENU_DefineMenuItem (WndHandle, "SecondBranch/Build",    "Build", 0, 0, NORENDER);

       XWindowRender (WndHandle);

       YouCanExit=0;
       ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0);
    }
}
```

The result is the following:



The WMI1 item File has three WMI2 items (New, Open and Save).

WMI callbacks

The examples that we have shown till now, define the WMI items but without callbacks. A WMI callback (the prototype is the same for WMI1 or WMI2) is a routine of this type:

```
ndint64 CallBack (char WndHandle);
```

For example, we can modify the previous source in this way:

```
#include <nanodesktop.h>

int  WndHandle;
char YouCanExit;


static ndint64 TheNewCallBack (char WndHandle)
{
    printf ("I am here \n");
}
```

```
int ndMain()
{
    int Counter;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (10, 10, 450, 200, "Prova", COLOR_WHITE, COLOR_LBLUE,
                              COLOR_BLACK, COLOR_WHITE, MENUSUPPORTED);

    if (WndHandle>=0)
    {
       ndLP_MaximizeWindow (WndHandle);

       ndWMENU_DefineMenuItem (WndHandle, "FirstBranch",          "General functions", 0, 0,
NORENDER);
       ndWMENU_DefineMenuItem (WndHandle, "FirstBranch/File",     "File", 0, 0, NORENDER);

       ndWMENU_DefineMenuItem (WndHandle, "FirstBranch/File.New",      "New", 0, &TheNewCallBack,
NORENDER);


       ndWMENU_DefineMenuItem (WndHandle, "FirstBranch/File.Open",     "Open", 0, 0, NORENDER);
       ndWMENU_DefineMenuItem (WndHandle, "FirstBranch/File.Save",     "Save", 0, 0, NORENDER);

       ndWMENU_DefineMenuItem (WndHandle, "FirstBranch/About",    "About", 0, 0, NORENDER);

       ndWMENU_DefineMenuItem (WndHandle, "SecondBranch",         "Compile functions", 0, 0,
NORENDER);
       ndWMENU_DefineMenuItem (WndHandle, "SecondBranch/Compile",  "Compile", 0, 0, NORENDER);
       ndWMENU_DefineMenuItem (WndHandle, "SecondBranch/Build",    "Build", 0, 0, NORENDER);

       XWindowRender (WndHandle);

       YouCanExit=0;
       ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0);
    }
}
```

In this example, we have defined the callback TheNewCallBack and we have associated it to the WMI *FirstBranch/File.New*. When the user selects this item, the relative callback is recalled.

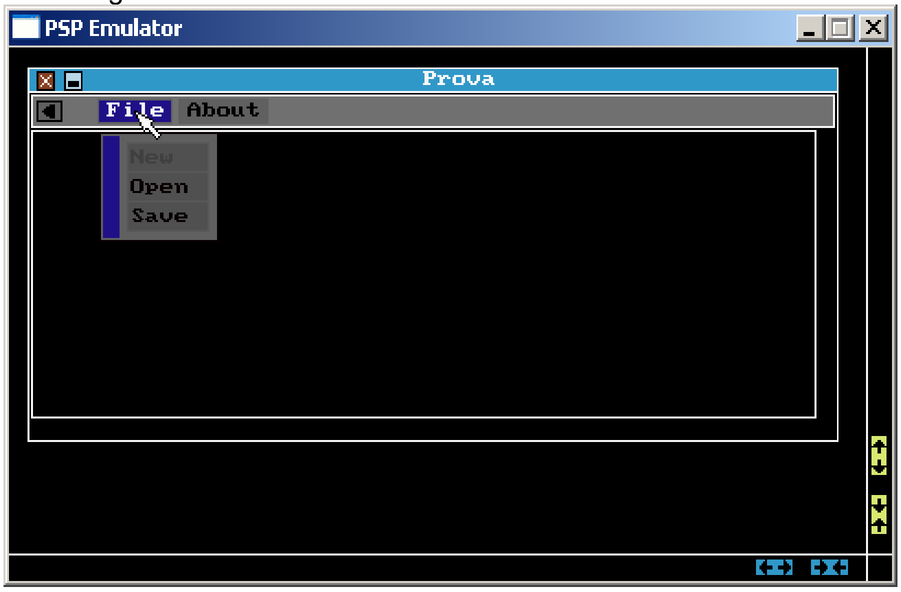
Remember this rule: when a WMI1 item has a sub-menu, the callback that has been associated to it won't be executed. In fact, Nanodesktop executes the switch of the WMI menu, making the menu items of lower levels appear.

Inhibit/Deinhibit the WMI element

A WMI callback can be inhibited, exactly in the same way as a button callback. There are two ways to do this. The former is to declare the inhibition when the WMI item is defined. For example

```
ndWMENU_DefineMenuItem (WndHandle, "FirstBranch/File.New",      "New", ND_WMI_INHIBITED,
&TheNewCallBack,  NORENDER);
```

The result is the following:



The second way to inhibit/deinhibit a WMI element, is to use the functions

```
char ndWMENU_InhibitL1WMI (char WndHandle, char *ZString, char RenderNow);
char ndWMENU_DeInhibitL1WMI (char WndHandle, char *ZString, char RenderNow);

char ndWMENU_InhibitL2WMI (char WndHandle, char *ZString, char RenderNow);
char ndWMENU_DeInhibitL2WMI (char WndHandle, char *ZString, char RenderNow);
```

These functions inhibit/deinhibit an element, after it has been identified using the pair Zstring:WndHandle.

Extra-context callbacks

Also the WMI callbacks are normally managed by the Phoenix Mouse Thread, so they are subjected to the same limitations of the button callbacks, and in particular to the *MCinMC trouble (see page 109).*

Fortunately, Nanodesktop allows you to define the WMI callback as extra-context. This can be obtained through a call like this:

```
ndWMENU_DefineMenuItem (WndHandle, "FirstBranch/File.New",       "New", ND_CALLBACK_IN_NEW_CONTEXT,
&TheNewCallBack,  NORENDER);
```

More information

There are also other functions that aren't explained here for space reasons. You can find more information about these function in the Nanodesktop sources:

(<ndenv>/PSP/SDK/Nanodesktop/src/ndCODE_WinMenuApi.c)

Chapter 16
Keyboards

In various applications there is the necessity to obtain input in the form of a key or a sequence of keys. So, it would be necessary to have a keyboard, but the problem is not so simple. Some devices or platforms, like the PSP or PSPE, don't support a real keyboard, so the entering of a string would be impossible. Other platforms support physical keyboards (like infrared keyboards or usb keyboards), but the programmer has to manage the drivers, the modules and various aspects of the programming.

Nanodesktop offers a complex API that solves many of these problems. It provides *virtual keyboards,* a graphical emulation of a keyboard on the screen, and it supports also, in the platforms that allow this, the managing of real keyboards like the Targus IR keyboard.

The API tries to hide from the programmer the differences between different types of keyboards: an nd application can use virtual keyboards on the PSPE, and an IR keyboard on the PSP *without any change in code.*

getchar

The simplest way to get input from a user, is to use a *standard libc function such as* **getchar**.

```
int getchar(void)
```

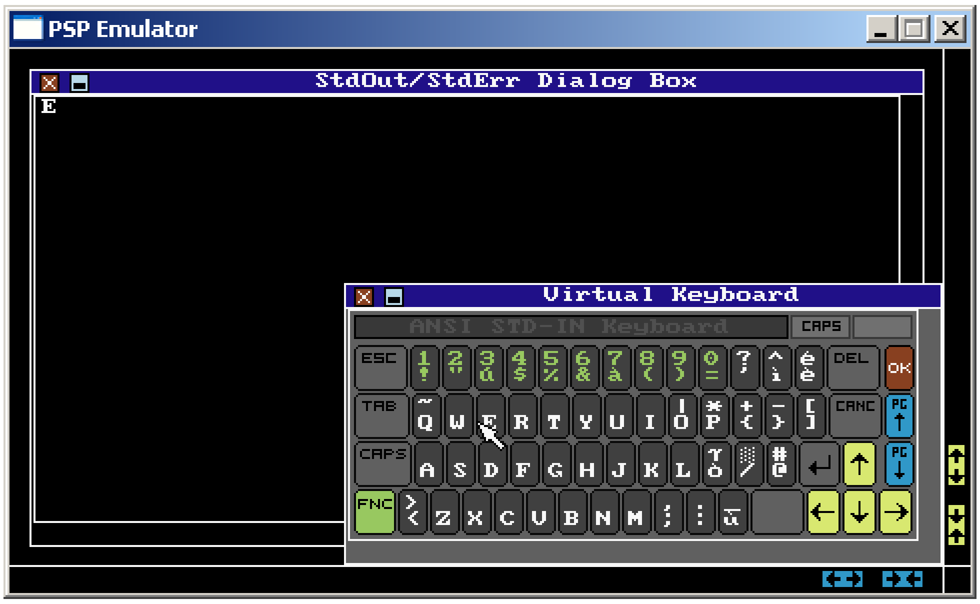
Effectively, the same result could be obtained using other equivalent methods (`getc (stdin)` or `fread (&AdrChar, 1, 1, stdin)`). But the idea is always the same: using a standard libc function that provides an ASCII code to the program.

The methods that use NanoC (standard libc for Nanodesktop) are limited by the normal limitations of standard c: the code returned by getchar is always an 8-bit code that doesn't support special keys or combinations of keys; furthermore, these routines don't support *callbacks* and the keyboard that is used is normally the one predefined by the API (but this value can be changed by a proper routine, see below).

This source shows the simple use of getchar:

```
int ndMain ()
{
    int Counter;
    char x;

    ndInitSystem ();

    for (Counter=0; Counter<5; Counter++)
    {
        x = getchar ();
    }
}
```

This is the result:



```
E
```

Note two things. The former is that getchar always opens a *StdOut/StdErr dialog box;* the second is that getchar outputs the chars that have been typed by the user. The virtual keyboard is disabled any time the system accepts a new key and all keycodes are 8-bit

fread (stdin)

An alternative way to solve part of the previous problem is to use fread on stdin stream.

```
int ndMain ()
{
     int Counter;
     char x;

     ndInitSystem ();

     for (Counter=0; Counter<5; Counter++)
     {
          fread (&x, 1, 1, stdin);
     }
}
```

This code is equivalent to that which we have seen before. Now, let's modify the code in the following way:

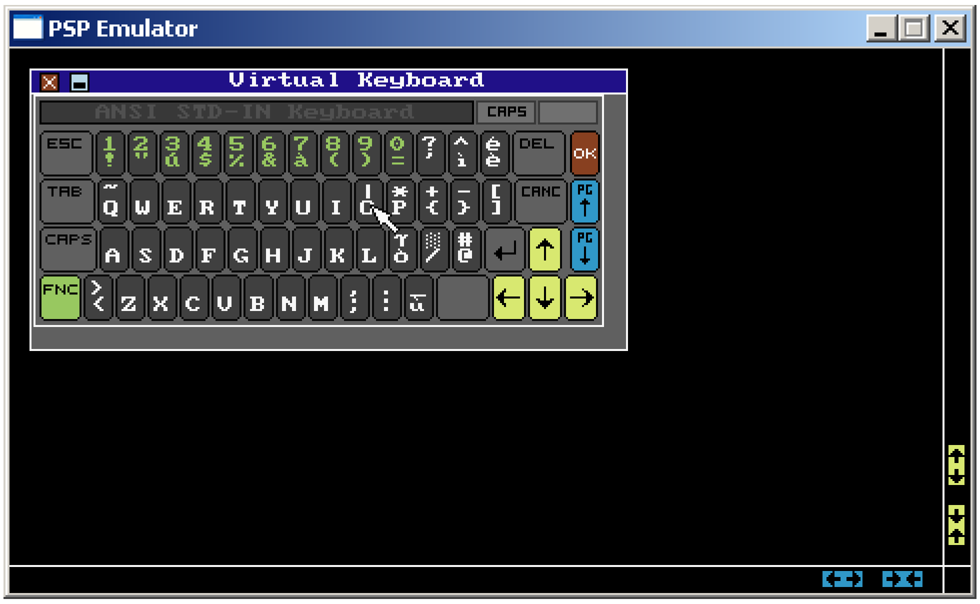
```
int ndMain ()
{
     int Counter;
     char x;

     ndInitSystem ();

     for (Counter=0; Counter<5; Counter++)
     {
          fread (&x, 1, -1, stdin);
     }

     ndCloseSystemKeyboard ();
}
```

The negative value in red does not have a meaning in the standard libc. This is an extension typical of Nanodesktop: the negative sign says to nd that the virtual keyboard isn't to be closed automatically

when a single char is received. The closing of the keyboard is manually executed by the routine *ndCloseSystemKeyboard* ();
Now lets see some different code:

```
int ndMain ()
{
    int Counter;
    char x;

    ndInitSystem ();

    for (Counter=0; Counter<5; Counter++)
    {
        fread (&x, -1, -1, stdin);
    }

    ndCloseSystemKeyboard ();
}
```

The negative value in blue is another extension of nd: it disables the echo in the StdOut window.

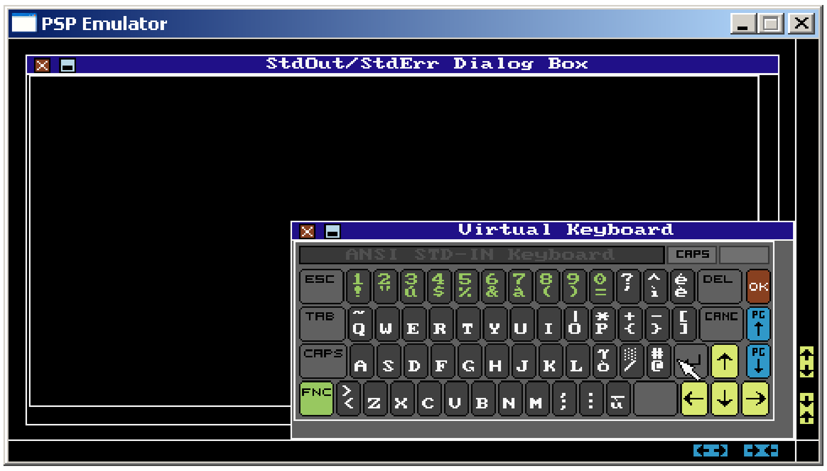


scanf

Another way to request a string is to use another standard libc function: *scanf*.

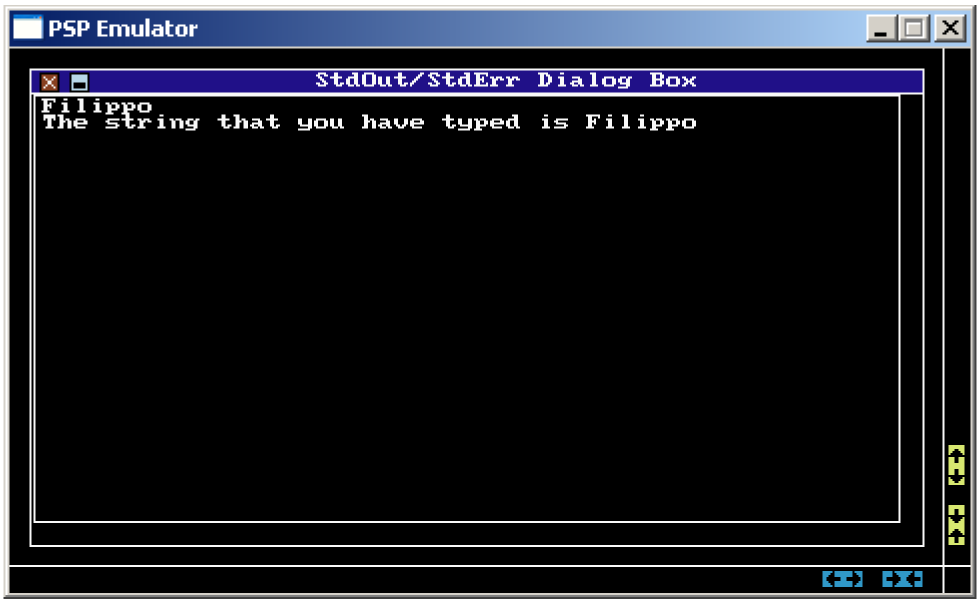
```
int ndMain ()
{
    char Stringa [255];

    ndInitSystem ();

    scanf ("%s ", Stringa);
    printf ("The string that you have typed is %s \n", Stringa);
}
```

Result:



scanf calls the standard keyboard of the system and captures each char. The echo is always enabled, for compatibility reasons with the ANSI C standard.

When you press ENTER, scanf returns control to the caller routine:



```
Filippo
The string that you have typed is Filippo
```

ndSET_SystemKeyboardType

The routines seen up till now don't allow you to explicitly declare what keyboard type you want. Nanodesktop uses only a standard type of keyboard for *all the* requests from the routines that belong to the standard C library. You can change the type of keyboard that is used when the system encounters the next standard C instruction, using the routine

```
char ndSET_SystemKeyboardType (int Type);
```

Where Type is the 32-bit value that identifies the type of keyboard.

ndHAL_VKB_GetASingleKey

However, the main problem of the routines that collect the chars and that are standardized by C is this: the code of each char is a simple 8 bit code. So, some special chars can be managed (for example, 0x10 is backspace), but the larger part remain unmanageable, such as the combination of keys or the special keys too.

If you want to manage the full power of the systems keyboard provided by Nanodesktop, you must program at a lower level, i.e using the *nanodesktop VKB calls.*

For Nanodesktop, each key has a 32 bit code that identifies it and that is called *keycode.* Each 32 bit is divided in to three fields: the former is composed by the more significant 16 bits and it is called *FIELDSK (field for special key)*, the second and the third are 8-bit fields called *Hi* and *Lo.*

XX XX XX XX

FIELDSK Hi Lo

FieldSK is bit-mapped. The more significant bit is always set to 0 (if it is 1, this means that there is an error). The other bits, instead, are assigned to the special-keys of the real keyboards (a bit for CTRL, a bit for ALT, a bit for FUNC etc.). The virtual keyboards, instead, don't emit a FIELDSK, so these bits are always set to 0 when you use a virtual keyboard.

The Hi code indicates the *type* of key. There are predefined constants that define the values that can be assumed by the Hi code. They are:

ND_GENERIC_SPECIALKEY (CTRL-ALT-ESC etc)

ND_GENERIC_FUNCKEY (FUNC key + letter, LoCode contains ASCII code of the letter)

ND_KEY_HAS_BEEN_RELEASED (some real keyboards generate codes also when a key is released and not only when it is pressed)

ND_NORMAL_KEY (0)(user has pressed a number or a letter)

The Lo code has three meanings.

- If the user has pressed a normal key, the LoCode is the ASCII code of the number or of the letter;
- if the user has pressed a combination of keys, composed by a letter and a special key, the LoCode indicates the letter in the combination;
- if the user has pressed only a special key, the LoCode indicates the key that has been pressed.

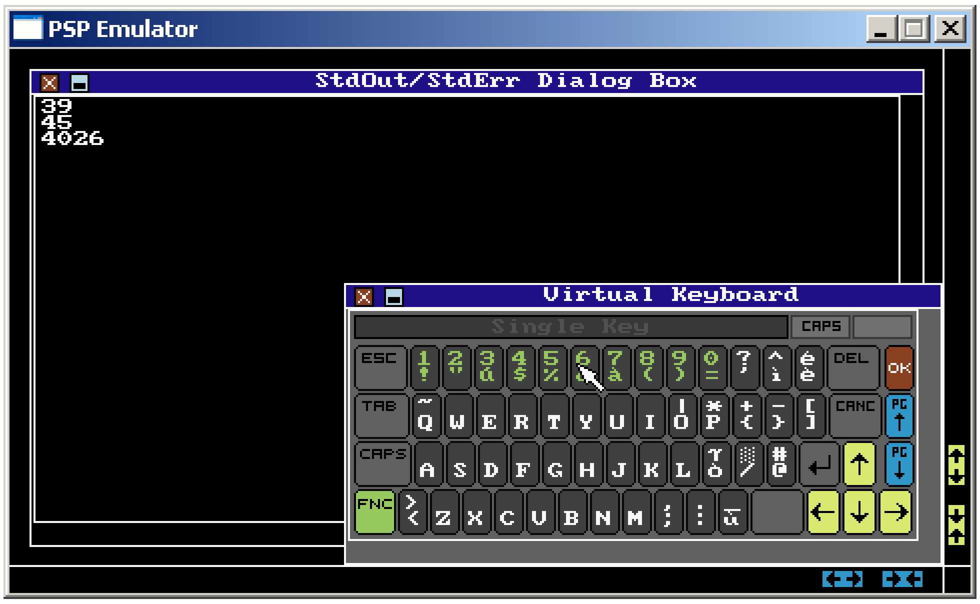
ndHAL_VKB_GetASingleKey suspends the current thread until the user has pressed a key; when this happens, it returns the 32-bit code of a single key to the caller thread and resumes it.

The codes returned by *ndHAL_VKB_GetASingleKey*, are always positive; if they are negative, it means that there is an error.

This is a simple example about the use of the function:

```
int ndMain ()
{
    int Counter;
    char Stringa [255];

    ndInitSystem ();

    for (Counter=0; Counter<10; Counter++)
    {
       printf ("%X \n", ndHAL_VKB_GetASingleKey ("Single Key", 0));
    }
}
```

The result is this:



```
39
45
4026
```

The first two buttons that have been pressed were letters. The third code indicates a combination FUNC + key.

This function always closes the virtual keyboard when a key is pressed: in the loop the virtual keyboard is continuously opened and closed, reopened and reclosed. To solve this problem, you have to use a more complex function, called **ndHAL_VKB_AssignMeKeyboard**

ndHAL_VKB_AssignMeKeyboard

All functions that we have seen till now are very simplified. Internally, Nanodesktop manages the keyboards through *keyboard instances.* You can have one or n keyboard instances at the same time in your program: for example, an application can have two text-boxes: if the user asks to open both text boxes, the system will have to manage two instances for the two virtual keyboards (and in fact, two virtual keyboards will appear on the screen).

A keyboard instance is opened and after some time it is closed. The function that opens a keyboard instance is called ndHAL_VKB_AssignMeKeyboard.

```
int ndHAL_VKB_AssignMeKeyboard (char *MessageToUser, int Type, ndint64 SystemOptions,
                                     ndint64 KeyboardStyle,
                                     void *KeyCallback,   void *KeyCallbackData,
                                     void *EndCallback,   void *EndCallbackData,
                                     void *BreakCallback, void *BreakCallbackData)
```

Now, we can see the meaning of all parameters.

MessageToUser is a string: this is the message that will be visualized in the window that contains the virtual keyboard assigned by the system. It is important that a virtual keyboard has a message on it, because there can be more keyboards on the screen at the same time, and in this case the user wouldn't know which keyboard corresponds to which element of the interface. When you choose a real keyboard, nd will ignore the MessageToUser parameter.

The parameter Type is a 32-bit value that identifies the type of keyboard that has to be assigned by the system. If you specify 0, Nanodesktop will use the standard keyboard type.

SystemOption is a 64-bit parameter that is bit-mapped. It is very important, and we'll talk about it soon.

KeyboardStyle is a 64-bit parameter that is used to pass some other parameters to the different virtual keyboards: normally you can set it to 0. In any case, particular values passed through this parameter allow us to program the virtual keyboard with the settings that you want.

KeyCallback, EndCallback, and BreakCallback are three pointers to three callbacks, that will be recalled during the normal work of the virtual keyboard.

The SystemOptions parameter

SystemOptions is a 64-bit parameter: it allows you to decide upon the behaviour of the keyboard. There are some predefined constants that can be passed or or-ed to SystemOptions value: the result is a set of features for the instance. There are 3 constants:

ND_KEYBOARD_CALLBACK_BASED
ND_ASK_ONLY_A_KEY
ND_KEYBOARD_NO_AUTOCLOSE

The behaviour of the keyboard is mainly based upon the use of the option **ND_KEYBOARD_CALLBACK_BASED.**

For example, this is a request to open an instance based on a callback based keyboard:

```
   ndHAL_VKB_AssignMeKeyboard ("Prova", 0, ND_KEYBOARD_CALLBACK_BASED | ND_ASK_ONLY_A_KEY,
                                        0,      KeyCallback,    KeyCallbackData,
                                                EndCallback,    EndCallbackData,
                                                BreakCallback, BreakCallbackData)
```

This is a request to open an instance, based on a ordinary keyboard (ordinary means not callback based):

```
   ndHAL_VKB_AssignMeKeyboard ("Prova", 0, ND_ASK_ONLY_A_KEY,
                                        0,      KeyCallback,    KeyCallbackData,
                                                EndCallback,    EndCallbackData,
                                                BreakCallback, BreakCallbackData)
```

Ordinary keyboards

Now, we'll see something about ordinary keyboards (the keyboards that are not callback based).

When a routine requests to the nd system the opening of an ordinary keyboard instance, the caller thread is suspended and the control is transferred to the vkeyboard code until the loop of chars requested is finished.

So, we can say that ordinary keyboards have a *blocking* behaviour: the execution of the thread is suspended till the procedure of sending keycodes has been terminated by the user.

Well, let's see what are the things that will happen when we launch ndHAL_VKB_AssignMeKeyboard:

a) the routine looks for a free vkb handle in the system. If it doesn't find this, the routine fails and the control returns immediately to the caller routine with a negative error code;

b) the routine assigns the vkb handle, so that this is reserved now. The keyboard instance is opened now and it will be closed only, automatically, by the system (when some condition happens), or manually by the programmer through functions such as *ndHAL_VKB_DestroyKeyboard*

c) *the caller thread is suspended*

d) the system begins to request the keys. Each time that a key is pressed, Nanodesktop executes the *keycallback* (if the user has defined one). The keycallback is a routine with the prototype:

```
void KeyCallback (int KeyPressedCode, void *KeyCallbackData, int VKBHandle, char WndHandle)
```

When the programmer calls ndHAL_VKB_AssignMeKeyboard, he can define a pair of parameters called KeyCallback and KeyCallbackData.

KeyCallback is a pointer to a callback that has been prepared by the programmer to manage the new keycode. The programmer can also disable this callback, setting to 0 the address passed to the routine.

KeyCallbackData is a pointer to a memory area prepared by the programmer. The pointer to this structure is received by the callback. The programmer can use this pointer for various uses: for example, it allows you to manage different virtual keyboards at the same time using the same code for all key callbacks.

Each time that the user presses a key, the system recalls the key callback. It must manage the new keycode i.e. it must execute the task that the programmer has foreseen when he was writing the source. Key callback routines receive the following information: the 32-bit code associated with the pressed key, the pointer KeyCallbackData, the handle of the virtual keyboard, and the handle of the window that contains the virtual keyboard (for a real keyboard, WndHandle is set to -1).

e) after you have managed the new keycode, the system must understand if the loop that requests the various keys can be terminated. This can be established only by the programmer: in fact, he is the only one that knows when the chars that he has collected are sufficient.

So Nanodesktop executes a second callback that must be provided by the programmer: the *end callback.* The end callback is a routine with the prototype:

```
char EndCallback (void *EndCallbackData, int VKBHandle, char WndHandle)
```

The end callback is assigned by the user through a parameter of ndHAL_VKB_AssignMeKeyboard. In reality, the programmer can also disable the end callback, passing 0 to the routine as a parameter: in this case the loop will continue forever (this is not completely true because the user could interrupt the loop using a particular sequence of commands in the BreakCallback code).

The user can also declare a pointer called EndCallbackData: this will be passed to the end callback. Using this pointer, the programmer can pass to the end callback particular information, necessary to understand if it is time to exit. The EndCallbackData pointer allows you also to manage different virtual keyboards at the same time with the same code.

The callback receives the following data: the EndCallbackData pointer, the VKBHandle, that is the handle of the virtual keyboard, and the WndHandle, that is the handle of the window that contains the virtual keyboard (for real keyboards this is -1).

If the end callback returns a value equal to 1 (or in any case different than 0), the process of requesting the keys will be interrupted and the system will continue with step f), otherwise the system will restart a new passage of the loop and a new key will be collected.

If the user needs only to collect a single key, he can avoid defining an explicit end callback. It is sufficient to specify the option ND_ASK_ONLY_A_KEY. In this case, the end callback will be ignored and the system will exit only after having collected a single key.

f) the loop is interrupted.

g) Normally, if you have requested a virtual keyboard (and not a real keyboard), when exiting, Nanodesktop disables the virtual keyboard (i.e. the keyboard is removed from the screen, and the vkb handle is made free).

The user, however, can also create virtual keyboards in which it is possible to exit without closing the relative instance. In this case, the control returns to the caller thread, *but the vkb handle remains busy and the virtual keyboard remains on the screen*.

This is useful because these instances can be reused again using particular routines such as *ndHAL_VKB_ReUseAnOpenedKeyboard. This particular option (quite rare...), can be obtained using the option **ND_KEYBOARD_NO_AUTOCLOSE**.*

h) <u>Nanodesktop resumes the caller thread</u> and returns a 32-bit code that contains some informations.

The 32-bit code that has been returned by *ndHAL_AssignMeKeyboard* is quite complex. It is always positive, because a negative value would indicate an error condition.

If the operation was successful, the less significant 8 bits (7-0 bits: they are called the *AlphaField*) of the returned code contain the vkb handle that had been assigned.

Be careful that this code has a real meaning only when you have specified the option NO_AUTOCLOSE: in fact, if you haven't done this, at the moment that the control returns to your thread, the keyboard handle has already been used and closed....

The more significant 24 bits of the returned code (32-8 bits: they are called the *BetaField*), contain the number of chars that have been collected by the routine.

The break callback

There is also a BreakCallback. This is a routine like this:

```
void BreakCallback (void *BreakCallbackData, int VKBHandle, char WndHandle)
```

The programmer usually doesn't define this type of callback, and sets to 0 the relative parameters in *ndHAL_AssignMeKeyboard*. But this type of callback can be useful.

BreakCallback is a routine that is executed by the system when the user presses the closebutton of the window that contains the virtual keyboard. It is executed also with real keyboards when selecting a particular escape sequence.

In some cases the application can intercept the request of close the virtual keyboard on the screen: BreakCallback is used for this scope. BreakCallback is ignored when you use the options *ASK_ONLY_A_KEY* or *ND_KEYBOARD_NO_AUTOCLOSE*.

Manage some keycodes

This table shows some keycodes (the table considers only the less significant 16-bits of the value: the most significant bits are ignored)

Key	**Lower 16-bit code**
Esc	0x80FF
Tab	0x8008
Delete	0x8010
Return	0x8020
Ok	0x8021
PageUP	0x8071
PageDOWN	0x8071
Arrows	0x8061, 0x8062, 0x8063, 0x8064

A note that is applicable *only for Virtual Keyboard 1.* This keyboard provides a set of FUNC key combinations

When the FUNC switch is activated each key returns a special key which starts with 0x40. This code can be recognized by the developer in a simple way: to recognize it, you have to do this bit operation:

```
([code] & 0xFF00) == 0x4000
```

Ordinary keyboards: an example

As we have seen till now, the use of ndHAL_VKB_AssignMeKeyboard is very, very complex. Here we have an example of the usage of the function.

This is source code written by **Daniele Colanardi**: it collects some keycodes and manages them through a dedicated callback

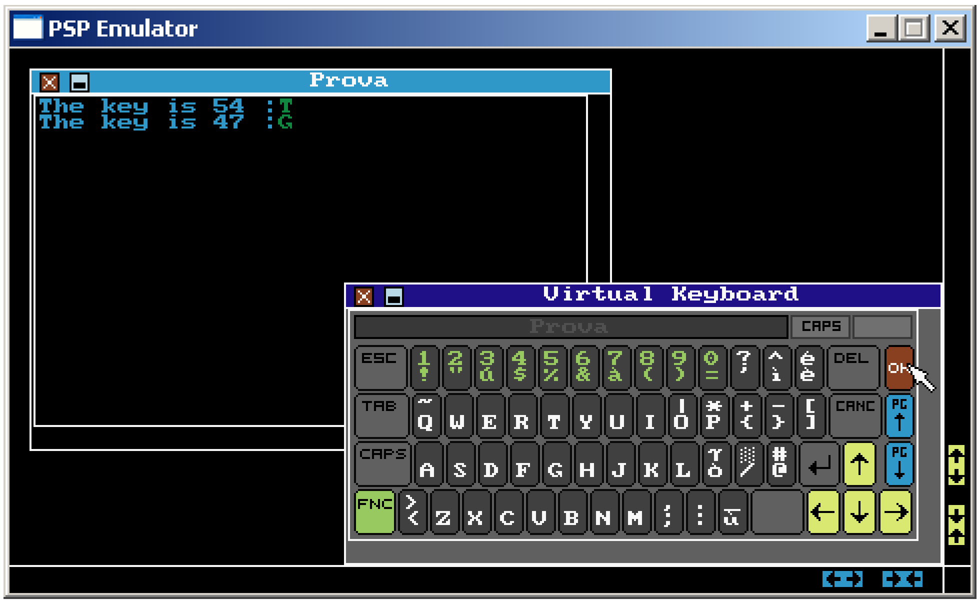
```
#include <nanodesktop.h>

int ndMainHandle;


struct MyInfoStruct_Type
{
       char myStr [256];
       int charCount;
       int LastCode;
};


static void KeyCall (int KeyPressedCode, void *KeyCallbackData, int VKBHandle, char WndHandle)
{
    struct MyInfoStruct_Type *MyInfoStruct = KeyCallbackData;
    // Recover the address of info struct

    if ((KeyPressedCode & 0xFF00)!=0)
    {
        if ((KeyPressedCode & 0xFF00)==0x4000)
           ndWS_WriteLn("Function+Btn pressed", COLOR_LBLUE, MainHandle, RENDER);

        switch (KeyPressedCode)
        {
           case 0x80FF:
                ndWS_WriteLn("Esc pressed, closing keyboard...", COLOR_LBLUE, MainHandle, RENDER);
                break;
           case 0x8008:
                ndWS_WriteLn("Tab Pressed", COLOR_LBLUE, MainHandle, RENDER);
                break;
           case 0x8010:
                ndWS_WriteLn("Delete pressed", COLOR_LBLUE, MainHandle, RENDER);
                break;
           case 0x8020:
                ndWS_WriteLn("Return Pressed", COLOR_LBLUE, MainHandle, RENDER);
                break;
```

```
            case 0x8021:
                 ndWS_WriteLn("Ok Pressed, closing keyboard...", COLOR_LBLUE, MainHandle, RENDER);
                 break;
            case 0x8071:
                 ndWS_WriteLn("PageUP Pressed", COLOR_LBLUE, MainHandle, RENDER);
                 break;
            case 0x8072:
                 ndWS_WriteLn("PageDOWN Pressed", COLOR_LBLUE, MainHandle, RENDER);
                 break;
            case 0x8061:
            case 0x8062:
            case 0x8063:
            case 0x8064:
                 ndWS_WriteLn("Arrows Pressed", COLOR_LBLUE, MainHandle, RENDER);
                 break;
         }

         MyInfoStruct->LastCode=KeyPressedCode;
      }
      else
      {
         ndWS_Print (MainHandle, COLOR_LBLUE, RENDER, "The key is %x :", KeyPressedCode);
         ndWS_WriteLn(&KeyPressedCode, COLOR_GREEN, MainHandle, RENDER);

         MyInfoStruct->myStr[MyInfoStruct->charCount] = KeyPressedCode;
         MyInfoStruct->charCount ++;
      }
}

static char EndCall (void *EndCallbackData, int VKBHandle, char WndHandle)
{
      struct MyInfoStruct_Type *MyInfoStruct = EndCallbackData;
      // Recover the address of info struct

      if (MyInfoStruct->LastCode==0x80FF || MyInfoStruct->LastCode==0x8021)
      {
         //If Esc or Ok was pressed...

         MyInfoStruct->myStr[MyInfoStruct->charCount] = "\0";
         return 1;
      }
      else
         return 0;
}

static void BreakCall (void *BreakCallbackData, int VKBHandle, char WndHandle)
{
       ndWS_WriteLn("Keyboard closed by window's close button", COLOR_BLUE, MainHandle, RENDER);
}

int ndMain()
{
    ndInitSystem();

    MainHandle=ndLP_CreateWindow (10, 10, 300, 200, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                  COLOR_BLACK, COLOR_WHITE, 0);

    if (MainHandle>=0)
    {
        ndLP_MaximizeWindow (MainHandle);

        struct MyInfoStruct_Type MyInfoStruct;
        MyInfoStruct.charCount = 0;
        memset (&(MyInfoStruct.charCount), 0, 254);

        int kh=ndHAL_VKB_AssignMeKeyboard("Prova",1,0,0,&KeyCall, &MyInfoStruct, &EndCall,
                   &MyInfoStruct, &BreakCall, &MyInfoStruct);

        if (kh<0)
        { //Error!
            kh = -kh;
            int code = -(kh & 0xFF); //Alpha field, handle or error code
            int secCode = (kh>>8) & 0xFFFFFF; //Beta field, chars count or sec. error code

            char msg2[100];
            char *msg;

            switch (code)
            {
               case ERR_NO_VKB_AVAILABLE:
                  msg="No Keyboard Available";
                  break;
               case ERR_VKB_WRONG_TYPE:
                  msg="Wrong type";
                  break;
               case ERR_I_CANNOT_CREATE_VKB:
                  msg= "Can't create keyboard";
                  break;
               default:
                  sprintf(msg, "Unkonw error: %d", code);
            }

            ndWS_WriteLn (msg, COLOR_RED, MainHandle, RENDER);
            ndWS_PrintLn (MainHandle, COLOR_RED, RENDER,"Secondary code: %d", secCode);
        }
        else
        {
            ndWS_PrintLn (MainHandle, COLOR_BLUE, RENDER,"Keyboard handle was: %d",(kh & 0xFF));
            ndWS_PrintLn (MainHandle, COLOR_BLUE, RENDER,"Total char: %d",((kh>>8) & 0xFFFFFF));

            ndWS_WriteLn (MyInfoStruct.myStr, COLOR_RED, MainHandle, RENDER);
        }
    }
}
```

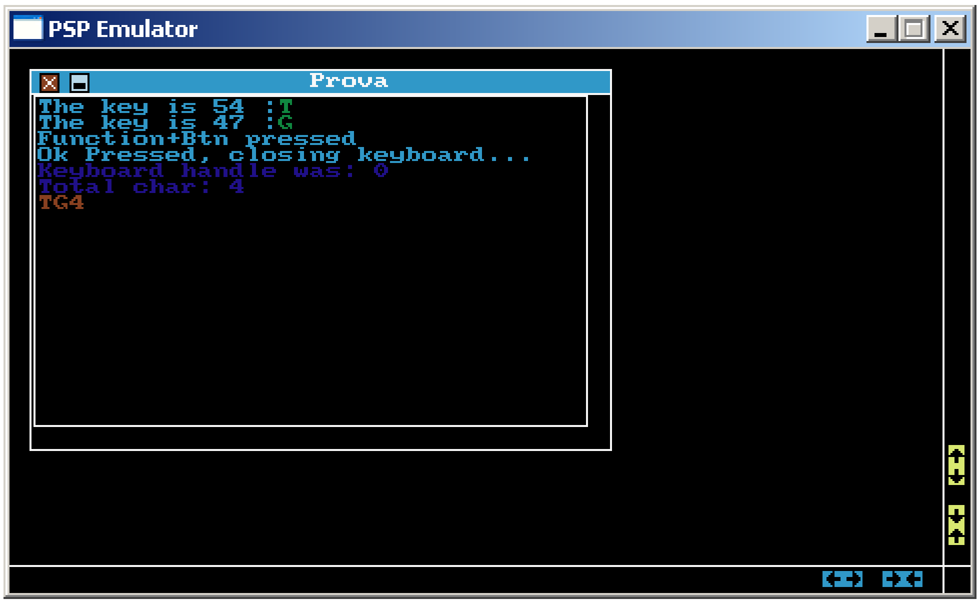
Be careful about the solutions that have been adopted. The programmer defines a struct t`MyInfoStruct` that contains the information needed by the callbacks to correctly work (see parts in red). So, when we pass the address of the struct through the pointers KeyCallbackData, EndCallbackData, BreakCallbackData, the respective callbacks can recover the address of MyInfoStruct and they can access the data they need (see the part in green).

In this struct, there are the following fields: *myStr* contains the string in which the chars will be copied by the callback, *charCount* is a counter that is incremented each time the key callback is called, *LastCode* is a 32-bit variable that contains the last code received.

When the user presses a key, the system recalls the key callback: it copies the char in myStr [charCount] (or manages the special key), updates the charCount index and saves a copy of the code in the variable *LastCode* (see part in cyan). This last operation is necessary so when the end callback will be recalled, it will be able to see the LastCode value, and it will be able to decide if it is opportune to exit (the exit happens if the user has pressed OK or ESC, respectively codes 0x8021 or 0x80FF).



```
The key is 54 :T
The key is 47 :G
```



```
The key is 54 :T
The key is 47 :G
Function+Btn pressed
Ok Pressed, closing keyboard...
Keyboard handle was: 0
Total char: 4
TG4
```

Callback based keyboards

Now we'll talk about keyboards that have been defined as *callback based*. We have seen that the behaviour of an ordinary keyboard is *blocking:* the thread that has called *ndHAL_AssignMeKeyboard* is blocked until the end condition doesn't happen.

Well, there are applications that need a *not blocking* behaviour: the keyboard is activated, and after this it remains independent from the caller thread.

The caller thread creates the keyboard and continues its work with its following instructions with no interruption: caller thread and keyboard remain independent.

The answer to this need is a *callback-based keyboard.* A keyboard can be defined cb-based, using an appropriate explicit constant in the 64-bit Options parameter:

```
   ndHAL_VKB_AssignMeKeyboard ("Prova", 0, ND_KEYBOARD_CALLBACK_BASED,
                                          0,     KeyCallback,    KeyCallbackData,
                                                 EndCallback,    EndCallbackData,
                                                 BreakCallback,  BreakCallbackData)
```

Let's see the events that happen when you start a cb-based keyboard:

a) the routine looks for a free vkb handle in the system. If it doesn't find it, the routine fails and the control returns immediately to the caller routine with a negative error code;

b) the routine assigns the vkb handle, so that this is reserved now. The keyboard instance is opened now. For virtual keyboards Nanodesktop creates the keyboard on the screen; for real keyboards, Nanodesktop creates a new thread that manages the new keys and that executes the key-callbacks

c) the control is immediately returned to the caller thread: exit code returned is the vkb handle of the keyboard (only the less significant 8-bits have meaning, **the others are set to 0**). The virtual keyboard that *has been created remains on the screen; for real keyboards, the thread remains in memory.*

As you can imagine, the real work in a cb-based keyboard is done by the code in the callbacks. When the user presses the closewindow button of the virtual keyboard (or presses a combination of keys that is equivalent to exit on the real keyboard), the system will execute the break-callback (an application can utilize this callback to be notified that the user has closed the cb-based keyboard), and after this it automatically disables the keyboard.

Be careful: the autoclosing or autodisabling of the keyboard won't be executed if you define the option *ND_KEYBOARD_NO_AUTOCLOSE*

```
ndHAL_VKB_AssignMeKeyboard ("Prova",0, ND_KEYBOARD_CALLBACK_BASED| ND_KEYBOARD_NO_AUTOCLOSE,
                                          0,     KeyCallback,    KeyCallbackData,
                                                     EndCallback,   EndCallbackData,
                                                     BreakCallback, BreakCallbackData)
```

This source code is equivalent to the previous program, but it uses a callback based keyboard. Note that it is necessary to call *ndProvideMeTheMouse_Until* to lock the thread while the keyboard is independently on the screen in the same moment

```
#include <nanodesktop.h>

int ndMainHandle;
int YouCanExit;


struct MyInfoStruct_Type
{
       char myStr [256];
       int charCount;
       int LastCode;
};


static void KeyCall (int KeyPressedCode, void *KeyCallbackData, int VKBHandle, char WndHandle)
{
    struct MyInfoStruct_Type *MyInfoStruct = KeyCallbackData;
    // Recover the address of info struct

    if ((KeyPressedCode & 0xFF00)!=0)
    {
        if ((KeyPressedCode & 0xFF00)==0x4000)
           ndWS_WriteLn("Function+Btn pressed", COLOR_LBLUE, MainHandle, RENDER);

        switch (KeyPressedCode)
        {
           case 0x80FF:
                ndWS_WriteLn("Esc pressed, closing keyboard...", COLOR_LBLUE, MainHandle, RENDER);
                break;
           case 0x8008:
                ndWS_WriteLn("Tab Pressed", COLOR_LBLUE, MainHandle, RENDER);
                break;
           case 0x8010:
                ndWS_WriteLn("Delete pressed", COLOR_LBLUE, MainHandle, RENDER);
                break;
           case 0x8020:
                ndWS_WriteLn("Return Pressed", COLOR_LBLUE, MainHandle, RENDER);
                break;
           case 0x8021:
                ndWS_WriteLn("Ok Pressed, closing keyboard...", COLOR_LBLUE, MainHandle, RENDER);
                break;
           case 0x8071:
                ndWS_WriteLn("PageUP Pressed", COLOR_LBLUE, MainHandle, RENDER);
                break;
           case 0x8072:
                ndWS_WriteLn("PageDOWN Pressed", COLOR_LBLUE, MainHandle, RENDER);
                break;
           case 0x8061:
           case 0x8062:
           case 0x8063:
           case 0x8064:
                ndWS_WriteLn("Arrows Pressed", COLOR_LBLUE, MainHandle, RENDER);
                break;
        }

        MyInfoStruct->LastCode=KeyPressedCode;
     }
     else
     {
        ndWS_Print (MainHandle, COLOR_LBLUE, RENDER, "The key is %x :", KeyPressedCode);
        ndWS_WriteLn(&KeyPressedCode, COLOR_GREEN, MainHandle, RENDER);

        MyInfoStruct->myStr[MyInfoStruct->charCount] = KeyPressedCode;
        MyInfoStruct->charCount ++;
     }
}

static char EndCall (void *EndCallbackData, int VKBHandle, char WndHandle)
{
     struct MyInfoStruct_Type *MyInfoStruct = EndCallbackData;
     // Recover the address of info struct



     if (MyInfoStruct->LastCode==0x80FF || MyInfoStruct->LastCode==0x8021)
     {
        //If Esc or Ok was pressed...

        MyInfoStruct->myStr[MyInfoStruct->charCount] = "\0";
        return 1;
     }
     else
        return 0;
}

static void BreakCall (void *BreakCallbackData, int VKBHandle, char WndHandle)
{
       ndWS_WriteLn("Keyboard closed by window's close button", COLOR_BLUE, MainHandle, RENDER);
}

int ndMain()
{
    ndInitSystem();

    MainHandle=ndLP_CreateWindow (10, 10, 300, 200, "Prova", COLOR_WHITE, COLOR_LBLUE,
                                  COLOR_BLACK, COLOR_WHITE, 0);

    if (MainHandle>=0)
    {
        ndLP_MaximizeWindow (MainHandle);

        struct MyInfoStruct_Type MyInfoStruct;
        MyInfoStruct.charCount = 0;

        int kh= ndHAL_VKB_AssignMeKeyboard("Prova",1,ND_KEYBOARD_CALLBACK_BASED,0,&KeyCall,
               &MyInfoStruct,&EndCall, &MyInfoStruct,&BreakCall,&MyInfoStruct);

        YouCanExit=0;
        ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0);
    }
}
```

ndHAL_VKB_ReUseAnOpenedKeyboard

When you define an ordinary keyboard using the option ND_KEYBOARD_NO_AUTOCLOSE, you obtain a keyboard that hasn't been closed when the char loop is terminated.

So your application may want to reuse a keyboard that is already open: nd allows this operation through the routine ndHAL_VKB_ReUseAnOpenedKeyboard

```
int ndHAL_VKB_ReUseAnOpenedKeyboard (ndint64 SystemOptions, int VKBHandle)
```

VKBHandle is the handle that identifies the opened keyboard. SystemOptions has the usual meaning.

ndHAL_VKB_DestroyKeyboard

If you have opened a virtual keyboard on the screen, you can manually close it at any moment using the routine ndHAL_VKB_DestroyKeyboard:

```
char ndHAL_VKB_DestroyKeyboard (char VKBHandle)
```

VKBHandle is the handle that identifies the opened keyboard.

Set the position of a virtual keyboard

Normally, when you start a virtual keyboard using ndHAL_VKB_AssignMeKeyboard, nd uses its *position generator,* to determinate automatically the best position for the window. However, you can force a particular position using a particular 64-bit key in the KeyboardStyle option.

For example, if you want to create a cb based keyboard that has the left upper corner at pixel (100;100), you can use:

```
ndHAL_VKB_AssignMeKeyboard("Virtual Keyboard",1,ND_KEYBOARD_CALLBACK_BASED,
                           NDKEY_POSX (100) | NDKEY_POSY (100),
                           &KeyCall,&MyInfoStruct,
                           &EndCall, &MyInfoStruct,
                           &BreakCall,&MyInfoStruct);
```

ndHAL_VKB_ThisKeybTypeIsAvailable

This function returns 1 if a particular keyboard type is available on the platform:

```
ndHAL_VKB_ThisKeybTypeIsAvailable (int Type)
```

Chapter 17
Textboxes/TextAreas

Now we'll talk about TextBoxes/TextAreas. These elements are provided to allow the user to enter a text string.

The function that creates a textbox on the screen is called *ndTBOX_CreateTextArea.* This function creates the textbox in a window and, if all is ok, it returns the *textbox handle*, a number that identifies the textbox in the following operations. If the routine returns a negative value instead, it means that there is an error. You can find more information about the error codes in the Nanodesktop sources:

(<ndenv>/PSP/SDK/Nanodesktop/src/ndCODE_TextBox.c)

Let's see the prototype of ndTBOX_CreateTextArea:

```
int ndTBOX_CreateTextArea    (unsigned short int PosX1, unsigned short int PosY1,
                              unsigned short int PosX2, unsigned short int PosY2,
                              char *DescriptorString, ndint64 Options, void *TextData,
                              TypeColor TextColorOn,  TypeColor BgTextColorOn,
                              TypeColor TextColorOff, TypeColor BgTextColorOff,
                              TypeColor TextColorInh, TypeColor BgTextColorInh,
                              void *cbProcessValueBefore, void *cbProcessValueForValidation,
                              void *cbProcessValueAfter,  void *ProcessValueData,
                              char WndHandle, char RenderNow)
```

This prototype is very complex, and so we'll have to explain the meaning of each parameter in great detail.

We'll begin by saying that the textbox always has a rectangular shape. The position of the textbox is defined by the first four parameters (PosX1, PosY1, PosX2, PosY2).

Each textbox also has a name that is defined by the *DescriptorString.* This name is shown in the VirtualKeyboard, so the user can know in which textbox he is entering the chars, when he is writing in two textboxes at the same time.

The color pairs (TextColorOn:BgTextColorOn) (TextColorOff:BgTextColorOff) (TextColorInh:BgTextColorInh), represent the colors that have been assumed by the textbox when the textbox is activated, when it is deactivated, and when it is inhibited.

WndHandle identifies the window in which the textbox must be created, and RenderNow has the usual meaning.

The char dimensions of the TextDataArray

When we create a textbox, we have to indicate, in some way, the *char dims* of the textarea, i.e the number of chars in x-dimension and in y-dimension that constitute the array that is associated to the textbox.

This array is called *TextDataArray* and it is very important. When a user types a string in the textbox it is stored in the TextDataArray; when a program wants to read the strings that the user has entered in the box, it access to the memory area occupied by TextDataArray. Each textbox always has a TextDataArray that is associated with it.

The dimensions in bytes of a TextDataArray, is correlated to the char dims by the relation:

TextDataArray_Size = (NrCharsY+1)*(NrCharsX+1)

Be careful that the TextDataArray, when it has been allocated *explicitly by the user, must always follow the previous rule about the size: a bidimensional array of (NrCharsX+1)*(NrCharsY+1) bytes.*

It is very important to specify in an unambiguous way the char dims of a textbox, and to allocate the area of memory for the TextDataArray in the correct size.

When the programmer doesn't explicitly specify the char dims, Nanodesktop simply executes a calculation based on the width and the height (in pixels) of the textbox.

NrCharsX and NrCharsY are obtained automatically by these formulas:

NrCharsX = ((abs (PosX2-PosX1) – 4) / (sDimCharX+0)) - 1;
NrCharsY = ((abs (PosY2-PosY1) – 4) / (sDimCharY+2));

where sDimCharX and sDimCharY are usually set to 8 (except that you are using a TTF font, a function that will be introduced in a successive version of nd).

For example, a textbox of 200 pixels in width and of 20 pixels in height, if the programmer does not specify explicitly the char dims of the TextDataArray, has associated char dims:

NrCharsX = (196 /   8) – 1 = 23
NrCharsY = (16   / 10)   = 1

And the relative TextDataArray would be allocated with an instruction like this:

```
NrCharsX=23;
NrCharsY=1;
TextDataArray = malloc  ( (NrCharsX+1)*(NrCharsY+1)  );
memset (TextDataArray, 0,  (NrCharsX+1)*(NrCharsY+1));
```

The user can also define explicitly the char dims: in this case Nanodesktop should scroll the text in the textbox, because the rows and the columns are greater that those visable..

The char dims are defined through a key in the 64-bit Option parameter:

```
NDKEY_SETTEXTAREA (NrCharsX, NrCharsY).
```

For example, this is a call that defines a TextDataArray of 40 * 30 chars. Note the key that defines the char dims in explicit mode.

```
    ndTBOX_CreateTextArea    (10, 10, 200, 40, "Test textbox", NDKEY_SETTEXTAREA (40, 30), 0,
                              COLOR_BLACK,  COLOR_WHITE,
                              COLOR_BLACK,  COLOR_GRAY,
                              COLOR_BLACK,  COLOR_GRAY03,
                              NULL, NULL, NULL,
                               0,
                              WndHandle, NORENDER)
```

The allocation in ram of the TextDataArray: internal mode and external mode

Note also that in the previous example there is a 0 (evidenced in red) passed as a parameter to TextData.

The TextData parameter is a pointer to ram. In theory, the programmer should allocate an area in ram that is sufficiently large, and after he should pass the pointer as a TextData parameter.

So, for example, a correct model would be this:

```
void *TextDataArray;

TextDataArray = malloc (41*31);
memset (TextDataArray, 0, 41*31);

ndTBOX_CreateTextArea    (10, 10, 200, 40, "Test textbox",
                             NDKEY_SETTEXTAREA (40, 30), TextDataArray,
                              COLOR_BLACK,  COLOR_WHITE,
                              COLOR_BLACK,  COLOR_GRAY,
                              COLOR_BLACK,  COLOR_GRAY03,
                              NULL, NULL, NULL,
                               0,
                              WndHandle, NORENDER)
```

This approach is called *external mode.* It has some advantages. The data that will appear in the textdata (for example, the strings that have to be set as default choices in the textbox), can be written in the textbox but *before CreateTextArea is called*. So, the textbox that contains the text, appears immediately on the screen.

With this approach, the programmer *first* allocates a buffer of the right dimensions, and *after* he calls ndTBOX_CreateTextArea. The address in ram of the TextDataArray is known by the programmer, *because it is passed explictly to ndTBOX_CreateTextArea*.

There is another possibility. It is called *internal mode*. If you pass a TextDataArray parameter set to 0, Nanodesktop will automatically allocate the memory area that is needed. This allocation happens internally: the programmer doesn't know where this area is in ram, because ndTBOX_CreateTextArea doesn't communicate it (it returns only the tbox handle).

So, to use a TextArea of 40*30 chars, a call like this is sufficient:

```
    ndTBOX_CreateTextArea    (10, 10, 200, 40, "Test textbox",
                             NDKEY_SETTEXTAREA (40, 30), 0,
                              COLOR_BLACK,  COLOR_WHITE,
                              COLOR_BLACK,  COLOR_GRAY,
                              COLOR_BLACK,  COLOR_GRAY03,
                              NULL, NULL, NULL,
                               0,
                              WndHandle, NORENDER)
```

The internal mode is usually more convenient than the external mode. The address of the TextDataArray can be obtained *after* the creation of the textbox using an appropriate function provided by Nanodesktop. The only trouble is that internal mode doesn't allow you to preset the default strings that must be visualized in the TextBox, but there is a very simple workaround for this.

An example

Now, we can see a simple program that creates a textbox. The textbox has two rows: by default the text that is visable in the area is “Nanodesktop is my default” “choice for PSP programming”.

```
#include <nanodesktop.h>

int  WndHandle;
char YouCanExit;
int  TBoxHandle;

int ndMain()
{
    int Counter;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (10, 10, 350, 200, "Textbox test", COLOR_WHITE, COLOR_LBLUE,
                                 COLOR_GRAY, COLOR_WHITE, 0);

    if (WndHandle>=0)
    {
        ndLP_MaximizeWindow (WndHandle);

        TBoxHandle =   ndTBOX_CreateTextArea(10,10,310,40,"Test textbox",
                       NDKEY_SETTEXTAREA (40, 2), 0,
                       COLOR_BLACK,  COLOR_WHITE,
                       COLOR_BLACK,  COLOR_GRAY08,
                       COLOR_BLACK,  COLOR_GRAY03,
                       NULL, NULL, NULL,
                       0,
                       WndHandle, NORENDER);

        strcpy ( ndTBOX_GetRowAddr (0, TBoxHandle, WndHandle), "Nanodesktop is my default");
        strcpy ( ndTBOX_GetRowAddr (1, TBoxHandle, WndHandle), "choice for PSP programming");

        ndTBOX_TextAreaUpdate (TBoxHandle, WndHandle, RENDER);

        XWindowRender (WndHandle);

        YouCanExit=0;
        ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0);
    }
}
```

Note that we are using the internal mode in this example: the parameter TextData passed to ndTBOX_CreateTextArea (in green), is set 0, so that Nanodesktop allocates automatically the space in ram for a buffer of 40*2 chars.

The space that is allocated is initially void: the user has to fill it with the default values and after he has to ask for a graphical update of the TextArea.

Let's see how the user can change easily the content of the TextAreaArray.

In blue, we see a function, called ndTBOX_GetRowAddr. This function returns the address of a particular row of the TextAreaArray in memory. This is very important because this address can be passed to routines such as strcpy or memcpy that can modify the area.

So `ndTBOX_GetRowAddr (0, TBoxHandle, WndHandle)` returns the address of the first row of the array. There is also another function: `ndTBOX_GetCharAddr (10, 0, TBoxHandle, WndHandle);` for example, returns the address of the char at column 10, row 0 in the array.

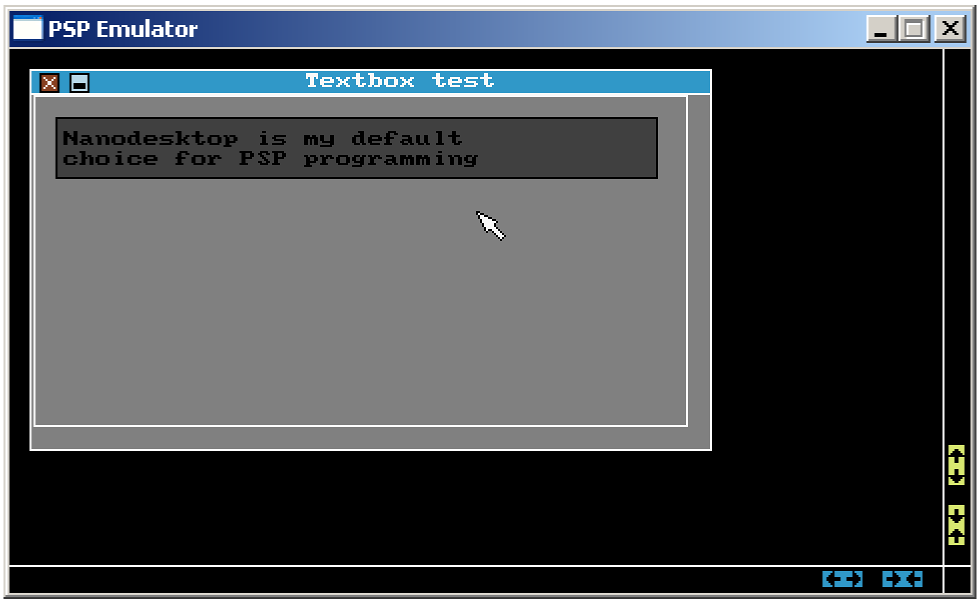
In any case, the two calls:

```
strcpy ( ndTBOX_GetRowAddr (0, TBoxHandle, WndHandle), "Nanodesktop is my default");
strcpy ( ndTBOX_GetRowAddr (1, TBoxHandle, WndHandle), "choice for PSP programming");
```

copy the wanted strings in the correct area. At this point it is only necessary to update (from a graphical point) the TextArea, using the function:

```
ndTBOX_TextAreaUpdate (TBoxHandle, WndHandle, RENDER);
```

And this is the result:



```
Nanodesktop is my default
choice for PSP programming
```

ndTBOX_GetNrCharsX, ndTBOX_GetNrCharsY

Functions such as *ndTBOX_GetRowAddr* or *ndTBOX_GetCharAddr* don't check if PosX or PosY parameters are included in the bounds of the textarea.

This creates a problem: the programmer must know how many rows and columns belong to the textarea. When the dimensions have been defined explicitly through the key NDKEY_SETTEXTAREA (x,y), this isn't a problem (the dimensions are known because the programmer defines them), but when the dimensions are calculated automatically by Nanodesktop (see page 146), it is necessary to have a function that returns this information.

Nanodesktop provides the following functions:

```
ndTBOX_GetNrCharsX (int TextBoxHandle, char WndHandle);
ndTBOX_GetNrCharsY (int TextBoxHandle, char WndHandle);
```

They return the number of columns and of rows that are associated with a TextBox.

ndTBOX_InhibitTextArea, ndTBOX_DeInhibitTextArea

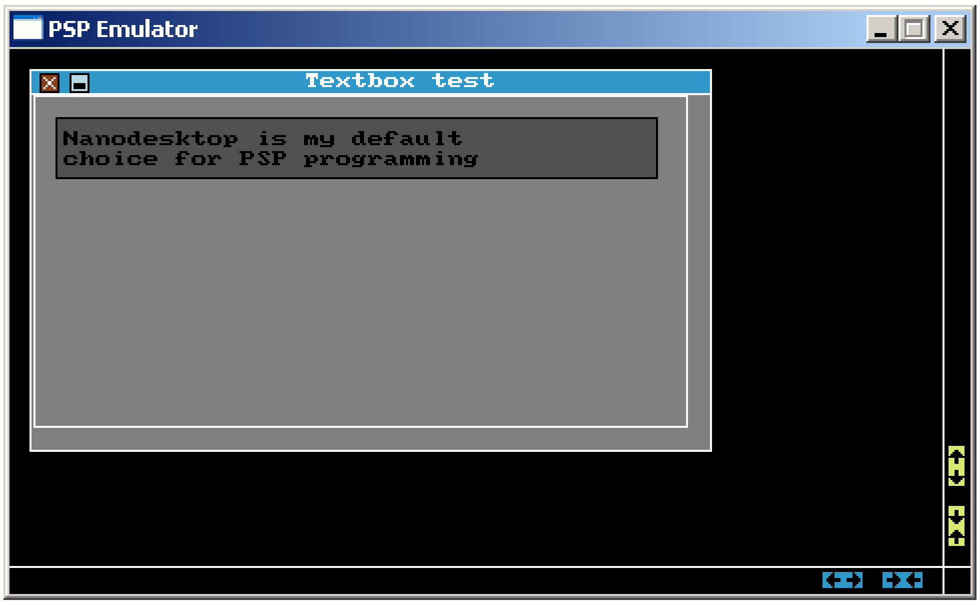
There are cases in which an application can want to inhibit a text area, for example because in that moment the option isn't active. There are two functions to inhibit or uninhibit a text area.

```
char ndTBOX_InhibitTextArea (int TextBoxHandle, char WndHandle, char RenderNow);
char ndTBOX_DeInhibitTextArea (int TextBoxHandle, char WndHandle, char RenderNow);
```

These functions can inhibit or uninhibit the element. When an element is inhibited, it doesn't react to the mouse pointer. View this source:

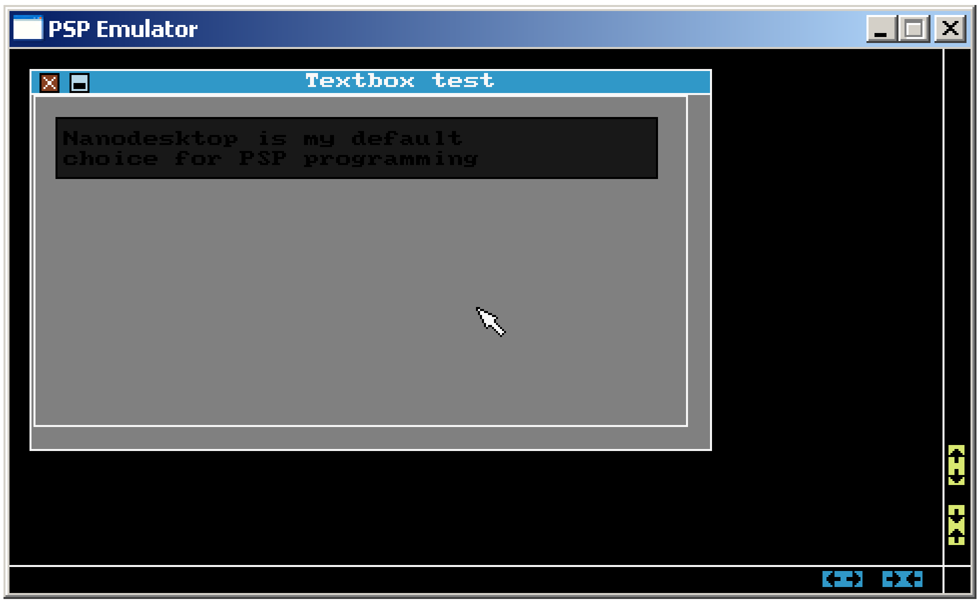
```
#include <nanodesktop.h>

int  WndHandle;
char YouCanExit;
int  TBoxHandle;

int ndMain()
{
    int Counter;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (10, 10, 350, 200, "Textbox test", COLOR_WHITE, COLOR_LBLUE,
                                 COLOR_GRAY, COLOR_WHITE, 0);

    if (WndHandle>=0)
    {
        ndLP_MaximizeWindow (WndHandle);

        TBoxHandle =   ndTBOX_CreateTextArea(10,10,310,40,"Test textbox",
                       NDKEY_SETTEXTAREA (40, 2), 0,
                       COLOR_BLACK,  COLOR_WHITE,
                       COLOR_BLACK,  COLOR_GRAY10,
                       COLOR_BLACK,  COLOR_GRAY03,
                       NULL, NULL, NULL,
                       0,
                       WndHandle, NORENDER);

        strcpy ( ndTBOX_GetRowAddr (0, TBoxHandle, WndHandle), "Nanodesktop is my default");
        strcpy ( ndTBOX_GetRowAddr (1, TBoxHandle, WndHandle), "choice for PSP programming");

        ndTBOX_TextAreaUpdate (TBoxHandle, WndHandle, NORENDER);

        XWindowRender (WndHandle);

        ndDelay (3);
        ndTBOX_InhibitTextArea (TBoxHandle, WndHandle, RENDER);

        YouCanExit=0;
        ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0);
    }
}
```

At first we'll see the following image:



Nanodesktop is my default
choice for PSP programming

but, after 3 seconds, the textbox becomes inhibited:



Nanodesktop is my default
choice for PSP programming

It is also possible to create an inhibited textbox in a different way.

It's sufficient to use the 64-bit key ND_TEXTAREA_INHIBITED in the Options parameter passed to ndTBOX_CreateTextBox, like this:

```
TBoxHandle =    ndTBOX_CreateTextArea(10,10,310,40,"Test textbox",
                      NDKEY_SETTEXTAREA (40, 2) | ND_TEXTAREA_INHIBITED, 0,
                      COLOR_BLACK,  COLOR_WHITE,
                      COLOR_BLACK,  COLOR_GRAY10,
                      COLOR_BLACK,  COLOR_GRAY03,
                      NULL, NULL, NULL,
                      0,
                      WndHandle, NORENDER);
```

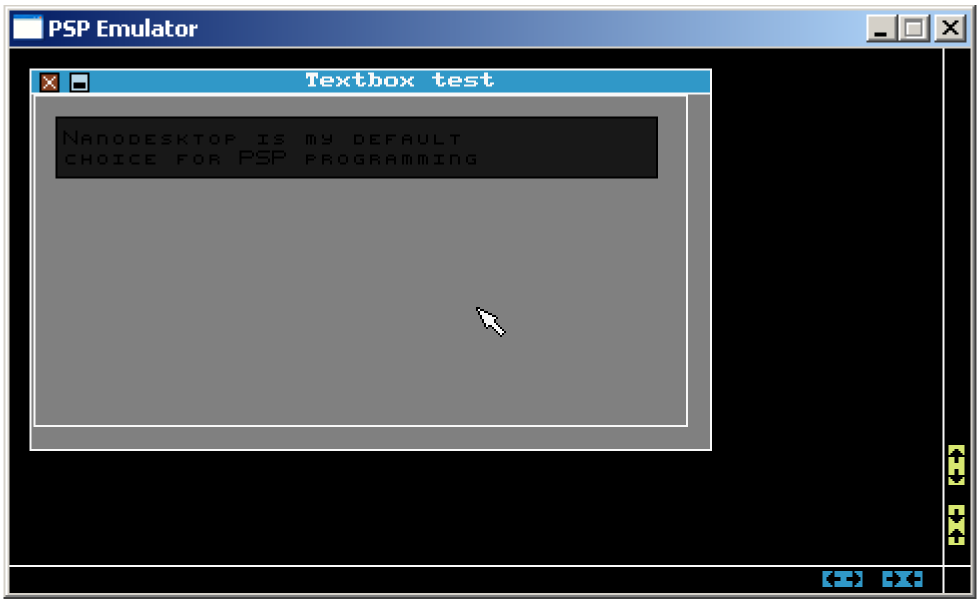
Fonts

Nanodesktop allows also the use of different fonts in the text area. For selecting a font, you can simply use another 64-bit key for the Options parameter: NDKEY_FONT (x).

See this source:

```
#include <nanodesktop.h>

int  WndHandle;
char YouCanExit;
int  TBoxHandle;

int ndMain()
{
    int Counter;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (10, 10, 350, 200, "Textbox test", COLOR_WHITE, COLOR_LBLUE,
                                 COLOR_GRAY, COLOR_WHITE, 0);

    if (WndHandle>=0)
    {
        ndLP_MaximizeWindow (WndHandle);

        TBoxHandle =   ndTBOX_CreateTextArea(10,10,310,40,"Test textbox",
                       NDKEY_SETTEXTAREA (40, 2) | ND_TEXTAREA_INHIBITED | NDKEY_FONT (3), 0,
                       COLOR_BLACK,  COLOR_WHITE,
                       COLOR_BLACK,  COLOR_GRAY10,
                       COLOR_BLACK,  COLOR_GRAY03,
                       NULL, NULL, NULL,
                       0,
                       WndHandle, NORENDER);

        strcpy ( ndTBOX_GetRowAddr (0, TBoxHandle, WndHandle), "Nanodesktop is my default");
        strcpy ( ndTBOX_GetRowAddr (1, TBoxHandle, WndHandle), "choice for PSP programming");

        ndTBOX_TextAreaUpdate (TBoxHandle, WndHandle, NORENDER);

        XWindowRender (WndHandle);

        YouCanExit=0;
        ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0);
    }
}
```

Result:



Callbacks

The examples that we have seen till now, are only theoretical. The text boxes that we have defined have associated no callbacks, so they are unable to manage the data that is entered into them.

To manage the data that is entered in a text box or in a text area, it is necessary to define one or more *textbox callbacks*.

A tbox callback is a routine with prototype:

```
void  cbProcessValueBefore  (void  *ProcessValueData,  void  *TextData,  int  TextBoxHandle,  char
TextBoxWndHandle);

char  cbProcessValueForValidation (void *ProcessValueData, void *TextData, int TextBoxHandle, char
TextBoxWndHandle);

void  cbProcessValueAfter  (void  *ProcessValueData,  void  *TextData,  int  TextBoxHandle,  char
TextBoxWndHandle);
```

In theory, each textbox has associated three callbacks, called *cbProcessValueBefore*, *cbProcessValueForValidation*, *cbProcessValueAfter.* All these callbacks have the same prototype seen before, despite the fact that they have different functions.

The address of this callback must be declared in the call to ndTBOX_CreateTextArea

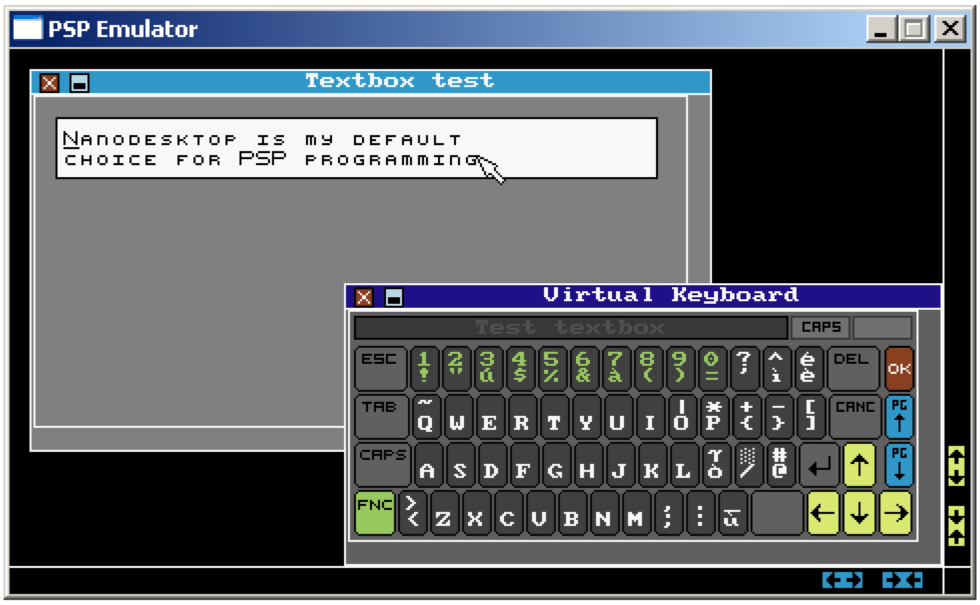
```
TBoxHandle =   ndTBOX_CreateTextArea(10,10,310,40,"Test textbox",
                       0, 0,
                       COLOR_BLACK,  COLOR_WHITE,
                       COLOR_BLACK,  COLOR_GRAY10,
                       COLOR_BLACK,  COLOR_GRAY03,
                       &cbProcessValueBefore, &cbProcessValueForValidation, &cbProcessValueAfter,
                       ProcessValueData,
                       WndHandle, NORENDER);
```

In reality, the programmer can “don't define” one or more tbox callbacks: in this case the relative parameter must be set to 0.

ndTBOX_CreateTextArea also uses a parameter called *void *ProcessValueData*. This is a generic pointer, and it is passed to all callbacks each time it is executed. The user can use this pointer to pass the address of a structure that belongs to their own program: so the tbox callback can obtain information from the user application.

Now, let's see the functions of the three callbacks.

When you active a textbox....



Nanodesktop is my default choice for PSP programming

... Nanodesktop opens the default keyboard (in this case, the Virtual Keyboard), and after it copies the content of the TextDataArray into a *reserve buffer.* So, if you write text and afterwards you change your mind, it is sufficient to press ESC or to exit from the Virtual Keyboard to recover the old content of the text box. The changes created by the keyboard, are made in the reserve buffer, preserving the actual content of the text box until the changes have been validated.

The callbacks are recalled only when you press the OK button (or equivalent if you use IR keyboard...).

Let's see what happens when you press OK.

The system executes (if it has been defined) the *cbProcessValueBefore* callback.

This is a callback that can be defined by the programmer for various purposes (usually this routine acts as a filter of what has been typed by the user). The parameter *ProcessValueData* passed to the callback is the same that has been defined when the programmer created the text box; the parameter TextData is a pointer to the reserve buffer; the pair (TextBoxHandle:TextBoxWndHandle) identify the textbox that is calling the callback.

So, using this information, the ProcessValueBefore callback can filter or transform the data typed by the user.

The second callback that is executed is *cbProcessValueForValidation*. This callback is important because it says to the system if the new string that has been typed by the user is acceptable for the program. This callback must be programmed by the author of the program: if the new content of the text box is acceptable, the callback will return the value 1, otherwise it will return the value 0 and the old content will be restored. If the user doesn't define a ProcessValueForValidation callback, then the data are considered *always* validated.

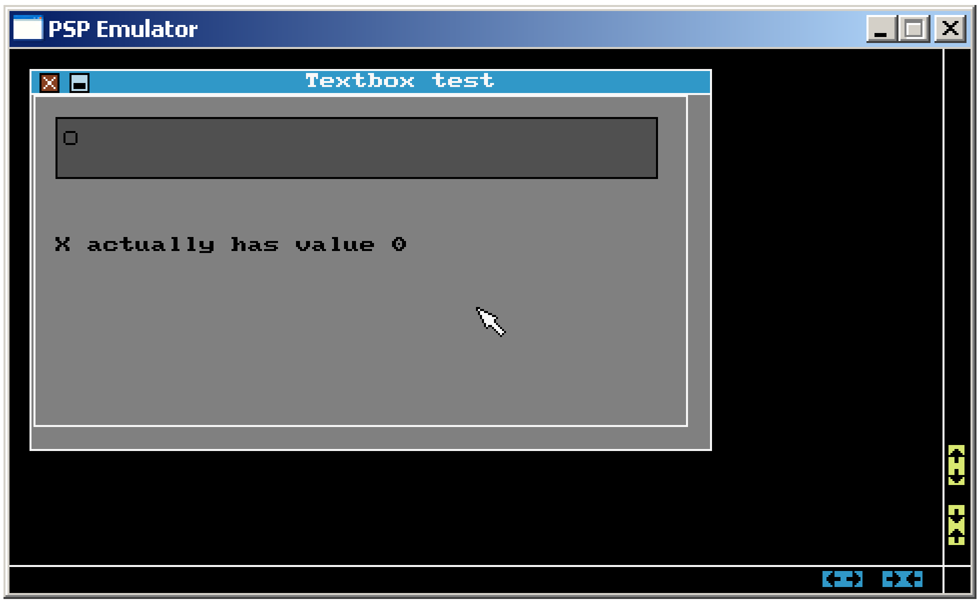
The third callback that can be executed is *cbProcessValueAfter.* In this case the callback acts directly on TextDataArray and not only on the reserve buffer, so the changes that it makes on the data are immediately shown in textbox area. The parameter TextData passed to the callback is the address of the TextData array, the other parameters have the same sense.

An example

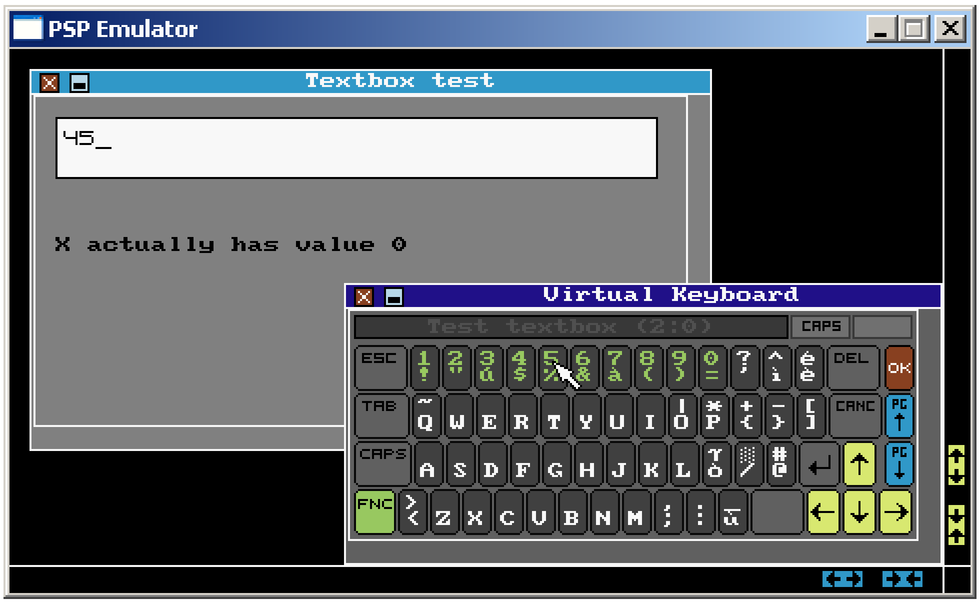
The program that follows creates a textbox on the screen and puts the string “0” in it. The user can enter a numerical value. When the user types a string, the callback for validation is called. This callback verifies if the string that has been entered contains only digits or if there are also letters. In this second case, the string is refused and the numerical value of the variable X remains unchanged.

```
#include <nanodesktop.h>

int  WndHandle;
char YouCanExit;
int  TBoxHandle;
int  X;

void DrawValue ()
{
     ndWS_DrawRectangle (10, 60, 310, 80, COLOR_GRAY, COLOR_GRAY, WndHandle, NORENDER);
      ndWS_GrPrintLn (10, 70, COLOR_BLACK, COLOR_GRAY, WndHandle, NORENDER, "X actually has value %d
", X);
     XWindowRender (WndHandle);
}

static  int  cbValidateText  (void  *ProcessValueData,  void  *TextData,  int  TextBoxHandle,  char
TextBoxWndHandle)
{
    int Counter;
    int IsValid;
    char MyChar;

    IsValid=1;

    for (Counter=0; Counter<40; Counter++)
    {
        MyChar = *(char *)(TextData+Counter);

        if (MyChar==0) break;

        if (! ((MyChar>='0') && (MyChar<='9')) )
        {
              IsValid=0;
              break;
        }
    }

    if (IsValid)
    {
        X = atoi ( TextData );
        DrawValue ();
    }
    else
    {
        ndWS_DrawRectangle (10, 90, 310, 110, COLOR_GRAY, COLOR_GRAY, WndHandle, NORENDER);
        ndWS_GrWriteLn (10, 100, "Error: invalid", COLOR_BLACK, COLOR_GRAY, WndHandle, RENDER);

        ndDelay (1);
        ndWS_DrawRectangle (10, 90, 310, 110, COLOR_GRAY, COLOR_GRAY, WndHandle, RENDER);
    }

    return IsValid;
}

int ndMain()
{
    int Counter;

    ndInitSystem();

    WndHandle=ndLP_CreateWindow (10, 10, 350, 200, "Textbox test", COLOR_WHITE, COLOR_LBLUE,
                                 COLOR_GRAY, COLOR_WHITE, 0);

    if (WndHandle>=0)
    {
        ndLP_MaximizeWindow (WndHandle);

        TBoxHandle =   ndTBOX_CreateTextArea(10,10,310,40,"Test textbox",
                       NDKEY_SETTEXTAREA (40, 1) | NDKEY_FONT (3), 0,
                       COLOR_BLACK,  COLOR_WHITE,
                       COLOR_BLACK,  COLOR_GRAY10,
                       COLOR_BLACK,  COLOR_GRAY03,
                       NULL, cbValidateText, NULL,
                       0,
                       WndHandle, NORENDER);

         strcpy ( ndTBOX_GetRowAddr (0, TBoxHandle, WndHandle), "0"); X = 0;
        ndTBOX_TextAreaUpdate (TBoxHandle, WndHandle, RENDER);

        DrawValue ();

        YouCanExit=0;
        ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0);
    }
}
```

This is the result:



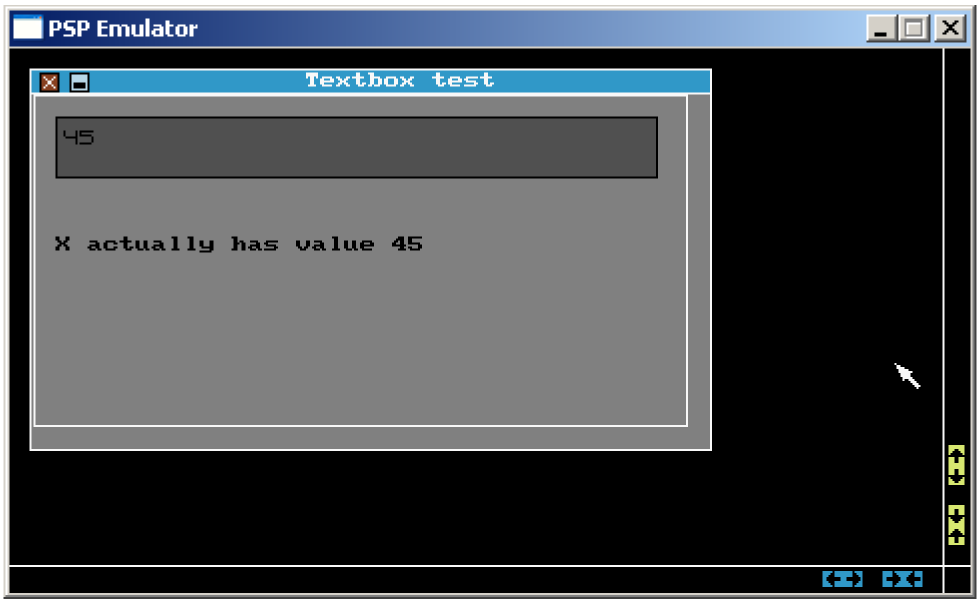
```
0
X actually has value 0
```



45_

X actually has value 0

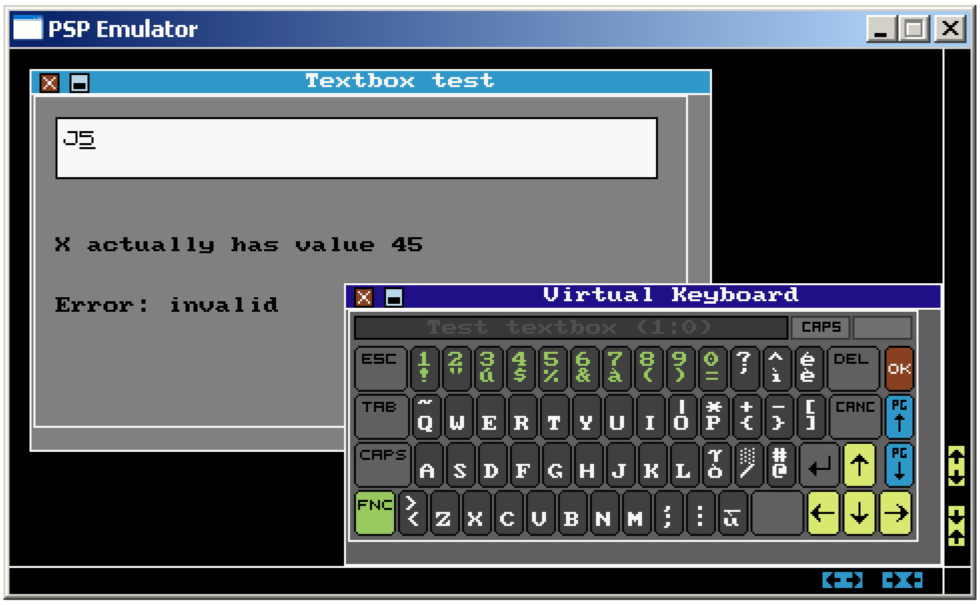
The new string has been validated



45

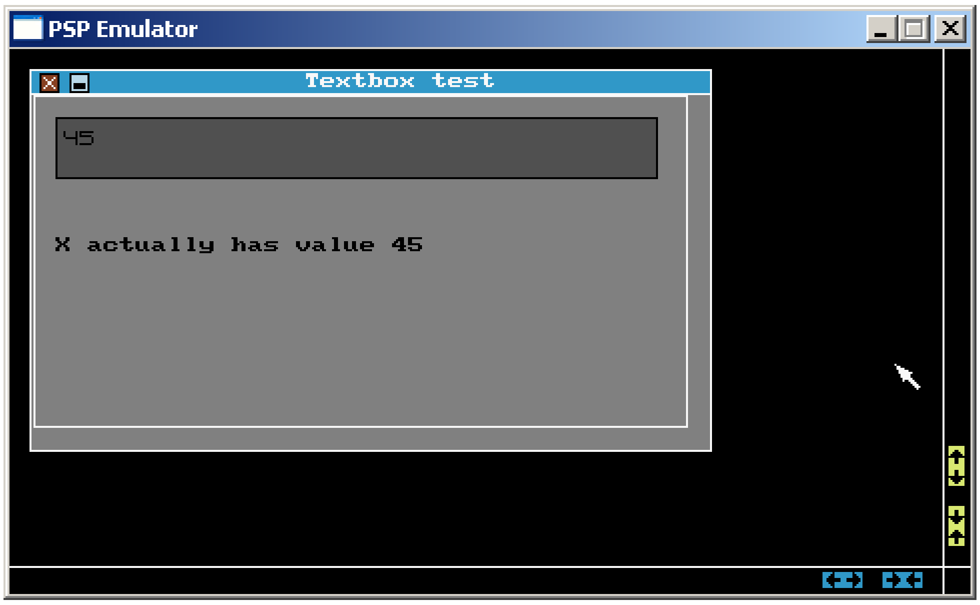
X actually has value 45

Now, try to enter a string that contains letters. The callback is executed, prints the error....



```
X actually has value 45

Error: invalid
```

.... and restores the old value....



```
X actually has value 45
```

Chapter 18
ListBoxes

Nanodesktop also supports list-boxes. Using these elements, a programmer can manage situations in which there are many different values for an option in his application.

ndLBOX_CreateListBox

The first thing to do is to *create the listbox.* Any list-box has a *listbox handle,* a number that identifies it in a unique way. You can use the following routine:

```
int  ndLBOX_CreateListBox (unsigned short int PosX1, unsigned short int PosY1,
                           unsigned short int PosX2, unsigned short int PosY2,
                           ndint64 Options, struct ndListBoxColors_Type *ListBoxColor,
                           char WndHandle, char RenderNow)
```

The parameters (PosX1, PosY1, PosX2, PosY2) define the coordinates of the rectangular area that will contain the list box.

Options is a 64-bit parameter that states important features of the list-box. The programmer can (as usual), define the features using particular 64-bit explicit constants that have to be declared or or-ed together. At the moment, we will set it to 0.

The parameter **ListBoxColor* is a pointer to a struct of type *ndListBoxColors_Type*. This struct contains the color parameters for the list-box. The programmer can create his own structures, and so he can use the color set that he wants, but he can also use two predefined system structures, that Nanodesktop provides as default, called `ndLBoxDefaultColor0` and `ndLBoxDefaultColor1`.

WndHandle indicates the window in which you want to create the list-box and RenderNow has the usual meaning.

An example call is this:

```
ListBoxHandle = ndLBOX_CreateListBox (10, 10, 300, 150, 0, 0, WndHandle, RENDER);
```

The ListBoxColor pointer is set to 0, so Nanodesktop assumes that the color set that has to be used is `ndLBoxDefaultColor0`

Another example of call is this:

```
ListBoxHandle = ndLBOX_CreateListBox (10, 10, 300, 150, 0, &ndLBoxDefaultColor1, WndHandle, RENDER);
```

The system creates a void listbox in memory (and in the window) and, if there are no errors, it returns the *lbox handle*.

Now, we must define the various items. This can be done this using *ndLBOX_AddItemToListBox.*

ndLBOX_AddItemToListBox

The prototype of ndLBOX_AddItemToListBox is this:

```
char ndLBOX_AddItemToListBox (char *NameLBoxItem, void *LBoxCallback, ndint64 LBoxCode, char
IsInhibited, int ListBoxHandle, char WndHandle)
```

This routine adds a new item to the list associated with the list box. NameLBoxItem is a string that identifies the item on the screen; LboxCallback is a pointer for the callback chosen by the programmer, LboxCode is a 64-bit code that will be passed to the callback.

The parameter IsInhibited is set to 1 only if the item is inhibited; otherwise it is set to 0.

The pair ListBoxHandle:WndHandle defines the listbox in which we are working.

Each item of the list box has an associated callback, called *listbox callback.* It is a routine with the following prototype:

```
void (*PntToLBoxCallback)(int NrItem, ndint64 LBoxCode, int ListBoxHandle, char WndHandle);
```

This routine, is executed each time that the user selects an item in the list. In theory, there is a callback for list box item, but the programmer can create items without associated callbacks: it is sufficient to specify the null pointer (0) as parameter LboxCallback in the call to LBOX_AddItemToListBox.

Let's analyze the data that the callback receives: the int NrItem indicates the ordering number of the item that has been selected. Using this number, a programmer can manage all items in a listbox with a unique lbox callback.

The parameter LboxCode is a 64 bit parameter that is passed by the programmer: this parameter had been associated to the item when the programmer had defined the element through ndLBOX_AddItemToListBox. Using this parameter, the programmer can pass some information to the callback.

The pair (ListBoxHandle:WndHandle) indicates which listbox has generated the execution of the callback.

When we add a new item, it isn't visualized immediately. It will be visualized only when the user will update the listbox using a routine called *ndLBOX_UpdateListBox*.

The simplest example

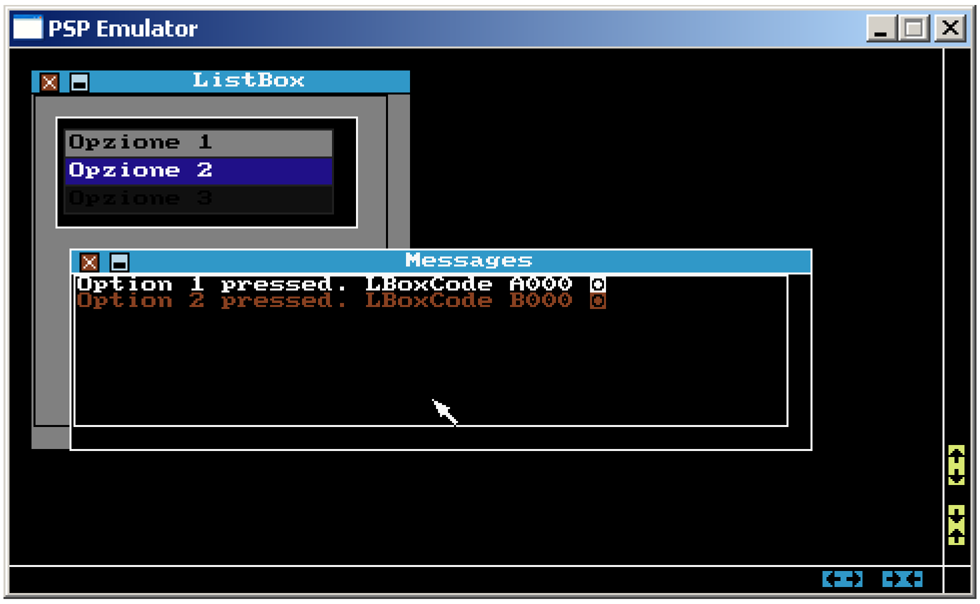
This is a very simple example: an application that uses a listbox with 3 items and 3 callbacks.

```
#include <nanodesktop.h>

int WndHandle;
int ListBoxHandle;

int WndDialogHandle;
int YouCanExit;


static void lbcbItem1 (int NrItem, ndint64 LBoxCode, int ListBoxHandle, char WndHandle)
{
     ndWS_PrintLn (WndDialogHandle, COLOR_WHITE, RENDER, "Option 1 pressed. LBoxCode %X \n",
(int)(LBoxCode));
}

static void lbcbItem2 (int NrItem, ndint64 LBoxCode, int ListBoxHandle, char WndHandle)
{
     ndWS_PrintLn (WndDialogHandle, COLOR_RED, RENDER, "Option 2 pressed. LBoxCode %X \n",
(int)(LBoxCode));
}

static void lbcbItem3 (int NrItem, ndint64 LBoxCode, int ListBoxHandle, char WndHandle)
{
     ndWS_PrintLn (WndDialogHandle, COLOR_LBLUE, RENDER, "Option 3 pressed. LBoxCode %X \n",
(int)(LBoxCode));
}


int ndMain()
{

   ndInitSystem ();

   WndHandle=ndLP_AllocateWindow (10, 10, 200, 200, "ListBox", COLOR_WHITE, COLOR_LBLUE, COLOR_GRAY,
COLOR_BLACK);
   ndLP_MaximizeWindow (WndHandle);

   WndDialogHandle=ndLP_AllocateWindow (30, 100, 400, 200, "Messages", COLOR_WHITE, COLOR_LBLUE,
COLOR_BLACK, COLOR_WHITE);
   ndLP_MaximizeWindow (WndDialogHandle);

   ListBoxHandle = ndLBOX_CreateListBox (10, 10, 160, 65, 0, 0, WndHandle, RENDER);

   ndLBOX_AddItemToListBox ("Opzione 1",  &lbcbItem1, 0xA000, 0, ListBoxHandle, WndHandle);
   ndLBOX_AddItemToListBox ("Opzione 2",  &lbcbItem2, 0xB000, 0, ListBoxHandle, WndHandle);
   ndLBOX_AddItemToListBox ("Opzione 3",  &lbcbItem3, 0xC000, 1, ListBoxHandle, WndHandle);
```

```
    ndLBOX_UpdateListBox (ListBoxHandle, WndHandle, RENDER);

    YouCanExit=0;
    ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0);
}
```

The system creates a listbox, defines 3 items (the third is inhibited), and asks the system to do a graphical update through *ndLBOX_UpdateListBox*.

There are three callbacks, called *lbcbItem1*, *lbcbItem2* and *lbcbItem3*. Each callback prints a message in a second window; moreover, it writes the LboxCode that the callback has received too (in hexadecimal form).



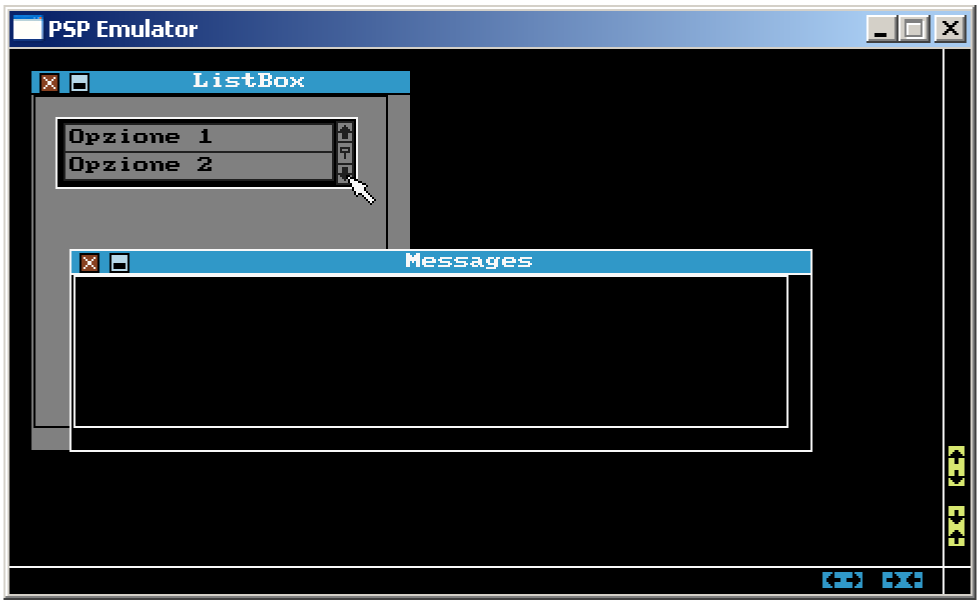
```
Option 1 pressed. LBoxCode A000
Option 2 pressed. LBoxCode B000
```

Autoscrolling

If the items are greater than the number visable in relation to the dimensions of the rectangle, Nanodesktop automatically manages the scroll of the list elements. For example, if we change the previous call in:

```
ListBoxHandle = ndLBOX_CreateListBox (10, 10, 160, 45, 0, 0, WndHandle, RENDER);
```

so that the height is reduced, the system reacts so:

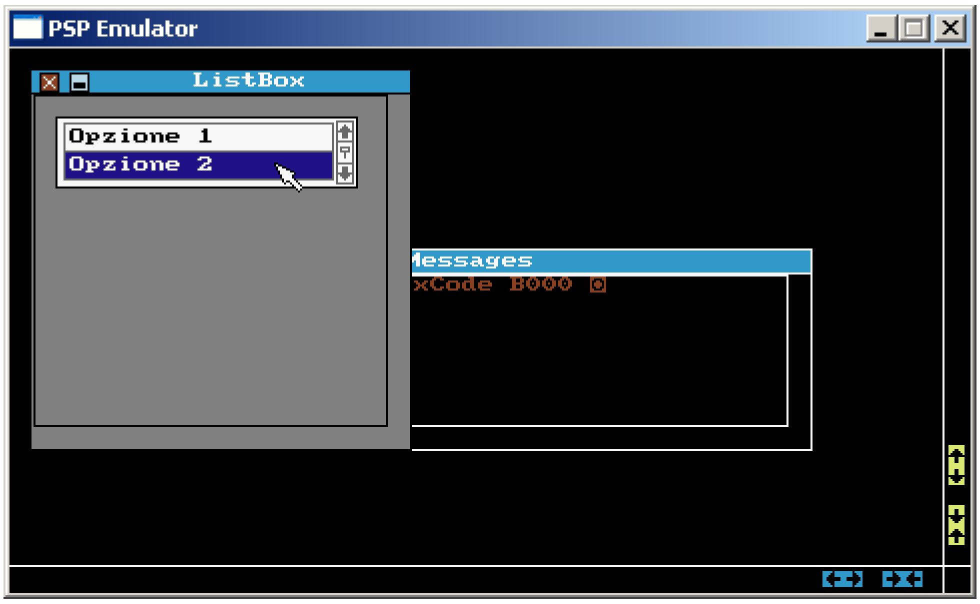


Change the colors

You can change the colors used by the list box in a very simple way. Let's change the call to ndLBOX_CreateListBox:

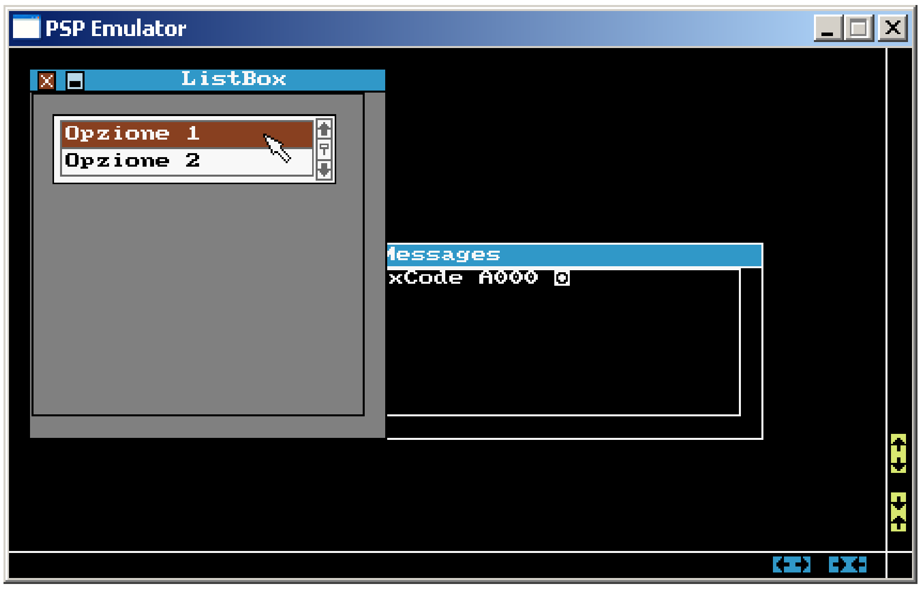
```
ListBoxHandle = ndLBOX_CreateListBox (10, 10, 160, 65, 0, &ndLBoxDefaultColor1, WndHandle, RENDER);
```

Here we are passing to the routine the address of a predefined system structure: Nanodesktop provides ndLBoxDefaultColor0 and ndLBoxDefaultColor1, that contain two default sets of colors.



Well, the user can also set his own colors: it is sufficient to create his own structure and to fill it with the desired colors. After, the programmer passes the address of the custom structure to ndLBOX_CreateListBox.

```
struct ndListBoxColors_Type MyLBoxColor =
{
    COLOR_BLACK,            // ColorBorder
    COLOR_WHITE,            // ColorBackground
    COLOR_GRAY13,           // ColorBorderSlot
    COLOR_WHITE,            // ColorSlot
    COLOR_RED,              // ColorSlotActive
    COLOR_WHITE,            // ColorSlotInh
    COLOR_BLACK,            // ColorTextSlot
    COLOR_WHITE,            // ColorTextSlotActive
    COLOR_GRAY28            // ColorTextSlotInh
};

ListBoxHandle = ndLBOX_CreateListBox (10, 10, 160, 45, 0, &MyLBoxColor, WndHandle, RENDER);
```



Strategy of highlighting

When the user selects a single item of the list-box, Nanodesktop has to decide if the item has to highlighted and for how much time....

This problem is called *strategy of highlighting.* Nanodesktop provides four predefined strategies: the programmer has to choose one of them when he creates the list box through *ndLBOX_CreateListBox*.

The correct strategy is chosen using an explicit constant that is passed using the 64-bit parameter Options in ndLBOX_CreateListBox. These constants are:

0 ND_LBOX_STABLE_EVIDENCE (default)

When an item is selected it remains highlighted permanently. It can be applied to only one highlighted element at a time.

1 ND_LBOX_FLASH_EVIDENCE

When an item is selected, it is highlighted by the system for some milliseconds; after it returns normal.

2 ND_LBOX_MULTI_EVIDENCE

When the user selects an item, it is highlighted. When the user selects a second item that is highlighted too. When the user selects an item that has already been highlighted, the item returns to normal. This mode allows the presence of different simultaneous items.

3 ND_LBOX_NULL_EVIDENCE

Do nothing. Nanodesktop leaves to the code contained in the callbacks and written by the programmer the task to show or unshow the items, using the appropriate functions.

For example, a call of this kind:

```
ListBoxHandle= ndLBOX_CreateListBox (10, 10, 160, 65, ND_LBOX_FLASH_EVIDENCE, 0, WndHandle, RENDER);
```

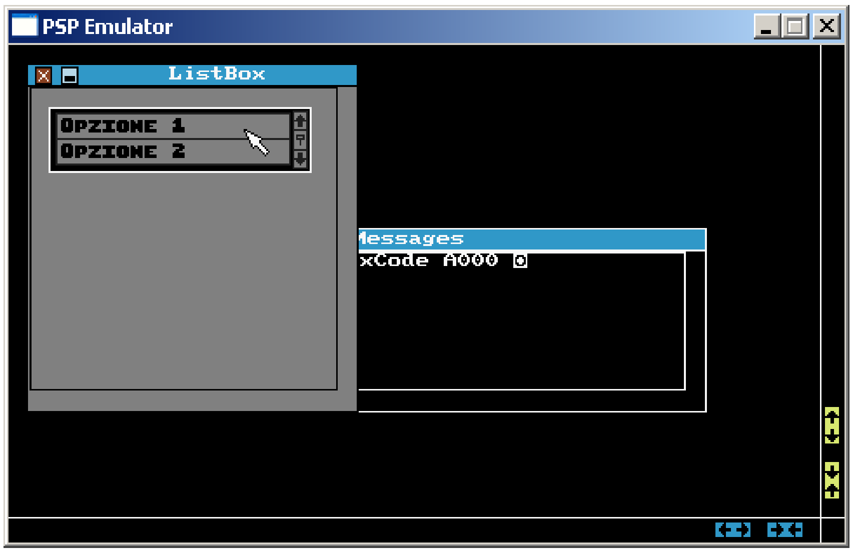
creates a listbox that uses ND_LBOX_FLASH_EVIDENCE as its strategy

Fonts

You can also set the font that is used in the list box. This is possible using a 64-bit key in the Options parameter. For example, if in the previous example the programmer writes a call like this:

```
ListBoxHandle = ndLBOX_CreateListBox (10, 10, 160, 45, NDKEY_FONT (4) | ND_LBOX_FLASH_EVIDENCE, 0,
WndHandle, RENDER);
```

the result is this:



Extra context callback

Lbox callbacks are callbacks similar to which that are associated with the buttons. So also these callbacks have the same troubles of MCinMC seen at page 110. For this reason, nd allows to define the lbox callbacks as extra-context. For example:

```
ListBoxHandle = ndLBOX_CreateListBox (10, 10, 160, 45, ND_CALLBACK_IN_NEW_CONTEXT | ND_LBOX_FLASH_EVIDENCE, 0, WndHandle, RENDER);
```

Under Nanodesktop, each element in the list box is identified by a number, called *NrItem.*

ndLBOX_FindListBoxItemByName

You can obtain the NrItem value through its name, using this function:

```
int ndLBOX_FindListBoxItemByName (char *Name, int ListBoxHandle, char WndHandle)
```

The function returns the numerical value (>0). If the result is negative, it means that there is an error or that the item hasn't been found (error ERR_STRINGID_NOT_FOUND).

ndLBOX_InhibitListBoxItem, ndLBOX_InhibitListBoxItemByName ndLBOX_DeInhibitListBoxItem, ndLBOX_DeInhibitListBoxItemByName

These four routines allow to inhibit or de-inhibit an item in the list-box. The prototypes are the following:

```
char ndLBOX_InhibitListBoxItem (int NrElement, int ListBoxHandle, char WndHandle, char RenderNow)
char ndLBOX_InhibitListBoxItemByName (char *Name, int ListBoxHandle, char WndHandle, char RenderNow)

char ndLBOX_DeInhibitListBoxItem (int NrElement, int ListBoxHandle, char WndHandle, char RenderNow)
char ndLBOX_DeInhibitListBoxItemByName (char *Name, int ListBoxHand, char WndHandle, char RenderNow)
```

The item can be identified by its name or by the value NrItem. If there are no errors, these routines return 0.

ndLBOX_EvidenceListBoxItem, ndLBOX_EvidenceListBoxItemByName ndLBOX_DeEvidenceListBoxItem, ndLBOX_DeEvidenceListBoxItemByName

These four routines allow to evidence or de-evidence an item in the list-box. The prototypes are the following:

```
char ndLBOX_EvidenceListBoxItem (int NrElement, int ListBoxHandle, char WndHandle, char RenderNow)
char ndLBOX_EvidenceListBoxItemByName (char *Name, int ListBoxHandle, char WndHandle, char RenderNow)

char ndLBOX_DeEvidenceListBoxItem (int NrElement, int ListBoxHandle, char WndHandle, char RenderNow)
char ndLBOX_DeEvidenceListBoxItemByName (char *Name, int ListBoxHandle, char WndHandle, char RenderNow)
```

ndLBOX_RemoveListBoxItem, ndLBOX_RemoveListBoxItemByName

These two routines allows to remove an item from the listbox. The changes become visible only when RenderNow is set to 1

```
char ndLBOX_RemoveListBoxItem (int NrElement, int ListBoxHandle, char WndHandle, char RenderNow)
char ndLBOX_RemoveListBoxItemByName (char *Name, int ListBoxHandle, char WndHandle, char RenderNow)
```

There are some other functions (like *ndLBOX_DestroyListBox*) that we don't documented here for space reasons. You can find other informations about them in Nanodesktop source:

<ndenv>\PSP\SDK\Nanodesktop\src\ndCODE_ListBox.c

Chapter 19

Progress bars

Nanodesktop can draw progress bars in a window. The support for progress bars is provided by a single function: ndTBAR_DrawProgressBar

```
char ndTBAR_DrawProgressBar (unsigned short int RRPosX1, unsigned short int RRPosY1,
                             unsigned short int RRPosX2, unsigned short int RRPosY2,
                             float Value, float MinValue, float MaxValue,
                             TypeColor StringColor, TypeColor BarColor,
                             TypeColor BGBarColor, TypeColor BorderColor,
                             ndint64 Attribute, char WndHandle,
                             char RenderNow)
```

The progress bar is created in a rectangular area: (RRPosX1, RRPosY1, RRPosX2, RRPosY2) are the coordinates of the rectangle in the window space.

(Value, MinValue, MaxValue) are three floats: the former is the value that controls the progress bar, MinValue and MaxValue are the minimum and the maximum number that can be assumed by the Value variable. Nanodesktop uses these two values to understand what percentage of the progress bar has to be highlighted and what percentage hasn't.

StringColor is the color of the string that will be drawn over the progress-bar; BarColor and BGBarColor represent the colors of the bar and of the background of the bar; BorderColor is the color of the border of the rectangle that contains the progress-bar.

WndHandle and RenderNow have the usual sense.

So, as we see, the only difficult aspect is the 64-bit parameter Attribute. This value can contain various information: the programmer sets the options using explicit constants that are declared or or-ed together.

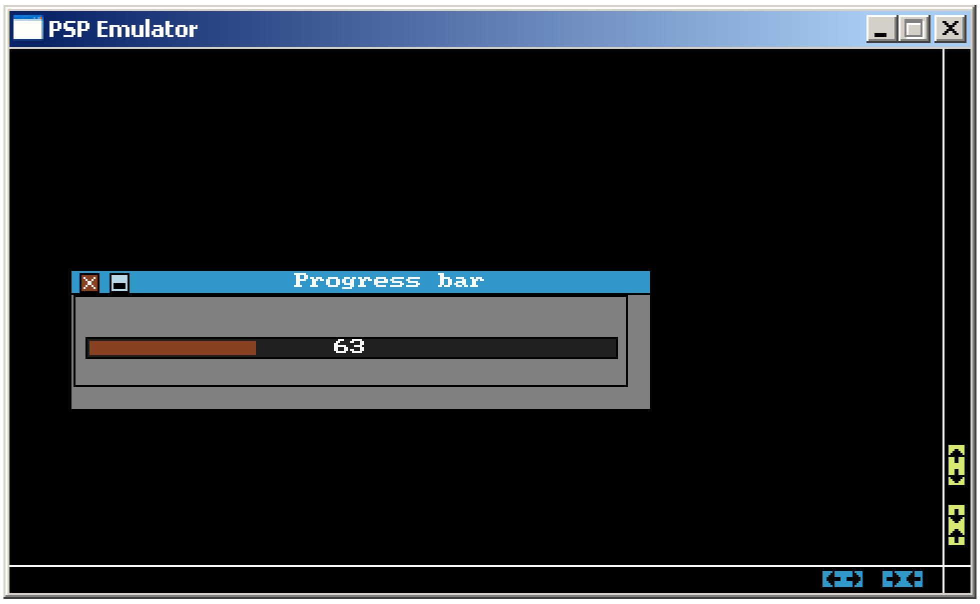
The 4 less significant bits (0-3) of Attribute contain the number of decimal digits that will be shown on the progress-bar, when it is drawn.

The 5 bit indicates the use of percent mode: the user can simply use the constant PBAR_PERCENT.

For example, see this source:

```
#include <nanodesktop.h>

int WndHandle;
int Counter;

int ndMain()
{
   ndInitSystem ();

   WndHandle=ndLP_AllocateWindow (30, 110, 320, 180, "Progress bar", COLOR_WHITE, COLOR_LBLUE,
COLOR_GRAY, COLOR_BLACK);
   ndLP_MaximizeWindow (WndHandle);

   for (Counter=0; Counter<200; Counter++)
   {
       ndTBAR_DrawProgressBar (5, 20, 270, 30, Counter, 0, 200,
                               COLOR_WHITE, COLOR_RED, COLOR_GRAY04, COLOR_BLACK,
                               0, WndHandle, RENDER);
   }

}
```

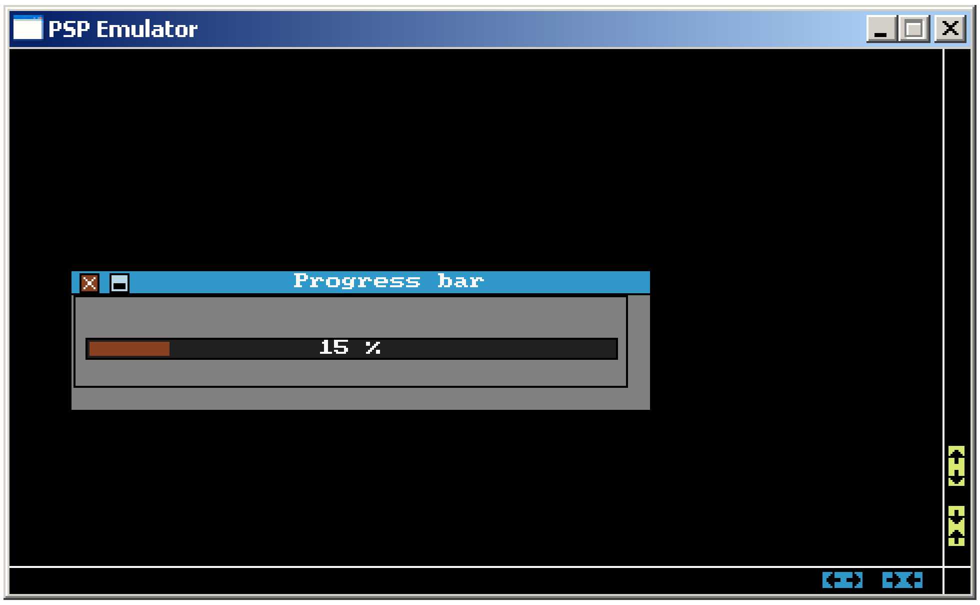
That shows this result:



If we want to have the percent visible, you can modify the source in the following way:

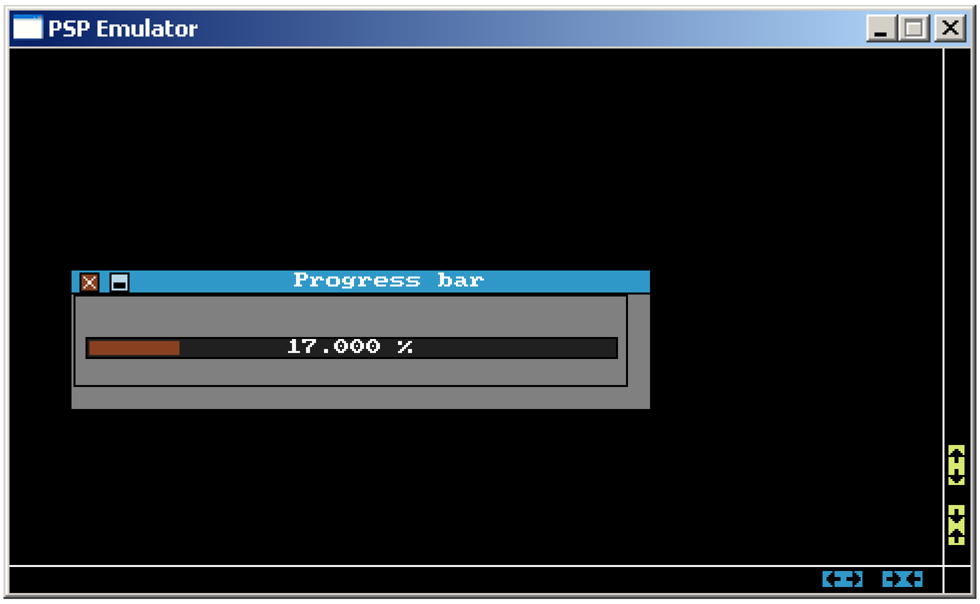
```
#include <nanodesktop.h>

int WndHandle;
int Counter;

int ndMain()
{
   ndInitSystem ();

   WndHandle=ndLP_AllocateWindow (30, 110, 320, 180, "Progress bar", COLOR_WHITE, COLOR_LBLUE,
COLOR_GRAY, COLOR_BLACK);
   ndLP_MaximizeWindow (WndHandle);

   for (Counter=0; Counter<200; Counter++)
   {
       ndTBAR_DrawProgressBar (5, 20, 270, 30, Counter, 0, 200,
                                 COLOR_WHITE, COLOR_RED, COLOR_GRAY04, COLOR_BLACK,
                                 PBAR_PERCENT, WndHandle, RENDER);
   }

}
```

This is what you'll obtain:



This third example shows 3 significant digits in the progress bar:

```
#include <nanodesktop.h>

int WndHandle;
int Counter;

int ndMain()
{
   ndInitSystem ();

   WndHandle=ndLP_AllocateWindow (30, 110, 320, 180, "Progress bar", COLOR_WHITE, COLOR_LBLUE,
COLOR_GRAY, COLOR_BLACK);
   ndLP_MaximizeWindow (WndHandle);

   for (Counter=0; Counter<200; Counter++)
   {
       ndTBAR_DrawProgressBar (5, 20, 270, 30, Counter, 0, 200,
                                COLOR_WHITE, COLOR_RED, COLOR_GRAY04, COLOR_BLACK,
                                PBAR_PERCENT | 3, WndHandle, RENDER);
   }

}
```

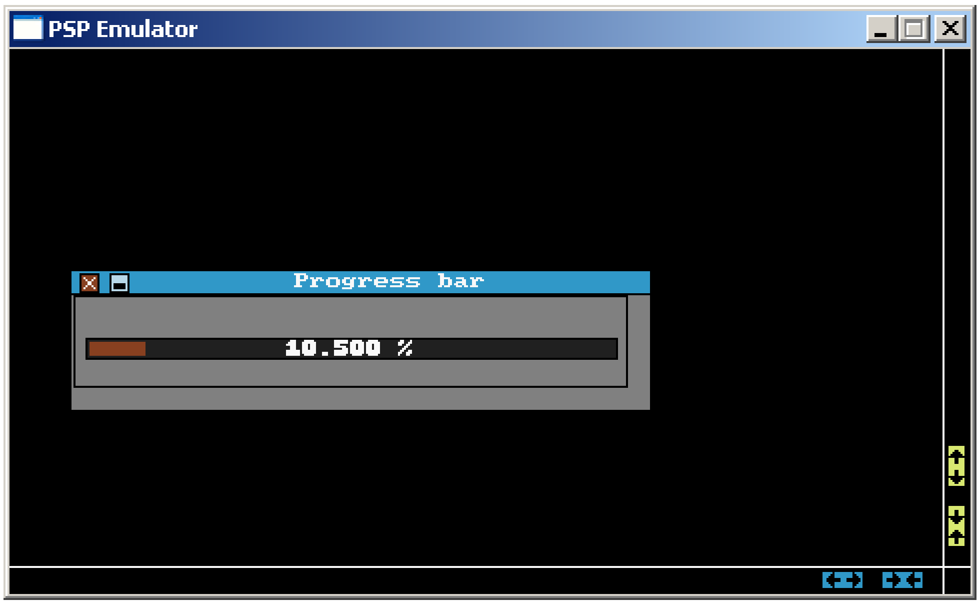


Fonts

Now, we'll see how to set the font used by the progress-bar. It's very simple. You can use the key KEY_FONT in the Attribute parameter.

```
#include <nanodesktop.h>

int WndHandle;
int Counter;

int ndMain()
{
   ndInitSystem ();

   WndHandle=ndLP_AllocateWindow (30, 110, 320, 180, "Progress bar", COLOR_WHITE, COLOR_LBLUE,
COLOR_GRAY, COLOR_BLACK);
   ndLP_MaximizeWindow (WndHandle);

   for (Counter=0; Counter<200; Counter++)
   {
       ndTBAR_DrawProgressBar (5, 20, 270, 30, Counter, 0, 200,
                                COLOR_WHITE, COLOR_RED, COLOR_GRAY04, COLOR_BLACK,
                                NDKEY_FONT (4) | PBAR_PERCENT | 3, WndHandle, RENDER);
   }

}
```

This is what you'll obtain:



Chapter 20

Check-boxes

The Nanodesktop graphical system can also manage check-boxes. Using check-boxes, the user of an application can enable or disable an option.

The Cbox can be created using the function *ndCBOX_CreateCBox:*

```
int ndCBOX_CreateCBox (unsigned short int PosX, unsigned short int PosY,
                       char *StringID, char IsOn, ndint64 Features,
                       int *AdrVariable,
                       TypeColor MainColor,
                       void *CBoxCallback, char WndHandle, char RenderNow)
```

It creates a check box in the window space and returns the *cbox handle* (if the returned value is negative, it means that there is an error). The C-Box's are rectangles of fixed dimensions: the user must declare the position of the upper corner through the parameters (PosX, PosY).

StringID is a string that identifies the check-box. IsOn specifies if the c-box, at the moment of its creation, has to enabled or disabled.

Features is a 64-bit parameter: as usual, it is bitmapped, and it allows us to pass options to the element.

Each C-Box is associated with an integer control variable: *AdrVariable is a pointer to the associated variable. When the user changes the status of the check-box, nd updates the value of the control variable (this value can be 0 or 1). The application can access the status of a check-box, simply accessing the variable associated to it.

CboxCallback is a pointer to a callback that is associated with c-box. The user can disable the c-box callback, simply passing 0 as parameter.

WndHandle and RenderNow have the usual meaning.

CboxCallback

A c-box callback is a routine with the prototype:

```
void CBoxCallback (int ActualValue, char *StringID, int CBoxHandle, char WndHandle)
```

The int *ActualValue* is the value of the control variable defined when you have created the check box.

When you modify the status of a c-box, the system updates the control variable, and *after,* it calls the assigned cbox callback to elaborate the change of status. For example, some applications may want to be notified of the fact the an option has been changed by the user. This can be easily done, defining a cbox callback that manages the event.

StringID is the string that is associated with the cbox. The pair (CboxHandle:WndHandle) indicates the cbox that has generated the event managed by the callback. It is evident that it is possible to manage different events using the same callback.

A very simple example

See this source:

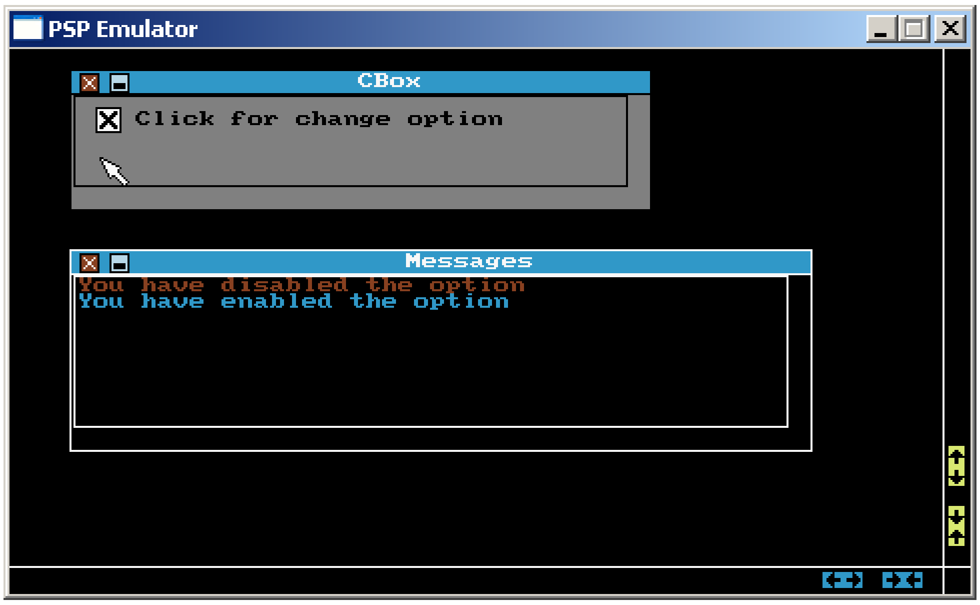
```
#include <nanodesktop.h>

int WndHandle;
int WndDialogHandle;
int YouCanExit;

int cbox0_CtrVariable;
int cbox0_Handle;
```

```
static void cbox0_Callback (int ActualValue, char *StringID, int CBoxHandle, char WndHandle)
{
      switch (ActualValue)
      {
          case 0:
          {
                ndWS_WriteLn ("You have disabled the option", COLOR_RED, WndDialogHandle, RENDER);
                break;
          }

          case 1:
          {
                ndWS_WriteLn ("You have enabled the option", COLOR_LBLUE, WndDialogHandle, RENDER);
                break;
          }
      }
}

int ndMain()
{
   ndInitSystem ();

    WndHandle=ndLP_AllocateWindow (30, 10, 320, 80, "CBox", COLOR_WHITE, COLOR_LBLUE, COLOR_GRAY,
COLOR_BLACK);
   ndLP_MaximizeWindow (WndHandle);

    WndDialogHandle=ndLP_AllocateWindow (30, 100, 400, 200, "Messages", COLOR_WHITE, COLOR_LBLUE,
COLOR_BLACK, COLOR_WHITE);
   ndLP_MaximizeWindow (WndDialogHandle);

   cbox0_CtrVariable=1;
   cbox0_Handle = ndCBOX_CreateCBox (10, 5, "cbox0", cbox0_CtrVariable, 0, &cbox0_CtrVariable,
                                          COLOR_WHITE, &cbox0_Callback, WndHandle, NORENDER);

   ndWS_GrWriteLn (30, 5+2, "Click for change option", COLOR_BLACK, COLOR_GRAY, WndHandle, RENDER);

   YouCanExit=0;
   ndProvideMeTheMouse_Until (&YouCanExit, ND_IS_EQUAL, 1, 0, 0);
}
```

This application creates a single check box in a window. Note a particular trick that it is much used with the check-boxes (the parts in red): the programmer sets the control variable (in this case *cbox0_CtrVariable*) to its initial value, and after it calls CreateCBox using the value cbox0_CtrVariable as the parameter *IsOn*.

So the cbox will appear *enabled* only if the respective control variable is *on*. This trick appears in the code as two successive references to the same control variable in the same call: the former is a reference *to the value* of the variable, and the second is a reference *to the address* of the variable.

The callback is called cbox0_Callback: it writes a message on the screen.
The result is this:



```
You have disabled the option
You have enabled the option
```

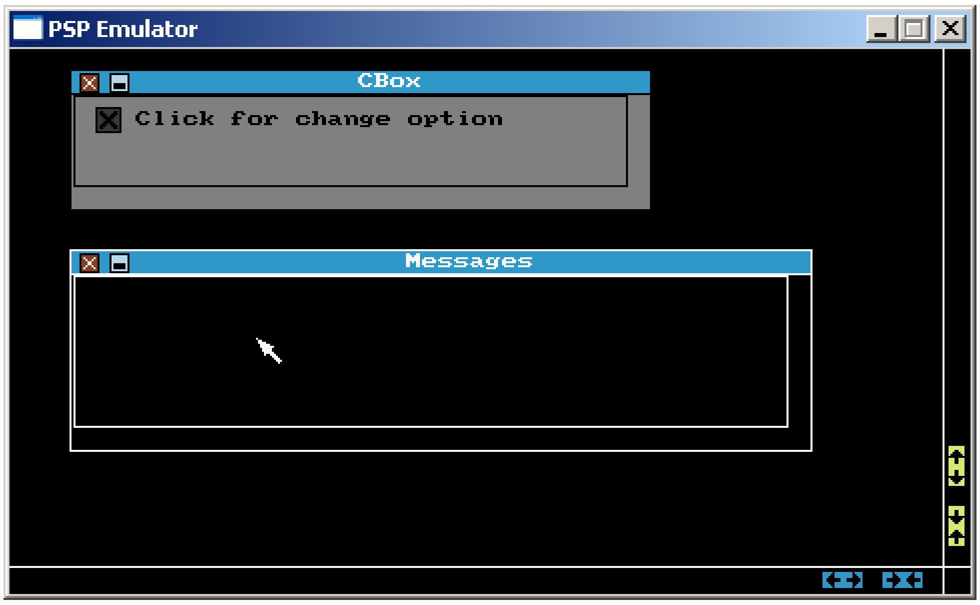
Inhibited check boxes

You can create check boxes that are inhibited from the first instant. To do that, you have to pass an explicit constant as *Features* parameter: *ND_CBOX_INHIBITED*.

Let's modify the previous source in the following way: the function that creates the c-box becomes:

```
cbox0_Handle = ndCBOX_CreateCBox (10, 5, "cbox0", cbox0_CtrVariable, ND_CBOX_INHIBITED,
                                  &cbox0_CtrVariable, COLOR_WHITE, &cbox0_Callback,
                                  WndHandle, NORENDER);
```

This is the result: the check box is inhibited:



An inhibited check box can be de-inhibited at any moment simply using a routine such as ndCBOX_DeInhibitCBox.

Extra context callback

The cbox callbacks executed by the Phoenix Mouse Thread have the same problems (*MCinMC trouble,* see page 111) as the buttons callbacks. So, if your cbox callback calls (itself or one of its subroutines) a routine that uses internally MouseControl (see page 116), you must define this cbox callback as *extra-context* to avoid crashing the system.

In the case of the check-boxes, this can be done using an explicit constant that is passed (or or-ed) in the Attribute parameter. For example, this call:

```
cbox0_Handle = ndCBOX_CreateCBox (10, 5, "cbox0", cbox0_CtrVariable,
                                  ND_CBOX_INHIBITED | ND_EXEC_IN_NEW_CONTEXT,
                                  &cbox0_CtrVariable, COLOR_WHITE, &cbox0_Callback,
                                  WndHandle, NORENDER);
```

creates an inhibited cbox with callback executed extra-context.

The dimensions of the check boxes

All check boxes are rectangles of 12*12 pixels. There is actually no way to change these values.

ndCBOX_InhibitCBox, ndCBOX_DeInhibitCBox

In any moment the user can manually inhibit or deinhibit a check box, using these routines:

```
char ndCBOX_InhibitCBox (int CBoxHandle, char WndHandle, char RenderNow)
char ndCBOX_DeInhibitCBox (int CBoxHandle, char WndHandle, char RenderNow)
```

The pair (CboxHandle:WndHandle) identifies the check box.

ndCBOX_SetCboxOn, ndCBOX_SetCboxOff

In any moment you can change the state (on or off) of a check-box, using these routines:

```
char ndCBOX_SetCboxOn (int CBoxHandle, char WndHandle, char ExecCallBack, char RenderNow)
char ndCBOX_SetCboxOff (int CBoxHandle, char WndHandle, char ExecCallBack, char RenderNow)
```

The pair (CboxHandle:WndHandle) identifies the check box. If the programmer sets to 1 the parameter ExecCallback, Nanodesktop will change the value of the control variable, and after it will recall automatically the cbox callback to process the event.

ndCBOX_ChangeCallback

This routine allows you to change the callback associated with a check box:

```
char ndCBOX_ChangeCallback (int CBoxHandle, char WndHandle, void* NewCallBack)
```

ndCBOX_DestroyCBox

It destroys the check box.

```
char ndCBOX_DestroyCBox (int CBoxHandle, char WndHandle, char OnlyLogical, TypeColor Color, char
RenderNow)
```

Chapter 21

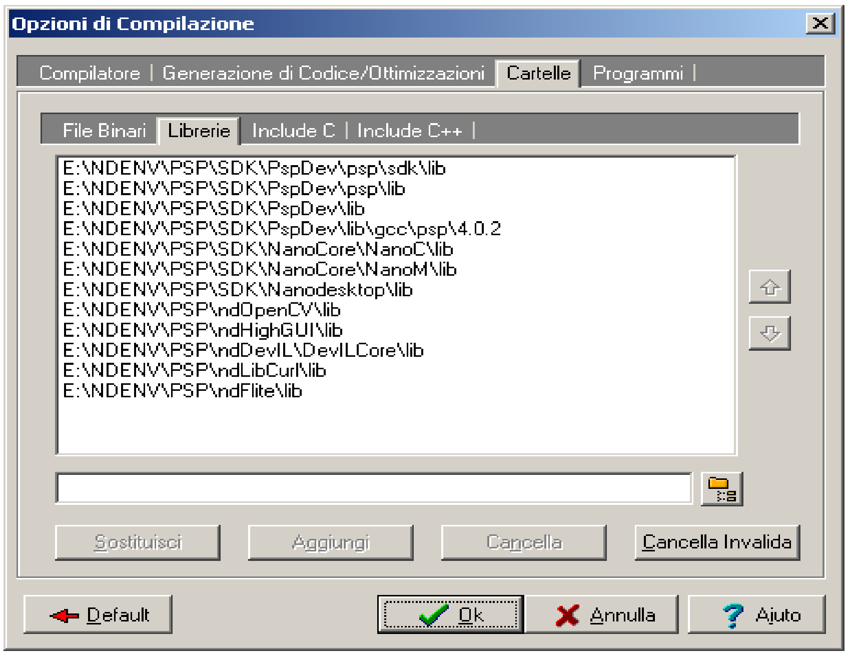
True Type fonts

At beginning from version 0.3.4 Nanodesktop offers support for the *TrueType fonts.* Nd applications can visualize the fonts that are normally used in Microsoft Windows or in Xwindow Graphical System.

Enable TrueType support

The support for TTF font is usually disabled by default. In this way, the compiler hasn't to link the ndTrueType library and so the system can save memory. If you want to use TTF support, you must enable it.

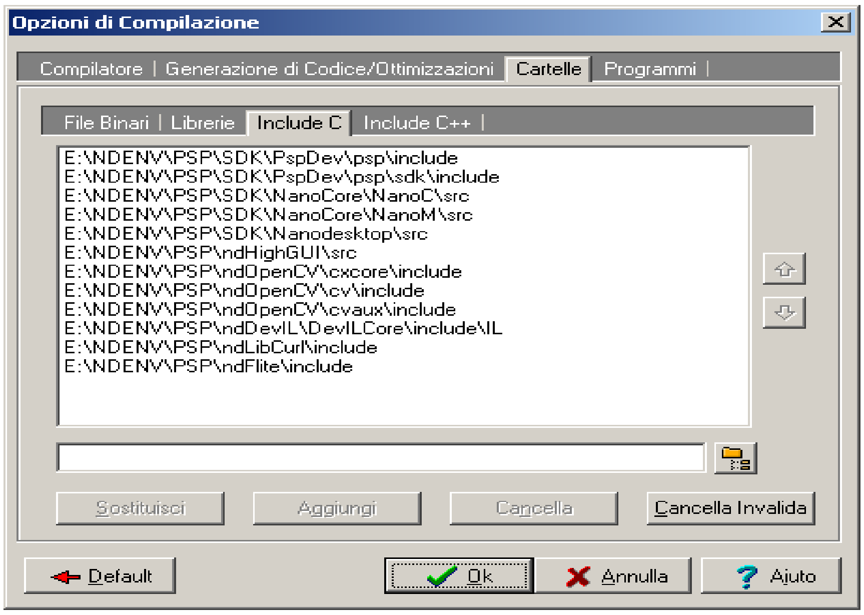
First step: add the following directory to the list of directories for linker:



```
E:\NDENV\PSP\SDK\PspDev\psp\sdk\lib
E:\NDENV\PSP\SDK\PspDev\psp\lib
E:\NDENV\PSP\SDK\PspDev\lib
E:\NDENV\PSP\SDK\PspDev\lib\gcc\psp\4.0.2
E:\NDENV\PSP\SDK\NanoCore\NanoC\lib
E:\NDENV\PSP\SDK\NanoCore\NanoM\lib
E:\NDENV\PSP\SDK\Nanodesktop\lib
E:\NDENV\PSP\ndOpenCV\lib
E:\NDENV\PSP\ndHighGUI\lib
E:\NDENV\PSP\ndDevIL\DevILCore\lib
E:\NDENV\PSP\ndLibCurl\lib
E:\NDENV\PSP\ndFlite\lib
```

Add **<ndenv path>\PSP\ndFreeType\lib**

Second step: Add to the list of directories for *include* the following directory:

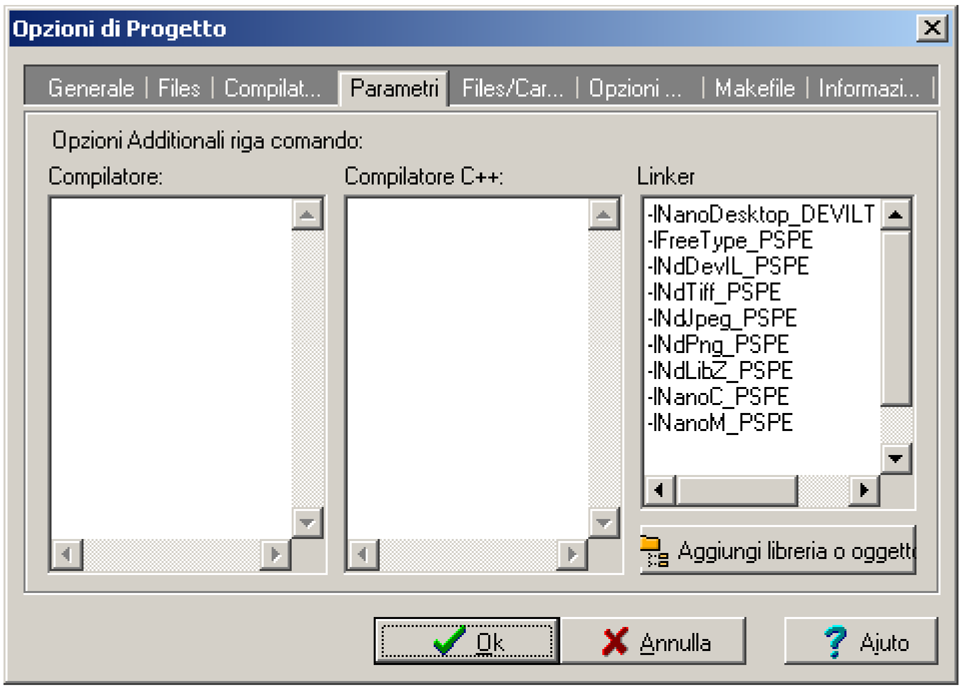


```
E:\NDENV\PSP\SDK\PspDev\psp\include
E:\NDENV\PSP\SDK\PspDev\psp\sdk\include
E:\NDENV\PSP\SDK\NanoCore\NanoC\src
E:\NDENV\PSP\SDK\NanoCore\NanoM\src
E:\NDENV\PSP\SDK\Nanodesktop\src
E:\NDENV\PSP\ndHighGUI\src
E:\NDENV\PSP\ndOpenCV\cxcore\include
E:\NDENV\PSP\ndOpenCV\cv\include
E:\NDENV\PSP\ndOpenCV\cvaux\include
E:\NDENV\PSP\ndDevIL\DevILCore\include\IL
E:\NDENV\PSP\ndLibCurl\include
E:\NDENV\PSP\ndFlite\include
```

Add **<ndenv path>\PSP\ndFreeType\include**

Third step: In your project you must link a version of Nanodesktop that supports DEVIL and TTF.

Keep in mind that if you enable TTF, you must also enable DEVIL, so you have to link also the libraries required by DEVIL for images support (see page 70).


Opzioni di Progetto
Generale | Files | Compilat... | Parametri | Files/Car... | Opzioni ... | Makefile | Informazi...
Opzioni Additionali riga comando:
Compilatore:
Compilatore C++:
Linker
Aggiungi libreria o oggett
Ok
Annulla
Aiuto

```
-lNanoDesktop_DEVILT
-lFreeType_PSPE
-lNdDevIL_PSPE
-lNdTiff_PSPE
-lNdJpeg_PSPE
-lNdPng_PSPE
-lNdLibZ_PSPE
-lNanoC_PSPE
-lNanoM_PSPE
```

In other words, you must link the following libraries:

– *for PSPE platform:*

  -lndDevIL_DEVILTTF_PSPE -lFreeType_PSPE -lndJpeg_PSPE -lndLibz_PSPE -lndPng_PSPE -lndTiff_PSPE

– *for PSP platform:*

  -lndDevIL_DEVILTTF_PSP -lFreeType_PSP -lndJpeg_PSP -lndLibz_PSP -lndPng_PSP -lndTiff_PSP

– *for KSU PSP:*

  -lndDevIL_DEVILTTF_KSU_PSP -lFreeType_PSP -lndJpeg_PSP -lndLibz_PSP -lndPng_PSP -lndTiff_PSP

– *for CFW PSP:*

  lndDevIL_DEVILTTF_CFW_PSP -lFreeType_PSP -lndJpeg_PSP -lndLibz_PSP -lndPng_PSP -lndTiff_PSP

*(Be careful: in the previous notes **-l** means **-elle** and not -i)*

Ok, now you're ready to use TrueType fonts.

The first example

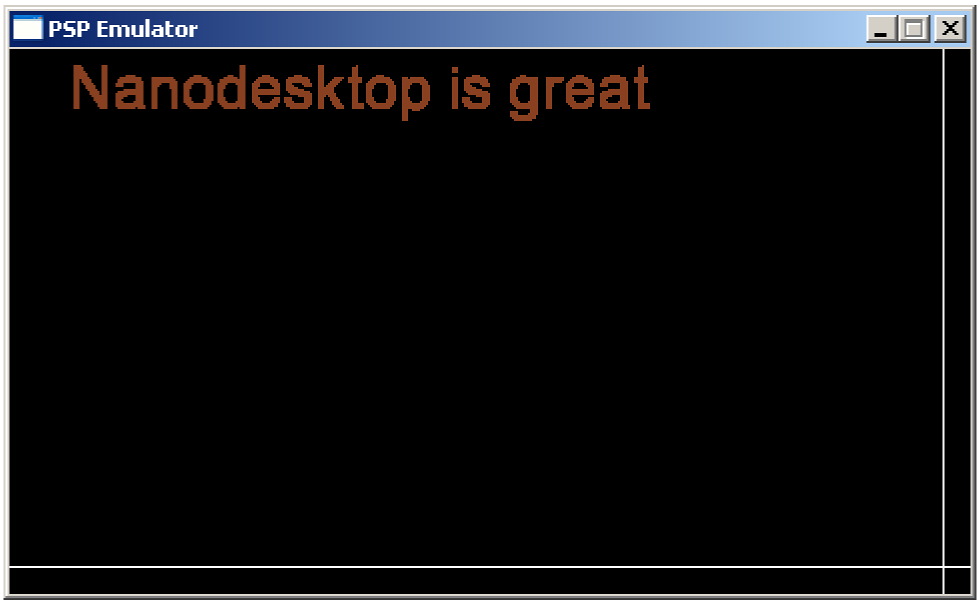
Now, we'll see a simple example about font usage. We want to use the font *Arial.TTF* in way of writing a string in the base screen. The first thing to do is to load the font in memory: so the system will able to assign a *font handle* to the new font.

Let's see this source:

```
#include <nanodesktop.h>

int ndMain()
{
    int FntHandle1;

    ndInitSystem ();

    FntHandle1=ndFONT_LoadFont ("ms0:/arial.ttf");

    ndBS_GrWriteLn (30,  30, "Nanodesktop is great", COLOR_RED, COLOR_BLACK,
                              NDKEY_FONT (FntHandle1) | NDKEY_SETFNTSIZE (30) | RENDER | TRASP);

    ndFONT_DestroyFont (FntHandle1);
}
```

ndFONT_LoadFont loads the font from the disk and assigns to it a handle. If the returned value is negative, there is an error. *ndFONT_DestroyFont* releases the font, frees the handle and the associated memory .

The most important thing is the *64 bit Options* parameter passed to *ndBS_GrWriteLn*. We see a new key called *NDKEY_SETFNTSIZE (x).* This key specifies the size of the chars that must be shown. The size is expressed in pixels. In this case, the chars of the TrueType fonts have dimensions of 30x30 pixels. This is the result:



The key *NDKEY_SETFNTSIZE (x)* isn't used (and, in fact, the system ignores it) when you use an NTF (Nanodesktop text format) font instead of a TTF font, because in that context it should be lack of sense. In fact, all NTF fonts provide chars of 8x8 pixels.

So, only TTF fonts have customizable sizes.

A second point that has to be noted by the reader. **When you use a NTF font, the parameters RRPosPixelX and RRPosPixelY (see ndBS_GrWriteLn prototype) indicate the upper left corner of the rectangle that will contain the written string.**

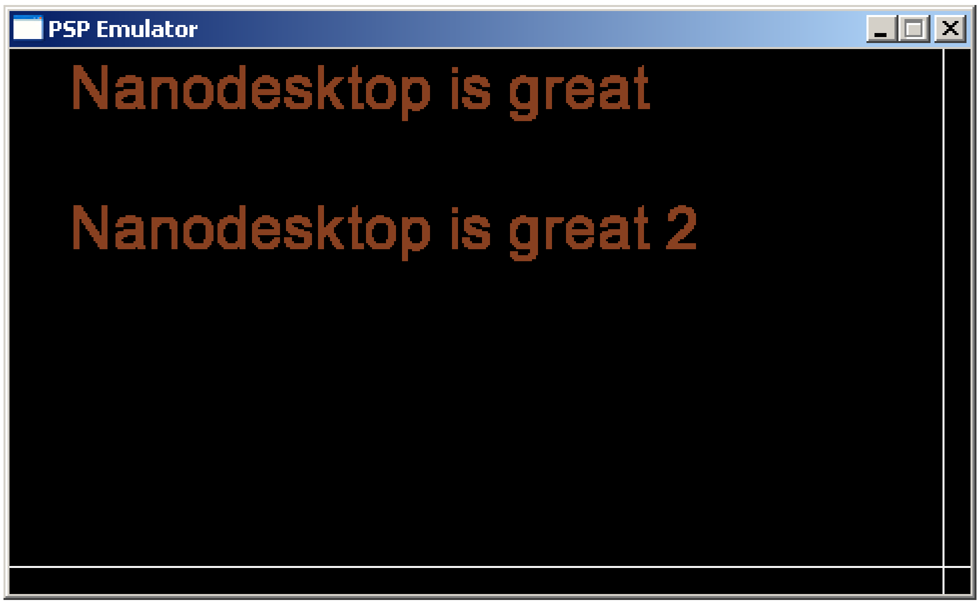
When you use a TTF font, instead, the parameters RRPosPixelX and RRPosPixelY indicate the *lower* left corner of the rectangle that will contain the written string.

Nanodesktop stores the sizes of the chars that you have chosen in your last call. So, if you don't specify the size of the char in the following calls, the system will use automatically the last dimensions .

See this example:

```
#include <nanodesktop.h>


int ndMain()
{
    int FntHandle1;

    ndInitSystem ();

    FntHandle1=ndFONT_LoadFont ("ms0:/arial.ttf");

    ndBS_GrWriteLn (30,  30, "Nanodesktop is great", COLOR_RED, COLOR_BLACK,
                              NDKEY_FONT (FntHandle1) | NDKEY_SETFNTSIZE (30) | RENDER | TRASP);

    ndBS_GrWriteLn (30,  100, "Nanodesktop is great 2", COLOR_RED, COLOR_BLACK,
                              NDKEY_FONT (FntHandle1) | RENDER | TRASP);


    ndFONT_DestroyFont (FntHandle1);
}
```

As you see, in the second call the programmer doesn't specify the size of the font. Nd uses the last size defined (30 pixels*30 pixels):

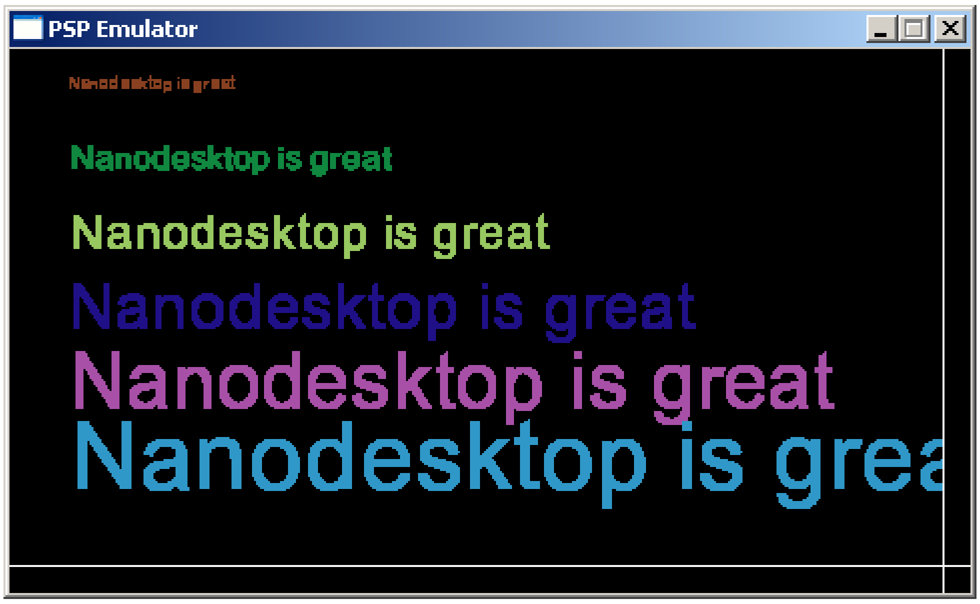


If you have never defined a custom size, nd will use a *default size of 16*16 pixels.*

The key NDKEY_SETFNTSIZE can be used also for interesting graphical effects. See this code:

```
#include <nanodesktop.h>


int ndMain()
{
    int FntHandle1, Counter;

    ndInitSystem ();

    FntHandle1=ndFONT_LoadFont ("ms0:/arial.ttf");

    for (Counter=1; Counter<7; Counter++)
    {

        ndBS_GrWriteLn (30,  20+40*(Counter-1), "Nanodesktop is great", Counter*0x1111, COLOR_BLACK,
                                  NDKEY_FONT (FntHandle1) | NDKEY_SETFNTSIZE (8*Counter) | RENDER |
TRASP);

    }

    ndFONT_DestroyFont (FntHandle1);
}
```

Here, the programmer uses the key NDKEY_SETFNTSIZE with different values. This is the result:



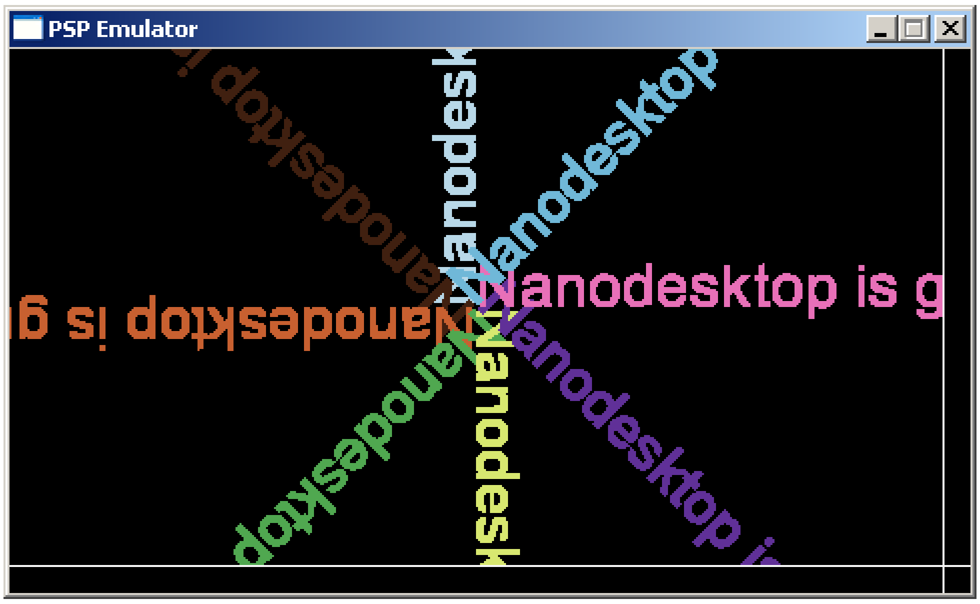
Rotation

Nanodesktop allows also to visualize a string using a rotation. In this way, you can visualize your string in oblique or vertical position... See this example:

```
#include <nanodesktop.h>


int ndMain()
{
    int FntHandle1, Counter;
    int Rot;
    int ColorMul;
    int Color;
    int ErrRep;

    ndInitSystem ();

    FntHandle1=ndFONT_LoadFont ("ms0:/arial.ttf");

    for (Counter=0; Counter<1000; Counter++)
    {
       Rot = (Counter % 8)*(360/8) - 180;

       ColorMul = (Counter+1) % 15;
       Color    = 0x1111 * ColorMul;

        ndBS_GrWriteLn (Windows_MaxScreenX/2,  Windows_MaxScreenY/2, "Nanodesktop is great", Color,
                         COLOR_BLACK, NDKEY_FONT (FntHandle1) | NDKEY_SETFNTSIZE (30)
                         | NDKEY_SETFNTROTATE (Rot) | RENDER | TRASP);


    }

    ndFONT_DestroyFont (FntHandle1);
}
```

Note the new 64-bit key NDKEY_SETFNTROTATE (x). x is a parameter bounded between -180 and 180 degrees. It defines the orientation of the drawn text.

The source changes the orientation using the *for* cycle. The result is the following:



TrueType fonts in windows

Now, we'll see how to use TrueType fonts in a window.

For first thing, keep in mind that you can use them *only in window space, and never in char map*. So, only *ndWS_GrWriteChar, ndWS_GrWriteLn, ndWS_GrPrintLn* can use TrueType fonts.

The other routines, as ndWS_WriteLn **will use only NTF fonts** and they will report an error if you attempt to use them with a TTF.

Second point: **you cannot use TrueType font for window title or for window menus. This limitation is imposed by technical reasons**.

Well, the usage of TTF fonts in window space is very similar to one seen for base screen. The routine ndWS_GrWriteLn has prototype:

```
char ndWS_GrWriteLn (short unsigned int RRPosPixelX, short unsigned int RRPosPixelY, char *str,
TypeColor Color, TypeColor BGColor, unsigned char WndHandle, ndint64 TextCode);
```

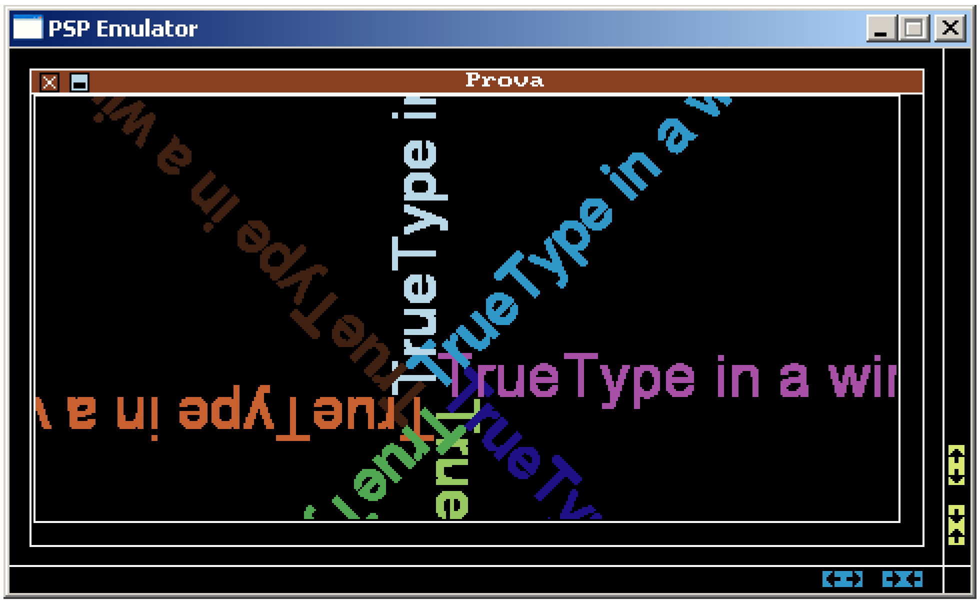
The parameter TextCode accepts the 64-bit key NDKEY_SETFONT (x), NDKEY_SETFNTSIZE (x) and NDKEY_SETFNTROTATE (x). So, if you want to use TTF fonts in a window, you have simply to use the previous keys.

See this program:

```
#include <nanodesktop.h>


int ndMain()
{
    int FntHandle1, Counter, WndHandle;
    int Rot;
    int ColorMul;
    int Color;
    int ErrRep;

    ndInitSystem ();
    WndHandle = ndLP_CreateWindow (10, 10, Windows_MaxScreenX - 10, Windows_MaxScreenY - 10,
                                   "Prova", COLOR_WHITE, COLOR_RED, COLOR_BLACK, COLOR_WHITE, 0);

    ndLP_MaximizeWindow (WndHandle);


    FntHandle1=ndFONT_LoadFont ("ms0:/arial.ttf");
    for (Counter=0; Counter<1000; Counter++)
    {
       Rot = (Counter % 8)*(360/8)-180;

       ColorMul = (Counter+1) % 15;
       Color    = 0x1111 * ColorMul;

       ndWS_GrWriteLn (200,  150, "TrueType in a window. Here", Color, COLOR_BLACK, WndHandle,
                       NDKEY_FONT (FntHandle1) | NDKEY_SETFNTSIZE (30) | NDKEY_SETFNTROTATE (Rot) |
                       RENDER | TRASP);


    }

    ndFONT_DestroyFont (FntHandle1);
}
```

The section in red creates a new window. See the routine ndWS_GrWriteLn and the use of the 64-bit keys in the last parameter.

This is the result:



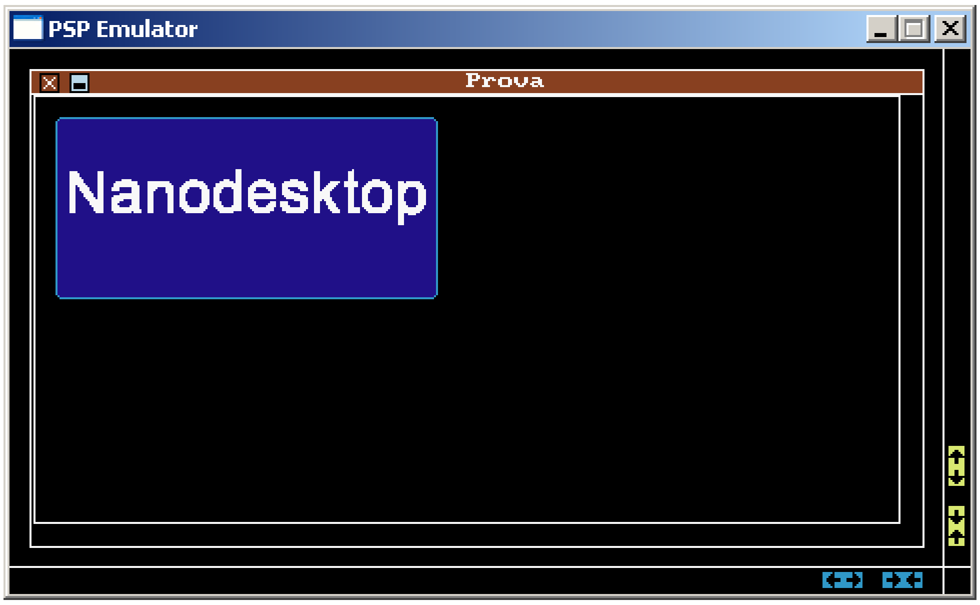
TrueType fonts in buttons

You can use TrueType fonts also in graphical elements as buttons. See this example:

```
#include <nanodesktop.h>


int ndMain()
{
    int FntHandle1, Counter, WndHandle, BtnHandle;
    int Rot, ColorMul, Color;
    int ErrRep;

    ndInitSystem ();
    WndHandle = ndLP_CreateWindow (10, 10, Windows_MaxScreenX - 10, Windows_MaxScreenY - 10,
                                   "Prova",COLOR_WHITE, COLOR_RED, COLOR_BLACK, COLOR_WHITE, 0);

    ndLP_MaximizeWindow (WndHandle);

    FntHandle1=ndFONT_LoadFont ("ms0:/arial.ttf");

    BtnHandle = ndCTRL_CreateButton               ( 10, 10, 200, 100,  WndHandle, "Button0",
                                                    "Nanodesktop", "",
                                                    COLOR_WHITE, COLOR_BLUE, COLOR_LBLUE,
                                                    NDKEY_FONT (FntHandle1) | NDKEY_SETFNTSIZE (30)
                                                    ND_BUTTON_ROUNDED, 0, 12345678, RENDER
       );

    ndFONT_DestroyFont (FntHandle1);
}
```

Note that the program uses the key NDKEY_SETFNTSIZE in ButtonStyle parameter. In this way, the programmer passes to nd an information about the font that has to be used in the button.

See the result:



Note: when the user wants to use a TTF font in a button, the text contained in the second button label (in our example *in green*) is **always ignored**.

Chapter 22
File managers

The goal of a File Manager is to allow the user to choose a file. From the programmer's point of view, a File Manager routine returns *always* a string that contains a name of a file.

Nanodesktop provides file manager routines. All file managers are available through the following routine:

```
char FileManager (char *Title, unsigned char TypeManager, unsigned char SaveMode, char
*StringOutputAddr)
```

The string *Title* is the message that will be written in the title bar of the manager.
TypeManager is a byte that identifies the type of file manager that you want;
SaveMode indicates if the programmer wants that the FileManager works in save mode;
**StringOutputAddr* is a pointer to the string that will contain the name of the file that has been chosen by the user.

When the program calls the FileManager, the caller thread is interrupted until the user has chosen the file.

If the user selects the file, the long name of it (path + name) is copied in the StringOutputAddress and the routine returns to the thread the value 0 (no errors).
If the user has interrupted the operation (for example, closing the files manager window), the routine copies to StringOutputAddress a void string and returns an error code *ERR_OPERATION_INTERRUPTED*

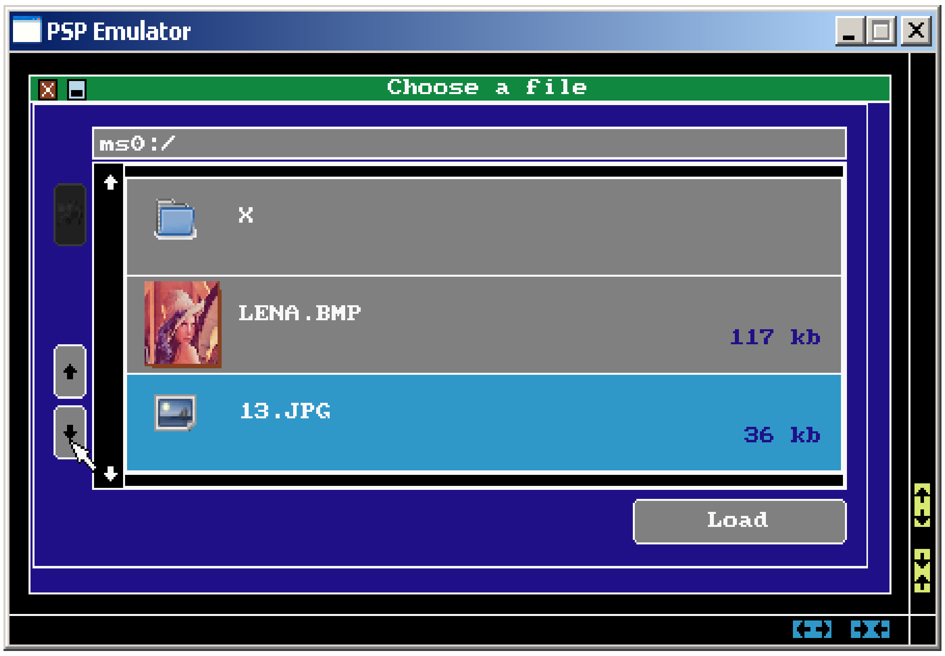
At the moment of the release of the last version of Nanodesktop (3.2.0), the system provides two types of FileManager: *T1FileManager* (code 0) and *T2FileManager* (code 1).

T1 File Manager

A simple example :

```
#include <nanodesktop.h>

int ErrRep;
char NameFile [256];

int ndMain()
{
   ndInitSystem ();

   ErrRep = FileManager ("Choose a file", 0, 0, &NameFile);

   if (!ErrRep)
   {
       printf ("The file that you have choosen is %s \n", NameFile);
   }
   else
   {
       printf ("Operation interrupted by the user \n");
   }
}
```

We have chosen file manager type 0 (*T1FileManager*) as we can see by the number in red; *SaveMode* is disabled.

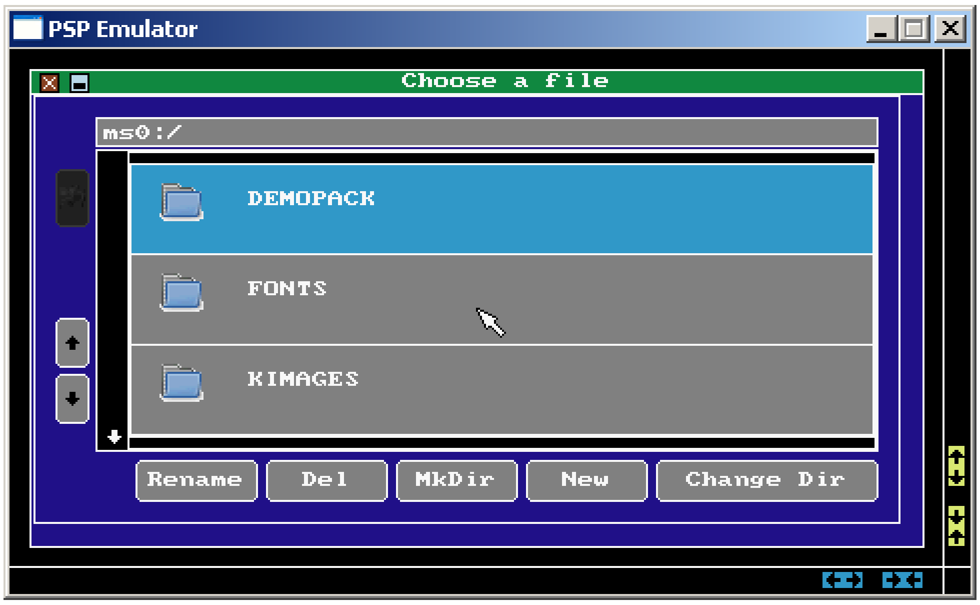


Each file manager can be started in *normal mode*, or in *save mode*. When a file manager is started in save mode, the system enables options that allow the user to modify the file system.

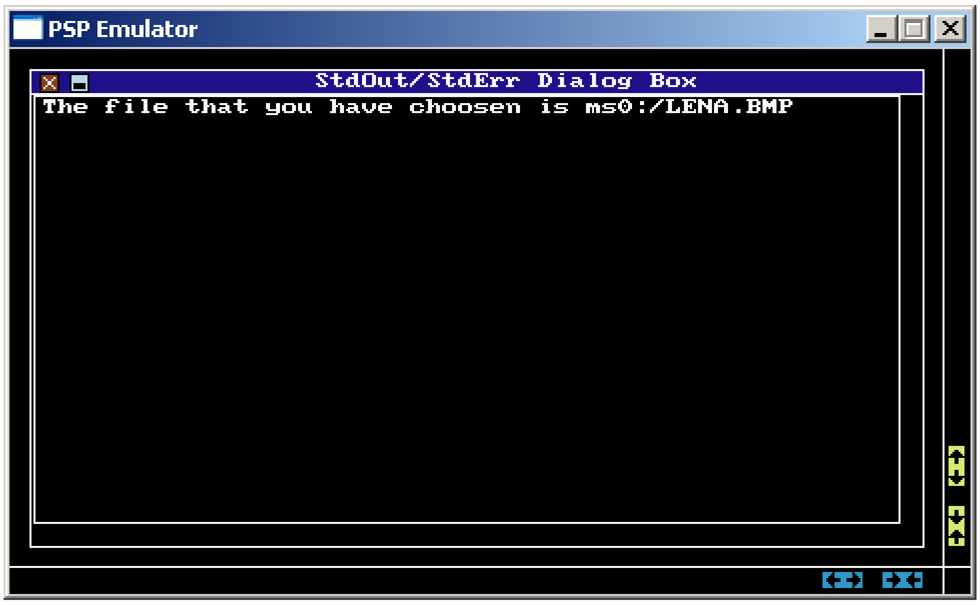
For example:

```
ErrRep = FileManager ("Choose a file", 0, 1, &NameFile);
```

starts the T1FileManager in save mode:

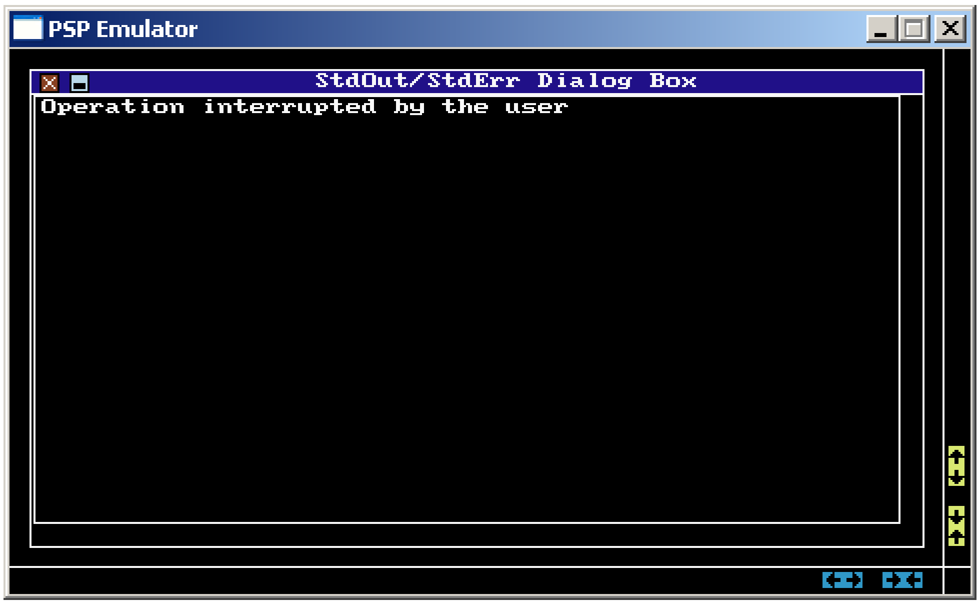


In the previous program the output is this: here we have chosen a file (LENA.BMP)



```
The file that you have choosen is ms0:/LENA.BMP
```

Here, instead, we have closed the file manager window before choosing a new file:



```
Operation interrupted by the user
```

ndFM_EnableThumbnail, ndFM_DisableThumbnail

T1FileManager creates a thumbnail of any image file that the system recognizes: so, you can disable this features using the routine

```
void ndFM_DisableThumbnail (void)
```

And you can re-enable it using the routine:

```
void ndFM_EnableThumbnail (void)
```

Chapter 23
Sound

Nanodesktop is able to manage the sound synthesizer of the Playstation Portable. The library manages either the raw sound, or the decoding of MP3/WAV files.

Using the raw synthesizer

For first thing, we learn how to use the raw synthesizer to produce a simple sound. Keep in mind that the sound functions of Nanodesktop are available only when you use the *KSU mode* (for PSP firmware 1.5) or the *CFW mode* (for the PSP with custom firmware). So, if you wants to use the following program, you must compile it in KSU o CFW mode (see chapter 4 for further details).

The first thing to do, is to obtain a *sound handle* by the system. The sound handle, defines a sound channel that is available in that platform. PSP, for example, supports till 8 channels for sound at the same time: each channel can be assigned to a sound handle.

The function that assigns a sound handle is the following:

```
int ndHAL_SND_OpenChannel (int NrDevice, int Frequency, int NrBits, int NrChannels, int VolumeSx,
int VolumeDx, ndint64 Options)
```

The parameter *NrDevice,* specifies the number of channel for Media Engine chip. It is bounded between 0 and 7. If you specify -1, nd assigns the first channel that isn't used in that moment.

The parameter *Frequency,* is the frequency of the channel. The correct values are 8000, 11025, 12000, 16000, 22050, 24000, 32000, 44100, 48000 (Hz). Nanodesktop is able to interpolate the sound and to adapt it to the standard frequency of the audio chip.

The parameter *NrBits,* can be 8 or 16 bit. It represents the bit depth of the data.

The parameter *NrChannels,* defines if the channel is MONO or STEREO. If you want a mono channel, use the constant *ND_AUDIO_MONO*, otherwise you use *ND_AUDIO_STEREO.*

The parameters *VolumeSx* and *VolumeDx,* define the volume of each subchannel. The values are bounded between 0 (mute) and 100 (full voice). If you've choosen the mono channel, the parameter *VolumeDx* is ignored.

The parameter *Options* is normally used for internal uses: you can simply put it to 0.

Well, let's imagine that you have assigned an handle to the channel with ndHAL_SND_OpenChannel. Now, you have to send to this channel your data. Nd provides the function ndHAL_SND_WriteToChannel:

```
 int ndHAL_SND_WriteToChannel (int NrChannel, void *samples, int NrSamples, float AmpFactor)
```

This function “writes” the data pointed by *samples to the channel *NrChannel.*
The parameter NrSamples is the number of samples and the float AmpFactor is an amplification factor, that is reserved for special uses (put it to 1 if you don't want to use it).

The routine returns a negative value if there is an error, and a positive value (the number of samples that have been reproduced) if there is no error.

When you have reproduced all samples, you can close the sound channel, using the routine:

```
int ndHAL_SND_CloseChannel (int NrChannel)
```

It releases the channel and makes available the sound handle.

A simple program

Now, we see a simple program that generates a sinusoid with a given frequency, and that sends it to a sound channel.

```
 #include <nanodesktop.h>

short int SndBuffer [10*8000];

int ndMain ()
{
    int SndHandle;
    int Counter, CounterB;
    float Angle;
    float Frequency;

    ndInitSystem ();

    Frequency = 100;       // 100 Hz

    // Prepare the samples

    for (Counter=0; Counter<10; Counter++)
    {
        for (CounterB=0; CounterB<8000; CounterB++)
        {
            Angle = Frequency * 6.28 * CounterB / 8000.0;
            SndBuffer [Counter*8000+CounterB] = (int)(32767 * sin (Angle));
        }
    }

    // Generate the samples

    SndHandle = ndHAL_SND_OpenChannel (-1, 8000, 16, 1, 100, 0, 0);

    if (SndHandle>=0)
    {
        ndHAL_SND_WriteToChannel (SndHandle, &SndBuffer, 10*8000, 1);
    }

    ndHAL_SND_CloseChannel (SndHandle);
}
```

This program fills a buffer with the data of a sinusoid, and after it provides to reproduce these data.

ndHAL_SND_SetVolume

This routine provides to change the volume of a given channel:

```
int ndHAL_SND_SetVolume (int NrChannel, int VolumeSx, int VolumeDx)
```

ndHAL_SND_FlushChannel

Provides to flush the internal audio buffer that Nanodesktop assigns to a given channel.

```
int ndHAL_SND_FlushChannel (int NrChannel)
```

Chapter 24

Decoding MP3/WAV Files

Nanodesktop is able to use *PSP Media Engine chip* to decode multimedia files, as WAVE or MP3. This function is available only in CFW or KSU mode.

ndHAL_MEDIA_LoadFile

The first thing to do, is to load the multimedia file, using the following routine:

```
int ndHAL_MEDIA_LoadFile (char *NameFile, int NrChannel, struct ndMediaObj_Type *ndMediaObj, void
*CallBack, ndint64 InfoField)
```

In this routine, the parameter *NameFile* is the name of the file that has to be loaded.

The parameter *NrChannel* is the ID of the channel that will reproduce the audio flux. This number is bounded between 0 and 7. If you use -1, the system will choice the first audio channel that is available.

The *struct ndMediaObj*, is a struct that will contain the informations about the media file. When you have loaded the media file, all following calls are executed passing the address of the struct ndMediaObj. Effectively, ndHAL_Media_LoadFile assigns a given media file to a given struct ndMediaObj_Type.

The pointer **Callback* points to a *media callback.* This is a routine like this:

```
int MediaCallback (ndint64 InfoField, struct ndMediaObj_Type *ndMediaObj,  int NrSample, int HH, int
MM, int SS)
```

The callback receives in input the InfoField information, the address of the struct ndMediaObj, the number of samples that have been reproduced till now, and the number of seconds (HH/MM/SS) that have passed till that moment. If you use in a smart mode the media callback, you can create applications that manages the MP3/WAV with histograms or progress bars.

If you set to 0 the parameter *Callback,* you will disable the callback.

The parameter *InfoField,* is a 64 bit value that will be passed to the media callback each time the routine is recalled.

Usually, a call to ndHAL_MEDIA_LoadFile appears like this:

```
struct ndMediaObj_Type ndMediaObj;

Result = ndHAL_MEDIA_LoadFile ("ms0:/MySong.MP3", -1, &ndMediaObj, 0, 0);
```

If the process of loading media file is successful, ndHAL_MEDIA_LoadFile returns the value 0 to the main program. All informations that you need about the audio file, are in the struct ndMediaObj_Type:

```
  struct ndMediaObj_Type
  {
      char  NameFile [262];
      char  Type;

      int   ndSndChannel;

      int   DecThreadID;
      int   DecEventID;

      int   DecoderState;
      int   old_DecoderState;

      int   DecoderError;
      int   DecoderExitRq;

      void*   CallBack;
      ndint64 InfoField;
      ndint64 Flags;

      int   NrTimesPlay;

      int   MediaHandle;

      int                   MP3_Channels;           // 1 - mono, 2 - stereo
      int                   MP3_SampleRate;         // 8000..44100
      int                   MP3_HwSampleRate;
      int                   MP3_HwSampleSize;
      int                   MP3_HwOutputMode;       // ND_MONO, ND_STEREO
      int                   MP3_SamplesForFrame;
      int                   MP3_FrameSize;
      int                   MP3_StereoMode;         // 0 Stereo, 1 Joint stereo, 2 Dual channel,
                                                     // 3 Mono
      int                   MP3_GetEDRAM;
      int                   MP3_Version;
      int                   MP3_Layer;
      int                   MP3_BitRateSeq;
      int                   MP3_BitRateIndex;
      int                   MP3_BitRate;
      int                   MP3_Padding;
      int                   MP3_FrameCostant;
      int                   MP3_SamplesDecoded;

      void                 *MP3_DataStart;
      unsigned long        *MP3_CommCodecArea;
      short int            *MP3_L1Buffer;
      char                 *MP3_FlexibleDataBuffer;
      char                  MP3_FlexibleDataBufferCreated;
      int                   MP3_FlexibleDataBufferOldSize;

      int                   WAV_Channels;           // 1 - mono, 2 - stereo
      int                   WAV_SampleRate;         // 8000..44100
      int                   WAV_HwSampleRate;
      int                   WAV_HwSampleSize;
      int                   WAV_HwOutputMode;       // ND_MONO, ND_STEREO
      int                   WAV_HwOutputBitRate;    // 8-16
      int                   WAV_BlockSize;
      int                   WAV_BitsForSample;
      int                   WAV_DataLength;
      int                   WAV_PosStartData;
      int                   WAV_SamplesDecoded;
      int                   WAV_SamplesRead;
      int                   WAV_SamplesInTotal;
      int                   WAV_HeaderError;

      void                  *WAV_L1Buffer;

};
```

ndHAL_MEDIA_Play

When you have loaded the media file with ndHAL_MEDIA_LoadFile, you can reproduce the file. For this task, you can use the routine

```
int ndHAL_MEDIA_Play (struct ndMediaObj_Type *ndMediaObj, ndint64 Options)
```

The first parameter is the struct that contains the informations about the file; the second parameter is a 64 bit value, bit-mapped, that contains some options about the reproduction.

The options available (if you want, otherwise set Options to 0) are two:

a) *ND_PLAY_AUTOREWIND*. In this case, when the system will arrive at the end of the media file, it will restart automatically the reproduction;

b) *ND_LOCK_UNTIL_SONG_FINISH*. In this case, the thread that contains the call to ndHAL_Media_Play, remains locked until the system has finished the reproduction of the media file.

ndHAL_MEDIA_Stop and ndHAL_MEDIA_Pause

The two routines have the same behaviour, and you can use the first or the second indifferently.

```
int ndHAL_MEDIA_Pause (struct ndMediaObj_Type *ndMediaObj)
int ndHAL_MEDIA_Stop (struct ndMediaObj_Type *ndMediaObj)
```

The routines stop the reproduction of the file.

ndHAL_MEDIA_GoToSample

This routine stops the reproduction of the file and moves the pointer to the indicated sample of the file.

```
int ndHAL_MEDIA_GoToSample (struct ndMediaObj_Type *ndMediaObj, int NrSample)
```

It is also possible to move the pointer to a precise instant of the file:

```
int ndHAL_MEDIA_GoToAnIstant (struct ndMediaObj_Type *ndMediaObj, int HH, int MM, float SS)
```

ndHAL_MEDIA_GetNrSamples

This routine returns the number of samples that are present in a multimedia file.

```
int ndHAL_MEDIA_GetNrSamples (struct ndMediaObj_Type *ndMediaObj)
```

With WAV files, the number can be returned immediately; for MP3 files, the system has to scan the entire archive to obtain the total number of samples that are present.

ndHAL_MEDIA_UnLoad

When you have reproduced the media file, you must release it before loading a new file. Nanodesktop provides a function called:

```
int ndHAL_MEDIA_UnLoad (struct ndMediaObj_Type *ndMediaObj)
```

This function releases the file and deassociate it from the struct ndMediaObj. When you want to load a new file, first release the old one, and after load the new one.

Chapter 25
Webcam support

The Nanodesktop applications integrate the support for the webcams. This means that the programmers that use Nanodesktop can access to the cameras and that they can grab the images and elaborate them in their programs.

The support for cameras is available only in KSU or CFW mode (see chapter 4).

When you use *Kernel Services To User mode,* the support for camera is provided using Eyeserver software (see Nanodesktop website). A software that runs on PC (or, in the future, on an embedded chip as *Fox board*), sends to Eyeclient (integrated in the code of Nanodesktop application) the images that have been collected by a webcam connected with the PC. This approach is only experimental: it will become more useful when Visilab will released versions of Eyeserver that run on FoxBoard.

In order to use Eyeserver under KSU applications, the user must copy a driver, called *ndUsbDriver.Prx,* in the root directory of the memory stick. This driver is downloadable from Visilab website, and it is present also in Nanodesktop distribution, in the folder:

```
<ndenv path>\PSP\SDK\Nanodesktop\src\3rdparty_modules\USB_PRX_Driver\prx
```

When you use *CFW mode* (i.e you have compiled your program to work under custom firmwares), the application can use either *Eyeserver support* or *GoCam support*. Eyeserver support is identical to one that is present in KSU mode: the only difference is that CFW programs require a different driver in the root folder of the memory stick, called *ndUsbDriverCF.Prx.*

E:\NDENV\PSP\SDK\Nanodesktop\src\3rdparty_modules\USB_PRX_Driver_for_CF\prx

If you don't want to use Eyeserver support, you can use GoCam support. Nanodesktop integrates the interface for the drivers for Sony GoCam. The programmers can access to the webcam and they can grab the images from it.

ndHAL_CAM_ActivateCamera

The first thing to do is to activate the camera. Nanodesktop provides a dedicated routine to do this: ndHAL_CAM_ActivateCamera.

```
int ndHAL_CAM_ActivateCamera (int NrCamera)
```

The system will try to do this and, if the operation is successful, it returns a handle that identifies the camera in all following operations. The returned handle is always non negative: if *ndHAL_CAM_ActivateCamera* returns a negative value, it means that there is an error. Here are some examples:

```
CameraHandle = ndHAL_CAM_ActivateCamera (0);
```
It tries to activate Sony GoCam and, if it isn't available, attempts to activate an Eyeserver connection.

```
CameraHandle = ndHAL_CAM_ActivateCamera (ND_USE_PSP_GOCAM);
```
It tries to activate Sony GoCam and, if it isn't available, returns an error code to the program.

```
CameraHandle = ndHAL_CAM_ActivateCamera (ND_USE_PSP_EYESVR);
```
It tries to activate an Eyeserver connection and, if it isn't available, returns an error code to the program.

Using ndHAL_ActivateCamera, you can manage, in a very simple way, the connection with your camera.

ndHAL_CAM_DisableCamera

This routine provides to deactivate the camera. The prototype is this:

```
void ndHAL_CAM_DisableCamera (int UsbHandleCamera)
```

ndHAL_CAM_GrabNewImage

If you want to grab an image from the webcam, you can use the function ndHAL_CAM_GrabNewImage. This single routine can execute all operations of grabbing that are available on Nanodesktop. The prototype of the function is this:

```
char ndHAL_CAM_GrabNewImage (int UsbHandleCamera, char Target, int AuxValue, ndint64 Options, char
RenderNow)
```

The first parameter is UsbHandleCamera: it is the handle returned by ndHAL_CAM_ActivateCamera at the time of activation.

The parameter *Target,* specifies where the collected data must be sent. There are 4 manifested constants that can be used for Target parameter:

ND_CAM_TO_WINDOW	The image must be sent to a window
ND_CAM_TO_BASESCREEN	The image must be sent to the basescreen
ND_CAM_TO_NDIMAGE_STRUCT	The image must be sent to a Nanodesktop image
ND_CAM_TO_CVIMAGE	The image must be sent to a OpenCV image

The parameters *AuxValue*, *Options* and *RenderNow*, have different meanings in base of the value of Target parameter.

Grab to the BaseScreen

Now, we'll see an example of grabbing to the BaseScreen.

When Target is set to *ND_CAM_TO_BASESCREEN,* the parameter AuxValue is ignored.

This is a simple program that provides to grab 1000 frames from the GoCam and to send them to the BaseScreen:

```
#include <nanodesktop.h>

int ndMain()
{
    int UsbHandleCamera;
    int Counter;

    ndInitSystem ();

    UsbHandleCamera = ndHAL_CAM_ActivateCamera (ND_USE_PSP_GOCAM);

    if (UsbHandleCamera>=0)
    {
       for (Counter=0; Counter<1000; Counter++)
       {
           ndHAL_CAM_GrabNewImage (UsbHandleCamera, ND_CAM_TO_BASESCREEN, 0, 0, RENDER);
       }
    }
    else
    {
       printf ("GoCam is not connected \n");
       ndDelay (2);
    }
}
```

The parameter *Options* can be used to pass informations about position. For example, using the 64-bit key *NDKEY_POSX (x)* | *NDKEY_POSY (y)*, you can say to the system that the image must be drawn in the base screen at the coordinates (x;y) (they are the coordinates of the upper left corner of the image).

The following program is identical to the previous one, but the image is drawn at the pixel (20;20) of the BaseScreen:

```
#include <nanodesktop.h>

int ndMain()
{
    int UsbHandleCamera;
    int Counter;

    ndInitSystem ();

    UsbHandleCamera = ndHAL_CAM_ActivateCamera (ND_USE_PSP_GOCAM);

    if (UsbHandleCamera>=0)
    {
       for (Counter=0; Counter<1000; Counter++)
       {
               ndHAL_CAM_GrabNewImage  (UsbHandleCamera,  ND_CAM_TO_BASESCREEN,  0,  NDKEY_POSX  (20)  |
NDKEY_POSY (20), RENDER);
       }
    }
    else
    {
       printf ("GoCam is not connected \n");
       ndDelay (2);
    }
}
```

Grab to a window space

You can grab the image and send it to a window space. In this way, the image will appear in a window and not in the base screen.

To do this, you can set the *Target* parameter in ndHAL_CAM_GrabNewImage at the value *ND_CAM_TO_WINDOW.* The parameter AuxValue must be set to the *window handle,* i.e to the handle of the window that will receive the new video informations.

Let's see this program:

```
#include <nanodesktop.h>

int ndMain()
{
    int UsbHandleCamera;
    int WndHandle;
    int Counter;

    ndInitSystem ();

    UsbHandleCamera = ndHAL_CAM_ActivateCamera (ND_USE_PSP_GOCAM);

    if (UsbHandleCamera>=0)
    {
        WndHandle = ndLP_CreateWindow (10, 10, 300, 250, "Camera result", COLOR_WHITE, COLOR_BLUE,
                                  COLOR_BLACK, COLOR_WHITE, 0);

       if (WndHandle>=0)
       {
           ndLP_MaximizeWindow (WndHandle);
           ndHAL_EnableMousePointer ();

           for (Counter=0; Counter<1000; Counter++)
           {
               ndHAL_CAM_GrabNewImage (UsbHandleCamera, ND_CAM_TO_WINDOW, WndHandle, 0, RENDER);
           }
       }
    }
    else
    {
       printf ("GoCam is not connected \n");
       ndDelay (2);
    }
}
```

Also in this case, we can program nd to draw the image in a precise position in the window space. For example, see this program:

```
#include <nanodesktop.h>

int ndMain()
{
    int UsbHandleCamera;
    int WndHandle;
    int Counter;

    ndInitSystem ();

    UsbHandleCamera = ndHAL_CAM_ActivateCamera (ND_USE_PSP_GOCAM);

    if (UsbHandleCamera>=0)
    {
        WndHandle = ndLP_CreateWindow (10, 10, 300, 250, "Camera result", COLOR_WHITE, COLOR_BLUE,
                                      COLOR_BLACK, COLOR_WHITE, 0);

       if (WndHandle>=0)
       {
           ndLP_MaximizeWindow (WndHandle);
           ndHAL_EnableMousePointer ();

           for (Counter=0; Counter<1000; Counter++)
           {
               ndHAL_CAM_GrabNewImage (UsbHandleCamera, ND_CAM_TO_WINDOW, WndHandle, NDKEY_POSX (20)
                                                 | NDKEY_POSY (20), RENDER);
           }
       }
    }
    else
    {
       printf ("GoCam is not connected \n");
       ndDelay (2);
    }
}
```

Note: the examples that we have seen, will work only in CFW mode (they use the GoCam). If you want to exit from these programs, keep pressed the HOME key on the console for some seconds: this will activate the system halter.

Grab to a Nanodesktop image

Now, we'll see how to grab the new images to a Nanodesktop image (see chapter 9).

In order to grab the new images to a struct ndImage_Type, you must set the *Target* parameter to *ND_CAM_TO_NDIMAGE_STRUCT.* The parameter *AuxValue* will be the address of the struct ndImage_Type that will receive the video informations.

Remember that the Nanodesktop image must be already allocated before calling ndHAL_CAM_GrabNewImage: for this task, you can use *ndIMG_CreateImage.*

See this program:

```
#include <nanodesktop.h>

int ndMain()
{
    struct ndImage_Type MyImage;
    int UsbHandleCamera;
    int WndHandle;
    int Counter;

    ndInitSystem ();

    ndIMG_CreateImage (&MyImage, 280, 230, NDMGKNB);
    // Allocate the new image

    UsbHandleCamera = ndHAL_CAM_ActivateCamera (ND_USE_PSP_GOCAM);

    if (UsbHandleCamera>=0)
    {
         WndHandle = ndLP_CreateWindow (10, 10, 300, 250, "Camera result", COLOR_WHITE, COLOR_BLUE,
COLOR_BLACK, COLOR_WHITE, 0);
```

```
        if (WndHandle>=0)
        {
            ndLP_MaximizeWindow (WndHandle);
            ndHAL_EnableMousePointer ();


            for (Counter=0; Counter<1000; Counter++)
            {
                ndHAL_CAM_GrabNewImage (UsbHandleCamera, ND_CAM_TO_NDIMAGE_STRUCT, &MyImage, 0, 0);
                // Put video informations in the ndImage

                ndIMG_ShowImage (&MyImage, 0, 0, WndHandle, RENDER);
                // Show the ndImage
            }
        }
    }
    else
    {
       printf ("GoCam is not connected \n");
       ndDelay (2);
    }
}
```

Note that it is very simple to create a software that transforms your PSP in a portable photo-camera. For example, you could modify the previous program, in way that the system saves a file into the memory stick when a key is pressed.

```
#include <nanodesktop.h>

int ndMain()
{
    struct ndImage_Type MyImage;
    struct ndDataControllerType ndPadRecord;

    int UsbHandleCamera;
    int WndHandle;
    int Counter;

    ndInitSystem ();

    ndIMG_CreateImage (&MyImage, 280, 230, NDMGKNB);
    // Allocate the new image

    UsbHandleCamera = ndHAL_CAM_ActivateCamera (ND_USE_PSP_GOCAM);

    if (UsbHandleCamera>=0)
    {
         WndHandle = ndLP_CreateWindow (10, 10, 300, 250, "Camera result", COLOR_WHITE, COLOR_BLUE,
COLOR_BLACK, COLOR_WHITE, 0);

        if (WndHandle>=0)
        {
            ndLP_MaximizeWindow (WndHandle);
            ndHAL_EnableMousePointer ();


            while (1)
            {
                ndHAL_CAM_GrabNewImage (UsbHandleCamera, ND_CAM_TO_NDIMAGE_STRUCT, &MyImage, 0, 0);
                // Put video informations in the ndImage

                ndIMG_ShowImage (&MyImage, 0, 0, WndHandle, RENDER);
                // Show the ndImage

                ndHAL_GetPad (&ndPadRecord);
                // Collect informations about pad state

                if (ndPadRecord.Buttons & PSP_CTRL_CROSS)
                {
                    ndIMG_SaveImage (&MyImage, "ms0:/photo.jpg");
                }
            }
        }
    }
    else
    {
       printf ("GoCam is not connected \n");
       ndDelay (2);
    }
}
```

Compile this program in CFW mode with Dev-IL support enabled (see page 70: the library that you need is *-lNanodesktop_DEVIL_CFW_PSP*).

When you press the cross button, nd will save the content of the nd image to the memory stick (file "ms0:/photo.jpg").

Grab to a OpenCV image

Nanodesktop supports also a dedicated version of Intel OpenCV (TM) libraries. These libraries allow the execution of image analysis algorithms, like face recognition, object recognition, segmentation and filtering. In this section, we have to learn how is possible to grab the cam image to a struct IplImage.

```
#include <nanodesktop.h>
#include <ndhighgui.h>

int ndMain()
{
    IplImage *cvImage;
    int UsbHandleCamera;

    cvInitSystem (0, 0);

    UsbHandleCamera = ndHAL_CAM_ActivateCamera (ND_USE_PSP_GOCAM);

    if (UsbHandleCamera>=0)
    {
           cvImage = cvCreateImage( cvSize(300, 240), 8, 3 );
           // Allocate the OpenCV image

           while (1)
           {
               ndHAL_CAM_GrabNewImage (UsbHandleCamera, ND_CAM_TO_CVIMAGE, cvImage, 0, 0);
               // Put video informations in the OpenCV image

               cvShowImage ("OpenCV image", cvImage);
               // Show the OpenCV image
           }

    }
    else
    {
       printf ("GoCam is not connected \n");
       ndDelay (2);
    }
}
```

Compile this source with the following libraries: *-lNanodesktop_DEVIL_CFW_PSP -lndHighGUI_PSP -lndCxCore_PSP -lndCv_PSP -lndCvAux_PSP -lNdDevIL_PSP -lNdTiff_PSP -lNdJpeg_PSP -lNdPng_PSP -lNdLibZ_PSP -lNanoC_PSP -lNanoM_PSP*

The routine ndHAL_CAM_GrabNewImage uses the parameter *Target* ND_CAM_TO_CVIMAGE; the parameter *AuxValue* is a pointer of type IplImage*: the standard image in OpenCV environment.

The space in memory must be allocated *before* grabbing: *cvCreateImage* provides to do this. Note that the video input is RGB 8 bits for pixels: so, the IplImage must be also with 3 channels.

In reality, the ndOpenCV user can utilize the standard OpenCV calls to manage the webcam, as *cvCaptureFromCAM*, *cvGrabFrame*, *cvRetrieveFrame* and *cvQueryFrame*.

ndHAL_CAM_ChangeResolution

You can change the resolution that is used by the webcam using the function:

```
char ndHAL_CAM_ChangeResolution (int UsbHandleCamera, int NewSizeX, int NewSizeY)
```

When you are using Eyeserver, the PSP will send an *information packet* to the Eyeserver that will order the change of resolution. When you are using GoCam, Nanodesktop will change its internal registers to do an interpolation of the image in input and to change the resolution collected by the camera.

ndHAL_CAM_ChangeTransferMode

Nanodesktop can manage a video input in *gray tones,* or in *RGB color.*

When you are using Eyeserver, there are three modes for transferring images through the USB cable between the server and the client:

- the *mode 8*, provides images in 256 gray tones; it is the fastest transfer mode;
- the *mode 16,* provides images in RGB555 format (16 bits for pixel): it is a good compromise between speed and colors fidelity;
- the *mode 24,* provides images in RGB888 format (24 bits for pixel): it is the best quality transfer mode.

Keep in mind that, under PSP platform, there is no real difference between mode 16 and mode 24 (the image will be visualized *always* in RGB555 format). So, the mode 16 is defined by default, because it gives a good compromise between quality and speed.

The transfer mode can be changed using the routine:

```
char ndHAL_CAM_ChangeTransferMode (int UsbHandleCamera, int NewMode)
```

where NewMode can be 8,16 or 24.

When you are using GoCam, the situation is this:

- the *mode 8* gives gray tones images (the driver converts automatically the images that come from the GoCam in gray tones images);
- the *mode 16 and the mode 24* give the same results: they returns images in RGB888 format.

The system is managed in a way that the application hasn't to worry about the differences between Eyeserver and GoCam: the system covers the difference.

Example:

```
ndHAL_CAM_ChangeTransferMode (UsbHandleCamera, 8);
// Switch the camera to gray tones mode
```

ndHAL_CAM_GetInfo

If you want to obtain some informations about the actual status of the camera, you can use the following routine:

```
char ndHAL_CAM_GetInfo (int UsbHandleCamera, int *DimImgX, int *DimImgY, int
*FrameCounter, int *TransferMode)
```

An example:

```
int _DimImgX, _DimImgY, _FrameCounter, _TransferMode;

ndHAL_CAM_GetInfo (UsbHandleCamera, &_DimImgX, &_DimImgY, &_FrameCounter,
&_TransferMode);

printf ("Dimensions of video: %d %d \n", _DimImgX, _DimImgY);
printf ("Frames decoded: %d \n", _FrameCounter);
printf ("Actual transfer mode: %d \n", _TransferMode);
```

Microphone support

The webcams usually have an integrated microphone, that allows the recording of voice and music.

For example, GoCam has a good microphone that can be used by the programmer (note: Eyeserver cam doesn't support any microphone).

Starting from release 0.3.4, Nanodesktop offers some routines that allow the recording of sounds through the cam microphone

ndHAL_CAM_ActivateCamMicrophone

This function activates the microphone that is integrated in the webcam, identified by its *UsbHandle*.

```
char ndHAL_CAM_ActivateCamMicrophone (int UsbHandleCamera, int WorkAreaAudioSize, int Frequency, int
Gain)
```

The parameter *WorkAreaSize* is the size of the memory area that will be allocated internally by nd. Set it to 4096 for normal uses. Frequency and Gain must be supported by the device: for Sony GoCam it is useful to set Frequency to 44100 and Gain to 30.

The routine returns 0 if the operation has been successful, while it returns a negative value in case of errors.

ndHAL_CAM_ReadAudioBlock

This routine reads an audio block from the microphone and stores it into a memory area that has been allocated by the user before.

```
int ndHAL_CAM_ReadAudioBlock (int UsbHandleCamera, void *Buffer, int NrBytesToRead)
```

It returns the number of samples read from the microphone

ndHAL_CAM_DeActivateCamMicrophone

This routine disables the microphone integrated in the cam.
(Every recording job that is active is stopped, see ndHAL_CAM_StopAudioRecording)

```
void ndHAL_CAM_DeActivateCamMicrophone (int UsbHandleCamera)
```

ndHAL_CAM_StartAudioRecording

This routine starts a *recording job.* Nanodesktop internally creates a thread that provides to record the audio flux coming from the microphone and to store it to a file.

```
char ndHAL_CAM_StartAudioRecording (int UsbHandleCamera, char *NameFile, int Frequency, int Gain)
```

The target file is created internally by the routine. The routine returns 0 if there are no errors, and a negative value in case of troubles. See ndHAL_WebCam.c for further informations about the possible error codes.

When the execution returns from *ndHAL_CAM_StartAudioRecording*, the thread is active and it is recording the sound in background. Under PSP it isn't allowed to create two recording jobs in the same time.

When you want to stop the recording, use the function *ndHAL_CAM_StopAudioRecording*.

ndHAL_CAM_StopAudioRecording

This routine stops the recording job that is active for a given camera. The thread is internally deleted and the content is stored to the respective file in memory stick (the file is closed by the routine).

```
void ndHAL_CAM_StopAudioRecording (int UsbHandleCamera)
```

An example

Now, let's see an example of using of the previous routines with the Sony GoCam:

```
#include <nanodesktop.h>


int  UsbHandleCamera;
char YouCanExit;


int ndMain()
{
    struct ndDataControllerType ndPadRecord;
    int WndHandle;
    int Counter;

    ndInitSystem ();
    UsbHandleCamera = ndHAL_CAM_ActivateCamera (ND_USE_PSP_GOCAM);

    if (UsbHandleCamera>=0)
    {
        WndHandle = ndLP_CreateWindow (10, 10, 400, 250, "Camera result", COLOR_WHITE, COLOR_BLUE,
        COLOR_BLACK, COLOR_WHITE, 0);

        if (WndHandle>=0)
        {
            ndLP_MaximizeWindow (WndHandle);
            ndHAL_EnableMousePointer ();

            ndHAL_CAM_StartAudioRecording (UsbHandleCamera, "ms0:/recordsample.wav", 44100, 30);

            YouCanExit=0;

            while (!YouCanExit)
            {
                ndHAL_CAM_GrabNewImage (UsbHandleCamera, ND_CAM_TO_WINDOW, WndHandle, 0, RENDER);
                // Put video informations in the ndImage

                ndHAL_GetPad (&ndPadRecord);
                // Collect informations about pad state

                if (ndPadRecord.Buttons & PSP_CTRL_CIRCLE)
                {
                   YouCanExit=1;
                }
            }

            ndHAL_CAM_StopAudioRecording (UsbHandleCamera);
            ndHAL_CAM_DisableCamera (UsbHandleCamera);
        }
    }
    else
    {
        printf ("GoCam is not connected \n");
        ndDelay (2);
    }
}
```

The previous program starts the audio recording and it stops itself only when the user presses the circle key on PSP. The sound is stored in the file *ms0:/recordsample.wav.*

Chapter 26

Text to speech synthesis

Nanodesktop supports the *Flite engine.* Flite is an experimental synthesis engine for text to speech conversion. Using ndFLite, you can do your PSP speaks !

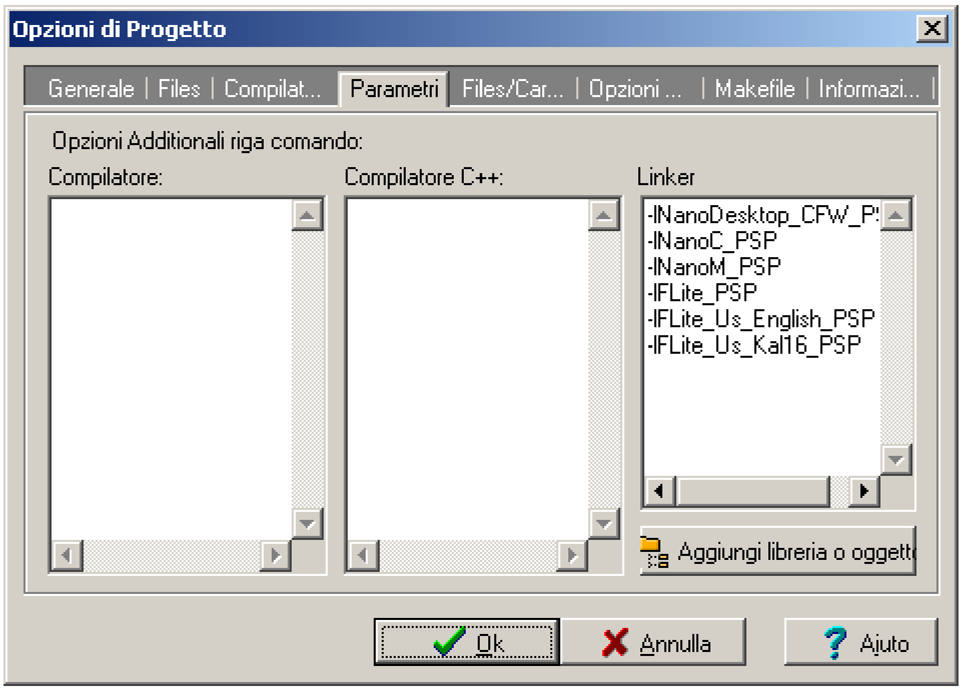
If you want to use the samples that are in this section, you need the media capabilities of nd. So, you must compile the programs in KSU mode (for firmware vers. 1.5) or in CFW mode (for the other custom firmwares). See chapter 4 for further details.

The simplest example

This is the simplest example of text to speech synthesis:

```
#include <nanodesktop.h>
#include "flite.h"

cst_voice *register_cmu_us_kal();
cst_voice *v;

int ndMain()
{
    int Counter;

    ndInitSystem ();
    flite_init();

    v = register_cmu_us_kal ();
    flite_text_to_speech("Hello, Joseph",v,"play");
}
```

It is very important to link the correct libraries:



```
-lNanoDesktop_CFW_P
-lNanoC_PSP
-lNanoM_PSP
-lFLite_PSP
-lFLite_Us_English_PSP
-lFLite_Us_Kal16_PSP
```

The Flite engine is contained in the libraries `-lFLite_PSP`, `-lFLite_Us_English_PSP` and `-lFLite_Us_Kal16_PSP`.

Note that `-lFLite_Us_Kal16_PSP` is the 16-bit version of the engine KAL. It is possible also to use an 8 bit version (`-lFLite_Us_Kal_PSP`). This version is faster and it occupies less memory, but the quality is worse. In theory, Nanodesktop can manage also other vocal engines, with different languages: the only condition is that the module can be recompiled and that it is declared compatible with Flite engine. Please contact Visilab research for further details.

It is also important that the files of the headers (flite.h) and the libraries are found by the compiler and by the linker. So, ensure that the directory ndFLite is set in the directories list (see page 22 and page 23).

After the recompilation of the previous sources, you will obtain a program that says “Hello, Joseph” from your PSP.

Text to speech for a WAVE file

The result of vocal synthesizer can be sent not only to Media Engine chip, but also to a simple .WAV file. This gives some advantages, because these files can be reproduced quickly a second time using the Media API support (see ndHAL_MEDIA_Play - page 189).

Some more advanced nd programs, like *Blind Assistant*, register vocal samples in the memory stick. When the program needs them, the system doesn't have to redo the necessary complex computations, but it limits itself to load and play the wave file that has been generated before.

See this program:

```
#include <nanodesktop.h>
#include "flite.h"

cst_voice *register_cmu_us_kal();
cst_voice *v;

int ndMain()
{
    int Counter;

    ndInitSystem ();
    flite_init();

    v = register_cmu_us_kal ();
    flite_text_to_speech("Hello, Joseph",v,"ms0:/HelloJoseph.WAV");
}
```

This program will create a file called HelloJoseph.WAV in the root folder of the memory stick. This file will contain the synthesized voice that we wanted.

Chapter 27

Network support

Nanodesktop also provides an API for network support. The functions that allow the use of wireless connections are available only under KSU or CFW mode.

Functions for connection control

These functions are used to enable/disable a network connection, or to interrogate PSP about the state of the hardware. Nanodesktop internally uses the routines of PSP firmware, so the user must have set the correct settings in the connection profiles of the dashboard *before the* executing the nd application.

At the actual stage of development, Nanodesktop supports only the *infrastructure mode,* and not the *ad-hoc mode.* This limitation will be removed in future versions of nd.

In order to control a network connection configured in the firmware, you must define a struct of type *ndNetworkObject_Type*.

```
struct ndNetworkObject_Type NetObj;
```

NetObj will store all informations about a given network connection.

This is the struct in detail. You can have access to its fields, if you want some information about your active connection:

```
struct ndNetworkObject_PSP_Type
{
        int  NrConfigInFirmw;
        char WndDialogBox0_IsOpen;
        char WndDialogBox0_Handle;
        char WndDialogBox1_IsOpen;
        char WndDialogBox1_Handle;
        char BtnHandle0, BtnHandle1;
        int  Ok_Switcher;

        char  Name [128];
        char  SSID [64];
        char  SECURE [64];
        char  WEPKEY [64];
        char  IS_STATIC_IP;

        union ndIPV4 IP;
        union ndIPV4 NETMASK;

        union ndIPV4 ROUTE;
        union ndIPV4 PRIMARY_DNS;
        union ndIPV4 SECONDARY_DNS;

        char PROXY_USER [128];
        char PROXY_PASS [128];

        unsigned char USE_PROXY;

        union ndIPV4 PROXY_SERVER;
        unsigned short PROXY_PORT;
};

struct ndNetworkObject_Type
{
        struct ndNetworkObject_PSP_Type Psp;

        // Common data
        unsigned char IPIsKnown;
        unsigned char IPV4str [32];
        union ndIPV4  IPV4;
};
```

If you want to manage two or more connections at the same time, you have to define two or more different structures: ndNetworkObject_Type. This isn't allowed on PSP hardware, but it would be theoretically possible under other platforms (remember that nd hides the differences between the platforms in a users program).

ndHAL_WLAN_OpenPSPAccessPoint

If you want to enable a network connection, you can use this function:

```
 int ndHAL_WLAN_OpenPSPAccessPoint (char NrAccessPoint, struct ndNetworkObject_Type *NetObj, ndint64
Options, int TimeOut)
```

When you use OpenPSPAccessPoint, the PSP tries to connect to the nearest wi-fi access point and the wi-fi led on PSP is activated.

NrAccessPoint is the number of the network profile in PSP firmware. The first profile is associated with the number 1.

The pointer to struct **NetObj,* is the address of a struct of ndNetworkObject type, that will store the informations that are collected by the routine.

Option is a 64-bit parameter that contains some attributes for the connection. The content of the parameter is bit-mapped. You can set the features of the connection using some symbols that are or-ed together or using 64-bit keys. These are the symbols that are recognized:

ND_SILENT_AP_ACCESS Start the process of connection in silent-mode

TimeOut is the number of seconds that the system must wait during a wi-fi connection before declaring that the connection is down. If you specify the value 0 for this option, you'll disable the timeout counter.

The routine returns 0 if the connection is established without troubles, otherwise it returns a negative error code (see ndHAL_Network.c for further details about error codes). In case of an error in a network connection, two service codes are stored in the system variable (int): *ndNetInit_Failure_MainErrCode* and *ndNetInit_Failure_SecErrCode*.

Example:

```
#include <nanodesktop.h>

int ndMain ()
{
    struct ndNetworkObject_Type NetObj;
    int ErrRep;

    ndInitSystem ();

    ErrRep=ndHAL_WLAN_OpenPSPAccessPoint (1, &NetObj, 0, 10);

    if (!ErrRep)                          // Connessione riuscita
    {
        printf ("I am connected to network \n");
    }
    else                                  // Connessione fallita
    {
        printf ("Impossible to connect \n");
    }
}
```

Second example:

```
#include <nanodesktop.h>

int ndMain ()
{
    struct ndNetworkObject_Type NetObj;
    int ErrRep;

    ndInitSystem ();

    ErrRep=ndHAL_WLAN_OpenPSPAccessPoint (1, &NetObj, ND_SILENT_AP_ACCESS, 10);

    if (!ErrRep)                          // Connessione riuscita
    {
        printf ("I am connected to network \n");
    }
    else                                  // Connessione fallita
    {
        printf ("Impossible to connect \n");
    }
}
```

This last program executes the entire process of connection in silent mode...

ndHAL_WLAN_ClosePSPAccessPoint

This routine can close an already established connection to an access point.

```
int ndHAL_WLAN_ClosePSPAccessPoint (char NrAccessPoint)
```

It returns 0 if there are no errors

ndHAL_WLAN_PSPLanSwitchIsOpen

It returns 1 if the LAN switch of the PSP is open (wi-fi adaptor is on) and 0 if it isn't.

```
int ndHAL_WLAN_PSPLanSwitchIsOpen ()
```

ndHAL_WLAN_GetAccessPointInfo

This routine retrieves some informations about the access point identified by *NrAccessPoint,* and stores it into the fields of the struct NetObj.

```
int ndHAL_WLAN_GetAccessPointInfo (int NrAccessPoint, struct ndNetworkObject_Type *NetObj)
```

Example:

```
#include <nanodesktop.h>

int ndMain ()
{
    struct ndNetworkObject_Type NetObj;
    int ErrRep;

    ndInitSystem ();

    ErrRep=ndHAL_WLAN_GetAccessPointInfo (int NrAccessPoint, struct ndNetworkObject_Type *NetObj)(1,
&NetObj, 0, 10);

    if (!ErrRep)                         // Connessione riuscita
    {
        ndHAL_WLAN_GetAccessPointInfo (1, &NetObj);

        printf ("The address of the connection is %s \n", NetObj.IPV4str);

        ndDelay (10);
        ndHAL_WLAN_GetAccessPointInfo (1, &NetObj);

        return 0;
    }
    else                                 // Connessione fallita
    {
        printf ("Impossible to connect \n");
    }
}
```

Standard network functions

Nanodesktop integrates a wrapper that translates the standard network functions of a program into the corresponding functions of the platform where the program runs.

Because of this, you can create a simple program that uses *receive, send, listen* and so on, without knowing the specific PSP network routines.

See this program: it is based on a standard source for a BSD client that you can find easily on internet.

```
#include <nanodesktop.h>

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>


#define MAX_NUMBER_OF_BODIES 10
#define FIELD_LENGTH 50

struct AgentMessage {
  char sender[FIELD_LENGTH];
  char receiver[FIELD_LENGTH];
  char subject[FIELD_LENGTH * 2];
  int  nBody;
  char body[MAX_NUMBER_OF_BODIES][FIELD_LENGTH];
};


int ndMain ()
{
  struct sockaddr_in agent_addr;
  struct hostent *hp;
  int agent_id;
  struct AgentMessage msg;
  SOCKET s;

  char sender_ip[] = "localhost";
  char dest_ip[] = "www.psp-ita.com";
  int port = 80;

  struct ndNetworkObject_Type NetObj;
  int ErrRep;

  ndInitSystem ();

  ErrRep=ndHAL_WLAN_OpenPSPAccessPoint (1, &NetObj, 0, 10);
  ndHAL_NET_EnableNetworkMonitor ();


  if (!ErrRep)
  {
          // make message
          memset ((char*)(&msg), 0, sizeof(struct AgentMessage));
          strcpy(msg.sender, sender_ip);
          strcpy(msg.receiver, dest_ip);
          strcpy(msg.subject, "READY");

          if ((agent_id = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
            printf("Error: cannot create socket!\n");
            exit(1);
          }

          if ((hp = gethostbyname(dest_ip)) == NULL)
          {
            printf("Error: cannot get the host information about ip %s!\n",dest_ip);
            exit(1);
          }

          printf("Sending message to server with ip %s, port %d\n", hp->h_name, port);


          memset(&agent_addr, 0, sizeof(agent_addr));

          agent_addr.sin_family = AF_INET;
          agent_addr.sin_port   = htons (port);
          memcpy ((char*)&agent_addr.sin_addr, (char*)hp->h_addr, hp->h_length);

          if ( (s = socket (PF_INET, SOCK_STREAM, 0)) < 0)
          {
             printf ("Simple talk socket \n");

             ndDelay (3.0);
             exit (1);
          }
```

```
            if ( connect (s, &agent_addr, sizeof(agent_addr)) < 0 )
            {
              printf("Error: cannot connect!\n");
              closesocket (s);

              ndDelay (3.0);
              exit(1);
            }

            if (send (s, (void*)&msg, sizeof(struct AgentMessage), 0) < 0)
            {
              printf("Error: cannot send message to server\n");

              ndDelay (3.0);
              exit(1);
            }

            close(agent_id);
            printf("message sent\n");

   }
   else
   {
       printf ("Impossible to connect to the network \n");

       ndDelay (3.0);
       exit (1);
   }
}
```

Only parts in red are specific for Nanodesktop. This homebrew is able to connect to a server (in our example, www.psp-ita.com port 80) and to send a message.

Recompile it in KSU or CFW mode and it will work.

Be careful: the previous example will work only if you have a wireless connection that supports NAT functions: it won't work on networks that use a simple HTTP proxy.

Standard network functions

Nanodesktop integrates a simple network monitor that allows you the control about network operations. To enable/disable that, you can use these functions:

```
extern void ndHAL_NET_EnableNetworkMonitor (void);
extern void ndHAL_NET_DisableNetworkMonitor (void);
```

Timeout

When you execute a *connect* operation, the system waits for some time before signalling the timeout to the caller program. You can change the default time using the function:

```
ndHAL_NET_SetupStdNetworkTimeOutValue (int Limit);
```

Chapter 28

VFPU and EMI

Sony PSP supports a fast coprocessor called VFPU (Vertex Floating Point Unit). This mathematical processor is able to speed up all mathematical operations executed by the PSP. The only limitation is that VFPU doesn't support *double* type (only 32 bit float).

At first, Nanodesktop wasn't released as the developers were constrained to use an *inline assembler* to program the VFPU. In fact, the psp-gcc compiler simply ignores the VFPU. A second problem is that PSPE doesn't emulate VFPU behaviour, so if you create a program that wants to use VFPU, you could not test it under the emulator.

Nanodesktop integrates a series of technologies that can handle the situation better.

XFPU

XFPU is a macro wrapper that hides the presence of VPFU in the Nanodesktop program another mathematical coprocessor. When you use the XFPU routines, the system checks internally if a coprocessor is present in that platform, and, if it isn't, it emulates via software (using the normal libm), the required operation.

This is the list of the XFPU mathematical operation:

```
ndHAL_XFPU_sinf
ndHAL_XFPU_cosf
ndHAL_XFPU_tanf
ndHAL_XFPU_asinf
ndHAL_XFPU_acosf
ndHAL_XFPU_atanf
ndHAL_XFPU_atan2f
ndHAL_XFPU_sinhf
ndHAL_XFPU_coshf
ndHAL_XFPU_tanhf
ndHAL_XFPU_sincos
ndHAL_XFPU_expf
ndHAL_XFPU_logf
ndHAL_XFPU_fabsf
ndHAL_XFPU_sqrtf
ndHAL_XFPU_powf
ndHAL_XFPU_fmodf
ndHAL_XFPU_fminf
ndHAL_XFPU_fmaxf
```

The prototypes are identical to the corresponding functions in **libm** library.

NanoM integrates internally the VFPULib by MrMrIce, so the wrapper (under PSP), simply translated the ndHAL_XFPU routines in VFPU routines. Under PSPE, instead, the wrapper translates the same calls to the normal calls of libm.

EMI

EMI (Enhanced Mathematical Interface) is the name of a component, integrated in *NanoM,* that allows to the programmer operations similar to SIMD (Single Istruction, Multiple Data) through VFPU.

The API is written totally in Allegrex assembler, and it allows the creation of complex programs without needing to use direct assembly (i.e using simplified C calls).

Under PSPE, the system uses automatically EMIEMU, that emulates the behaviour of EMI via software.

If you use ndHAL_XFPU prefix, the system will cover automatically the differences between EMI and EMIEMU, so the same code will work either on PSPE or on PSP.

ndHAL_XFPU_Load16FloatsToMatrix

*void ndHAL_XFPU_Load16FloatsToMatrix (int NrMatrix, float *Data)*

This function loads a vector of 16 floats (addressed by *Data) in a VFPU matrix (the value that are allowed are between 0 and 5; the matrix nr. 6 is reserved for internal uses).

ndHAL_XFPU_Load16IntsToMatrix

*void ndHAL_XFPU_Load16IntsToMatrix (int NrMatrix, int *Data)*

This function loads a vector of 16 ints (addressed by *Data) in a VFPU matrix (the value that are allowed are between 0 and 5; the matrix nr. 6 is reserved for internal uses).

ndHAL_XFPU_Load16UCharsToMatrix

*void ndHAL_XFPU_Load16UCharsToMatrix (int NrMatrix, unsigned char   *Data)*

This function loads a vector of 16 unsigned chars (addressed by *Data) in a VFPU matrix (the value that are allowed are between 0 and 5; the matrix nr. 6 is reserved for internal uses).

ndHAL_XFPU_Store16FloatsFromMatrix

*void ndHAL_XFPU_Store16FloatsFromMatrix (int NrMatrix, float *Data)*

This function transfers 16 floats from the VFPU nr. *NrMatrix* (0-5), to a normal C vector of float, addressed by *Data.

ndHAL_XFPU_Store16IntsFromMatrix

*void ndHAL_XFPU_Store16IntsFromMatrix (int NrMatrix, int *Data)*

This function transfers 16 integers from the VFPU nr. *NrMatrix* (0-5), to a normal C vector of integers, addressed by *Data.

ndHAL_XFPU_Store16UCharsFromMatrix

*void ndHAL_XFPU_Store16UCharsFromMatrix (int NrMatrix, int *Data)*

This function transfers 16 unsigned chars from the VFPU nr. *NrMatrix* (0-5), to a normal C vector of unsigned chars, addressed by *Data.

ndHAL_XFPU_NullMatrix

void ndHAL_XFPU_NullMatrix (int NrMatrix)

It puts to zero all elements of a VFPU matrix.

ndHAL_XFPU_MatrixSum

void ndHAL_XFPU_MatrixSum (int NrMatrix0, int NrMatrix1, int NrMatrixDest)

It sums the content of two VFPU matrix. The result is stored in the VFPU matrix MatrixDest.

ndHAL_XFPU_MatrixSub

void ndHAL_XFPU_MatrixSub (int NrMatrix0, int NrMatrix1, int NrMatrixDest)

It executes the matricial operation of subtraction: (MatrixDest = Matrix0-Matrix1);

ndHAL_XFPU_MatrixTrvMul

void ndHAL_XFPU_MatrixTrvMul (int NrMatrix0, int NrMatrix1, int NrMatrixDest)

It executes the trivial multiply of two matrix. It is done using the formula:

MatrixDest [i;j] = matrix0 [i;j] * matrix1 [i;j]

Please note that this operation is different than the *standard rows for columns matricial multiplying.*

ndHAL_XFPU_MatrixTrvDiv

void ndHAL_XFPU_MatrixTrvDiv (int NrMatrix0, int NrMatrix1, int NrMatrixDest)

It executes the trivial division of two matrix. It is done using the formula:

MatrixDest [i;j] = matrix0 [i;j] / matrix1 [i;j]

Please note that this operation is different than the *standard matricial division R = (A^-1) * A*

ndHAL_XFPU_MatrixCpy

void ndHAL_XFPU_MatrixCpy (int NrMatrixSrc, int NrMatrixDest)

Copy the content of MatrixSrc (0-5) into MatrixDest

ndHAL_XFPU_SequentialAdder

*void ndHAL_XFPU_SequentialAdder (int NrMatrix0, int NrMatrix1, float *CarryIn, float *CarryOut)*

This is a particular sequential adder. Let's suppose that Matrix0 and Matrix1 could be seen as vectors of 16 elements (pointed through an index between 0 and 15).

The first operation done by the routine is the following:

```
Matrix0 [0] = *CarryIn + Matrix1 [0];
```

I.e the content of the first element of the matrix0 is summed with the 32 float pointed by *CarryIn*.

For the following 15 elements of Matrix0, the system does a sum using this formula:

```
Matrix0 [n] = Matrix0 [n-1] + Matrix1 [n];
```

So, in Matrix0 you will be able to see the partial sums obtained by the adder. The final sum, is stored in a 32 bit floating point C variable, pointed by *CarryOut*.

Chapter 29
# ndOpenCV

Dario Ignazio
Filippo Battaglia

Intel OpenCV is a library for scientifical targets, used for applications of images analysis and artificial vision. This library provides hundreds of functionalities, as object tracking, face recognition, face detection, image filtering, image transformations, color processing, etc.

OpenCV is a registered trademark. All right reserved to Intel Corporation.

This library has been initially designed to run **only** on x86 platform (under Windows or Linux). The library is used by thousands of researchers in universities, centers of study etc. Applications for medical tasks, robotic automation, human interaction exist and they are based on this library.

One of the most important feature of Nanodesktop technology consists in the fact that it has made possible the creation of a version of OpenCV that can run on *embedded processors,* as the one installed in PSP. This custom version of Nanodesktop is called *ndOpenCV.*

ndOpenCV is derived from the code of version 0.9.7 of x86 OpenCV. The library has been heavily modified - some sections have been totally rewritten, adapting the code to MIPS processor and using the VFPU to speed up computations.

OpenCV is composed by the following components:

– **cxcore, cv, and cvaux:** these libraries provide low level algorithms for image manipulations, thresholding, split and merge, mathematical functions, face detection and recognitions, motion tracking, etc.

– **highgui:** in x86 version of the library, it provide routines for showing/loading/saving OpenCV images, for accessing to webcam, for the interaction with the GUI of the system. This component is strongly dependent from the operating system that is used. So, under Nanodesktop, the original x86 highgui library has been replaced by a new library, called **ndHighGUI.**

The routines exported by nd versions of cxcore, cv and cvaux are identical to these exported by the original x86 versions of the same libraries.

You can read something about these routines here:

http://opencvlibrary.sourceforge.net/CxCore
http://opencvlibrary.sourceforge.net/CvReference
http://opencvlibrary.sourceforge.net/CvAux

The routines exported by ndHighGUI, instead, are very similar to the original x86 version, but with *some differences,* that are needed because x86 architecture is different than PSP/nd architecture.

See in the following pages the behaviour of any routine in ndHighGUI.

The source code of ndOpenCV is in the folder:

<ndenv folder>\PSP\ndOpenCV

while some examples are present in:

<ndenv folder>\PSP\ndOpenCV\samples

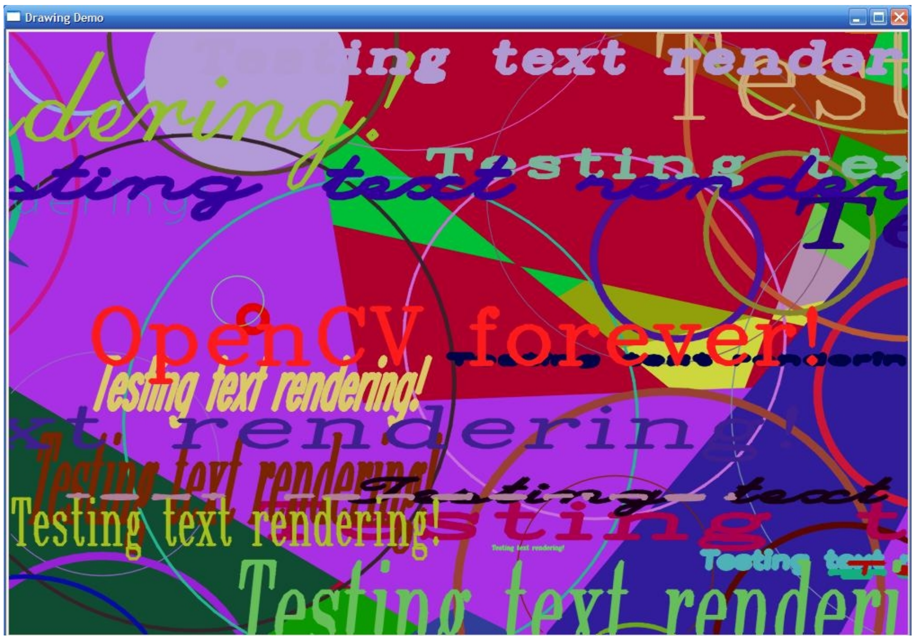
We haven't here space to explain the fundamentals of OpenCV programming. We remand you to *Intel x86 manual* for further details. In this book, we'll only see some samples written by Intel programmers for x86 OpenCV, and how the same code can be ported on Nanodesktop environment.

Example 1

This program has been written by Intel to show the graphical abilities of x86 OpenCV.

```
#ifdef _CH_
#pragma package <opencv>
#endif

#ifndef _EiC
#include "cv.h"
#include "highgui.h"
#include <stdlib.h>
#include <stdio.h>
#endif

#define NUMBER 100
#define DELAY 5
char wndname[] = "Drawing Demo";

CvScalar random_color(CvRNG* rng)
{
    int icolor = cvRandInt(rng);
    return CV_RGB(icolor&255, (icolor>>8)&255, (icolor>>16)&255);
}

int main( int argc, char** argv )
{
    int line_type = CV_AA; // change it to 8 to see non-antialiased graphics
    int i;
    CvPoint pt1,pt2;
    double angle;
    CvSize sz;
    CvPoint  ptt[6];
    CvPoint* pt[2];
    int  arr[2];
    CvFont font;
    CvRNG rng;
    int width = 1000, height = 700;
    int width3 = width*3, height3 = height*3;
    CvSize text_size;
    int ymin = 0;
    // Load the source image
    IplImage* image = cvCreateImage( cvSize(width,height), 8, 3 );
    IplImage* image2;

    // Create a window
    cvNamedWindow(wndname, 1 );
    cvZero( image );
    cvShowImage(wndname,image);

    rng = cvRNG((unsigned)-1);
    pt[0] = &(ptt[0]);
    pt[1] = &(ptt[3]);

    arr[0] = 3;
    arr[1] = 3;

    for (i = 0; i< NUMBER; i++)
    {
        pt1.x=cvRandInt(&rng) % width3 - width;
        pt1.y=cvRandInt(&rng) % height3 - height;
        pt2.x=cvRandInt(&rng) % width3 - width;
        pt2.y=cvRandInt(&rng) % height3 - height;

        cvLine( image, pt1, pt2, random_color(&rng), cvRandInt(&rng)%10, line_type, 0 );
        cvShowImage(wndname,image);
        cvWaitKey(DELAY);
    }

    for (i = 0; i< NUMBER; i++)
    {
        pt1.x=cvRandInt(&rng) % width3 - width;
        pt1.y=cvRandInt(&rng) % height3 - height;
        pt2.x=cvRandInt(&rng) % width3 - width;
        pt2.y=cvRandInt(&rng) % height3 - height;

        cvRectangle( image,pt1, pt2, random_color(&rng), cvRandInt(&rng)%10-1, line_type, 0 );
        cvShowImage(wndname,image);
        cvWaitKey(DELAY);
    }

    for (i = 0; i< NUMBER; i++)
    {
        pt1.x=cvRandInt(&rng) % width3 - width;
        pt1.y=cvRandInt(&rng) % height3 - height;
        sz.width =cvRandInt(&rng)%200;
        sz.height=cvRandInt(&rng)%200;
        angle = (cvRandInt(&rng)%1000)*0.180;

        cvEllipse( image, pt1, sz, angle, angle - 100, angle + 200,
                   random_color(&rng), cvRandInt(&rng)%10-1, line_type, 0 );
        cvShowImage(wndname,image);
        cvWaitKey(DELAY);
    }

    for (i = 0; i< NUMBER; i++)
    {
        pt[0][0].x=cvRandInt(&rng) % width3 - width;
        pt[0][0].y=cvRandInt(&rng) % height3 - height;
        pt[0][1].x=cvRandInt(&rng) % width3 - width;
        pt[0][1].y=cvRandInt(&rng) % height3 - height;
        pt[0][2].x=cvRandInt(&rng) % width3 - width;
        pt[0][2].y=cvRandInt(&rng) % height3 - height;
        pt[1][0].x=cvRandInt(&rng) % width3 - width;
        pt[1][0].y=cvRandInt(&rng) % height3 - height;
        pt[1][1].x=cvRandInt(&rng) % width3 - width;
        pt[1][1].y=cvRandInt(&rng) % height3 - height;
        pt[1][2].x=cvRandInt(&rng) % width3 - width;
        pt[1][2].y=cvRandInt(&rng) % height3 - height;

        cvPolyLine( image, pt, arr, 2, 1, random_color(&rng), cvRandInt(&rng)%10, line_type, 0 );
        cvShowImage(wndname,image);
        cvWaitKey(DELAY);
    }

    for (i = 0; i< NUMBER; i++)
    {
        pt[0][0].x=cvRandInt(&rng) % width3 - width;
        pt[0][0].y=cvRandInt(&rng) % height3 - height;
        pt[0][1].x=cvRandInt(&rng) % width3 - width;
        pt[0][1].y=cvRandInt(&rng) % height3 - height;
        pt[0][2].x=cvRandInt(&rng) % width3 - width;
        pt[0][2].y=cvRandInt(&rng) % height3 - height;
        pt[1][0].x=cvRandInt(&rng) % width3 - width;
        pt[1][0].y=cvRandInt(&rng) % height3 - height;
        pt[1][1].x=cvRandInt(&rng) % width3 - width;
        pt[1][1].y=cvRandInt(&rng) % height3 - height;
        pt[1][2].x=cvRandInt(&rng) % width3 - width;
        pt[1][2].y=cvRandInt(&rng) % height3 - height;

        cvFillPoly( image, pt, arr, 2, random_color(&rng), line_type, 0 );
        cvShowImage(wndname,image);
        cvWaitKey(DELAY);
    }

    for (i = 0; i< NUMBER; i++)
    {
        pt1.x=cvRandInt(&rng) % width3 - width;
        pt1.y=cvRandInt(&rng) % height3 - height;

        cvCircle( image, pt1, cvRandInt(&rng)%300, random_color(&rng),
                  cvRandInt(&rng)%10-1, line_type, 0 );
        cvShowImage(wndname,image);
        cvWaitKey(DELAY);
    }

    for (i = 1; i< NUMBER; i++)
    {
        pt1.x=cvRandInt(&rng) % width3 - width;
        pt1.y=cvRandInt(&rng) % height3 - height;

        cvInitFont( &font, cvRandInt(&rng) % 8,
                    (cvRandInt(&rng)%100)*0.05+0.1, (cvRandInt(&rng)%100)*0.05+0.1,
                    (cvRandInt(&rng)%5)*0.1, cvRound(cvRandInt(&rng)%10), line_type );

        cvPutText( image, "Testing text rendering!", pt1, &font, random_color(&rng));
        cvShowImage(wndname,image);
        cvWaitKey(DELAY);
    }

    cvInitFont( &font, CV_FONT_HERSHEY_COMPLEX, 3, 3, 0.0, 5, line_type );
    cvGetTextSize( "OpenCV forever!", &font, &text_size, &ymin );

    pt1.x = (width - text_size.width)/2;
    pt1.y = (height + text_size.height)/2;
    image2 = cvCloneImage(image);

    for( i = 0; i < 255; i++ )
    {
        cvSubS( image2, cvScalarAll(i), image, 0 );
        cvPutText( image, "OpenCV forever!", pt1, &font, CV_RGB(255,i,i));
        cvShowImage(wndname,image);
        cvWaitKey(DELAY);
    }

    // Wait for a key stroke; the same function arranges events processing
    cvWaitKey(0);
    cvReleaseImage(&image);
    cvReleaseImage(&image2);
    cvDestroyWindow(wndname);

    return 0;
}

#ifdef _EiC
main(1,"drawing.c");
#endif
```

Here the result in Windows:



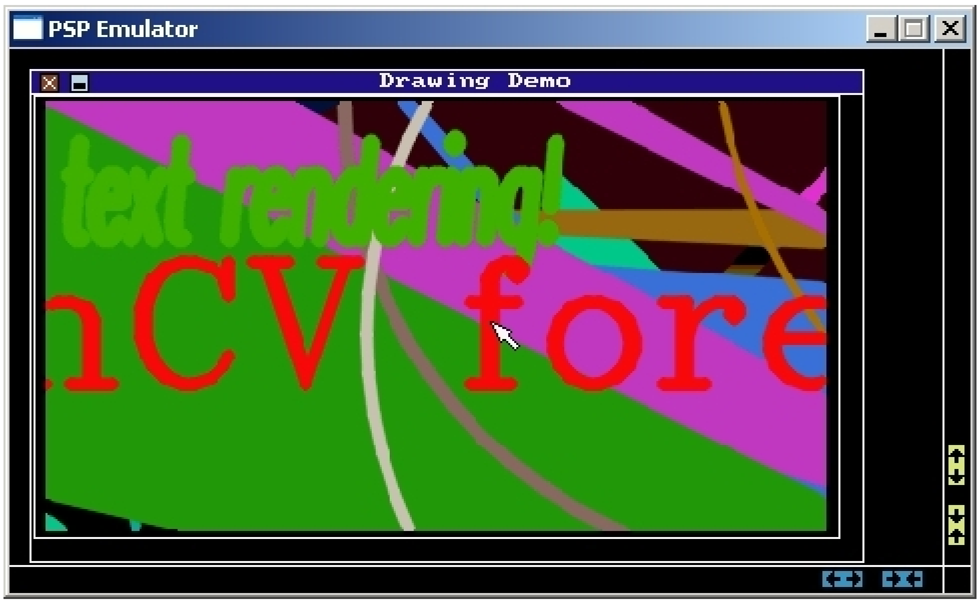
Ok, now see the equivalent example for nd:

```
#include <ndhighgui.h>

extern "C"
{


#define NUMBER 10
#define DELAY 5
char wndname[] = "Drawing Demo";

CvScalar random_color(CvRNG* rng)
{
    int icolor = cvRandInt(rng);
    return CV_RGB(icolor&255, (icolor>>8)&255, (icolor>>16)&255);
}

int ndMain(void)
{
    cvInitSystem (0, 0);

    int line_type = CV_AA; // change it to 8 to see non-antialiased graphics
    int i;
```

```
CvPoint pt1,pt2;
double angle;
CvSize sz;
CvPoint  ptt[6];
CvPoint* pt[2];
int  arr[2];
CvFont font;
CvRNG rng;
int width = 400, height = 220;
int width3 = width*3, height3 = height*3;
CvSize text_size;
int ymin = 0;
// Load the source image
IplImage* image = cvCreateImage( cvSize(width,height), 8, 3 );
IplImage* image2;

// Create a window
cvSetNextColorWnd (COLOR_WHITE, COLOR_BLUE, COLOR_BLACK, COLOR_WHITE);
cvNamedWindow(wndname, 1);
cvZero( image );
cvSetNextColorWnd (COLOR_WHITE, COLOR_BLUE, COLOR_BLACK, COLOR_WHITE);
cvShowImage(wndname,image);

rng = cvRNG((unsigned)-1);
pt[0] = &(ptt[0]);
pt[1] = &(ptt[3]);

arr[0] = 3;
arr[1] = 3;

for (i = 0; i< NUMBER; i++)
{
    pt1.x=cvRandInt(&rng) % width3 - width;
    pt1.y=cvRandInt(&rng) % height3 - height;
    pt2.x=cvRandInt(&rng) % width3 - width;
    pt2.y=cvRandInt(&rng) % height3 - height;

    cvLine( image, pt1, pt2, random_color(&rng), cvRandInt(&rng)%10, line_type, 0 );
    cvShowImage(wndname,image); //manca cvWaitkey
}

for (i = 0; i< NUMBER; i++)
{
    pt1.x=cvRandInt(&rng) % width3 - width;
    pt1.y=cvRandInt(&rng) % height3 - height;
    pt2.x=cvRandInt(&rng) % width3 - width;
    pt2.y=cvRandInt(&rng) % height3 - height;

    cvRectangle( image,pt1, pt2, random_color(&rng), cvRandInt(&rng)%10-1, line_type, 0 );
    cvShowImage(wndname,image);
}



for (i = 0; i< NUMBER; i++)
{
    pt1.x=cvRandInt(&rng) % width3 - width;
    pt1.y=cvRandInt(&rng) % height3 - height;
    sz.width =cvRandInt(&rng)%200;
    sz.height=cvRandInt(&rng)%200;
    angle = (cvRandInt(&rng)%1000)*0.180;

    cvEllipse( image, pt1, sz, angle, angle - 100, angle + 200,
               random_color(&rng), cvRandInt(&rng)%10-1, line_type, 0 );

    cvShowImage(wndname,image);
}



for (i = 0; i< NUMBER; i++)
{
    pt[0][0].x=cvRandInt(&rng) % width3 - width;
    pt[0][0].y=cvRandInt(&rng) % height3 - height;
    pt[0][1].x=cvRandInt(&rng) % width3 - width;
    pt[0][1].y=cvRandInt(&rng) % height3 - height;
    pt[0][2].x=cvRandInt(&rng) % width3 - width;
    pt[0][2].y=cvRandInt(&rng) % height3 - height;
    pt[1][0].x=cvRandInt(&rng) % width3 - width;
    pt[1][0].y=cvRandInt(&rng) % height3 - height;
    pt[1][1].x=cvRandInt(&rng) % width3 - width;
    pt[1][1].y=cvRandInt(&rng) % height3 - height;
    pt[1][2].x=cvRandInt(&rng) % width3 - width;
    pt[1][2].y=cvRandInt(&rng) % height3 - height;
```

```
        cvPolyLine( image, pt, arr, 2, 1, random_color(&rng), cvRandInt(&rng)%10, line_type, 0 );
        cvShowImage(wndname,image);
    }

    for (i = 0; i< NUMBER; i++)
    {
        pt[0][0].x=cvRandInt(&rng) % width3 - width;
        pt[0][0].y=cvRandInt(&rng) % height3 - height;
        pt[0][1].x=cvRandInt(&rng) % width3 - width;
        pt[0][1].y=cvRandInt(&rng) % height3 - height;
        pt[0][2].x=cvRandInt(&rng) % width3 - width;
        pt[0][2].y=cvRandInt(&rng) % height3 - height;
        pt[1][0].x=cvRandInt(&rng) % width3 - width;
        pt[1][0].y=cvRandInt(&rng) % height3 - height;
        pt[1][1].x=cvRandInt(&rng) % width3 - width;
        pt[1][1].y=cvRandInt(&rng) % height3 - height;
        pt[1][2].x=cvRandInt(&rng) % width3 - width;
        pt[1][2].y=cvRandInt(&rng) % height3 - height;

        cvFillPoly( image, pt, arr, 2, random_color(&rng), line_type, 0 );
        cvShowImage(wndname,image);
    }



    for (i = 0; i< NUMBER; i++)
    {
        pt1.x=cvRandInt(&rng) % width3 - width;
        pt1.y=cvRandInt(&rng) % height3 - height;

        cvCircle( image, pt1, cvRandInt(&rng)%300, random_color(&rng),
                  cvRandInt(&rng)%10-1, line_type, 0 );

        cvShowImage(wndname,image);
    }



    for (i = 1; i< NUMBER; i++)
    {
        pt1.x=cvRandInt(&rng) % width3 - width;
        pt1.y=cvRandInt(&rng) % height3 - height;

        cvInitFont( &font, cvRandInt(&rng) % 8,
                    (cvRandInt(&rng)%100)*0.05+0.1, (cvRandInt(&rng)%100)*0.05+0.1,
                    (cvRandInt(&rng)%5)*0.1, cvRound(cvRandInt(&rng)%10), line_type );

        cvPutText( image, "Testing text rendering!", pt1, &font, random_color(&rng));
        cvShowImage(wndname,image);
    }



    cvInitFont( &font, CV_FONT_HERSHEY_COMPLEX, 3, 3, 0.0, 5, line_type );

    cvGetTextSize( "OpenCV forever!", &font, &text_size, &ymin );

    pt1.x = (width - text_size.width)/2;
    pt1.y = (height + text_size.height)/2;
    image2 = cvCloneImage(image);

    for( i = 0; i < 10; i++ )
    {
        cvSubS( image2, cvScalarAll(i), image, 0 );
        cvPutText( image, "OpenCV forever!", pt1, &font, CV_RGB(255,i,i));

        cvShowImage(wndname,image);
    }

    cvWaitKey (0);

    // Wait for a key stroke; the same function arranges events processing
    cvReleaseImage(&image);
    cvReleaseImage(&image2);
    cvDestroyWindow(wndname);

    return 0;

}

}
```

The result is shown in the following page (PSPE).



Now, we have to see the differences. We have put in evidence them in *red.*

1) x86 example uses

```
#include "cv.h"
#include "highgui.h"
#include <stdlib.h>
#include <stdio.h>
```

while nd example uses

```
#include <ndhighgui.h>
```

Under nd you must use **#include <ndhighgui.h>** instead of highgui.h.

Note: the header file ndhighgui.h includes the files cxcore.h, cv.h, nanodesktop.h (this last includes some other headers as stdlib.h and stdio.h). So, the second example doesn't need to include explicitally stdlib.h and stdio.h (in any case, you can also include them explicitally: it isn't void, it is only unuseful).

2) x86 example begins with

```
int main( int argc, char** argv )
```

while nd example begins with

```
int ndMain(void)
{
    cvInitSystem (0, 0);

    [...]
}
```

The nd code begins with **ndMain**. Furthermore, the Opencv system must be initialized using **cvInitSystem (0, 0)**. cvInitSystem is facultative under x86 platform. **Under nd it is an obligation**.

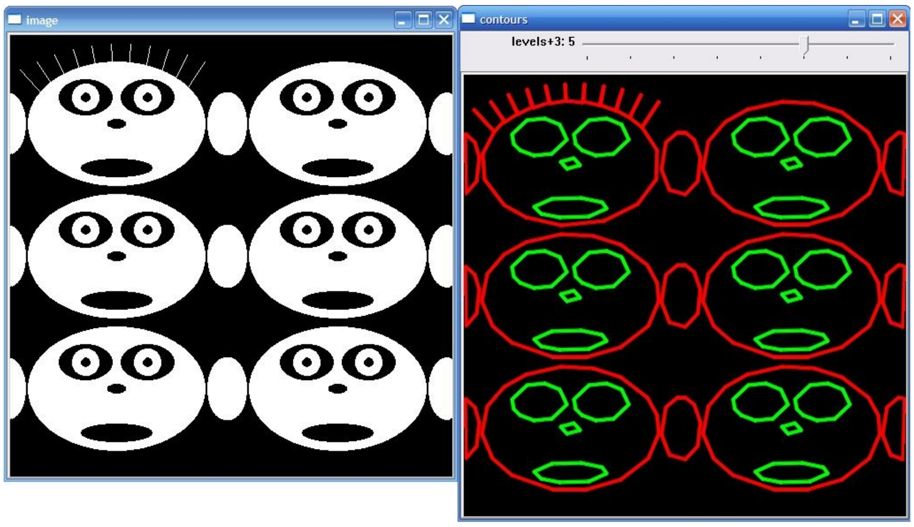
Note: cvInitSystem recalls internally ndInitSystem, so you can “don't call” the latter at the beginning of your program.

3) Note that nd version uses a new function, called *cvSetNextWndColor,* this determinates the colors of the next OpenCV window. This function doesn't exist in x86 environment, because these colors are defined by the GUI of the operating system (for example, these colors are influences by the desktop theme that you are using in KDE or Windows in that moment).

Example 2

Now, we'll see a new program in x86 version and in nd version. Here the x86 version:

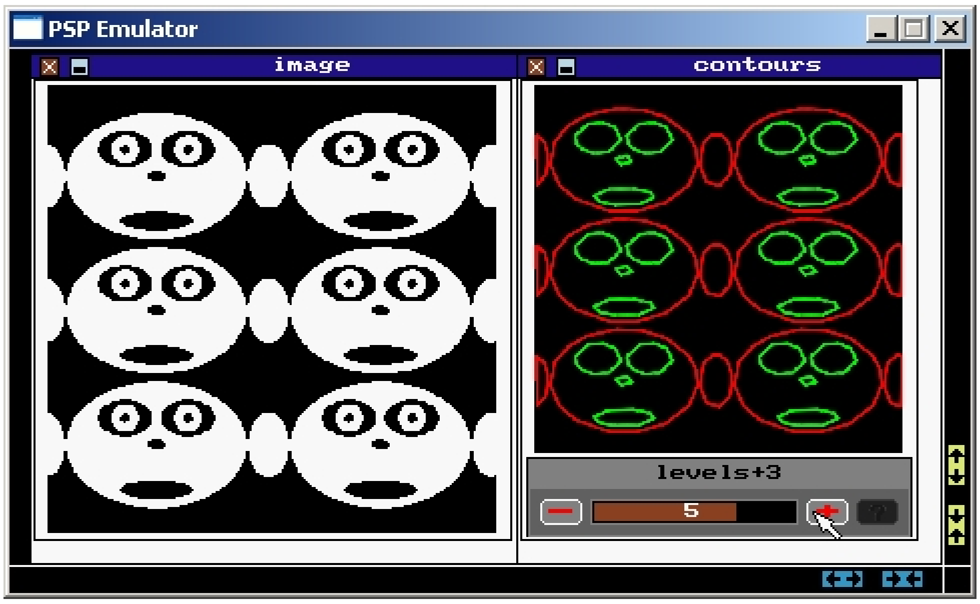
```
#ifdef _CH_
#pragma package <opencv>
#endif

#ifndef _EiC
#include "cv.h"
#include "highgui.h"
#include <math.h>
#endif

#define w 500
int levels = 3;
CvSeq* contours = 0;

void on_trackbar(int pos)
{
    IplImage* cnt_img = cvCreateImage( cvSize(w,w), 8, 3 );
    CvSeq* _contours = contours;
    int _levels = levels - 3;
    if( _levels <= 0 ) // get to the nearest face to make it look more funny
        _contours = _contours->h_next->h_next->h_next;
    cvZero( cnt_img );
    cvDrawContours( cnt_img, _contours, CV_RGB(255,0,0), CV_RGB(0,255,0), _levels, 3, CV_AA,
cvPoint(0,0) );
    cvShowImage( "contours", cnt_img );
    cvReleaseImage( &cnt_img );
}

int main( int argc, char** argv )
{
    int i, j;
    CvMemStorage* storage = cvCreateMemStorage(0);
    IplImage* img = cvCreateImage( cvSize(w,w), 8, 1 );

    cvZero( img );

    for( i=0; i < 6; i++ )
    {
        int dx = (i%2)*250 - 30;
        int dy = (i/2)*150;
        CvScalar white = cvRealScalar(255);
        CvScalar black = cvRealScalar(0);

        if( i == 0 )
        {
            for( j = 0; j <= 10; j++ )
            {
                double angle = (j+5)*CV_PI/21;
                cvLine(img, cvPoint(cvRound(dx+100+j*10-80*cos(angle)),
                    cvRound(dy+100-90*sin(angle))),
                    cvPoint(cvRound(dx+100+j*10-30*cos(angle)),
                    cvRound(dy+100-30*sin(angle))), white, 1, 8, 0);
            }
        }

        cvEllipse( img, cvPoint(dx+150, dy+100), cvSize(100,70), 0, 0, 360, white, -1, 8, 0 );
        cvEllipse( img, cvPoint(dx+115, dy+70), cvSize(30,20), 0, 0, 360, black, -1, 8, 0 );
        cvEllipse( img, cvPoint(dx+185, dy+70), cvSize(30,20), 0, 0, 360, black, -1, 8, 0 );
        cvEllipse( img, cvPoint(dx+115, dy+70), cvSize(15,15), 0, 0, 360, white, -1, 8, 0 );
        cvEllipse( img, cvPoint(dx+185, dy+70), cvSize(15,15), 0, 0, 360, white, -1, 8, 0 );
        cvEllipse( img, cvPoint(dx+115, dy+70), cvSize(5,5), 0, 0, 360, black, -1, 8, 0 );
        cvEllipse( img, cvPoint(dx+185, dy+70), cvSize(5,5), 0, 0, 360, black, -1, 8, 0 );
        cvEllipse( img, cvPoint(dx+150, dy+100), cvSize(10,5), 0, 0, 360, black, -1, 8, 0 );
        cvEllipse( img, cvPoint(dx+150, dy+150), cvSize(40,10), 0, 0, 360, black, -1, 8, 0 );
        cvEllipse( img, cvPoint(dx+27, dy+100), cvSize(20,35), 0, 0, 360, white, -1, 8, 0 );
        cvEllipse( img, cvPoint(dx+273, dy+100), cvSize(20,35), 0, 0, 360, white, -1, 8, 0 );
    }

    cvNamedWindow( "image", 1 );
    cvShowImage( "image", img );

    cvFindContours( img, storage, &contours, sizeof(CvContour),
                    CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, cvPoint(0,0) );

    // comment this out if you do not want approximation
    contours = cvApproxPoly( contours, sizeof(CvContour), storage, CV_POLY_APPROX_DP, 3, 1 );

    cvNamedWindow( "contours", 1 );
    cvCreateTrackbar( "levels+3", "contours", &levels, 7, on_trackbar );

    on_trackbar(0);
    cvWaitKey(0);
    cvReleaseMemStorage( &storage );
    cvReleaseImage( &img );

    return 0;
}

#ifdef _EiC
main(1,"");
#endif
```



And here the nd version:

```
extern "C"
{


#include "cv.h"
#include "ndhighgui.h"

#define w 500
int levels = 3;
CvSeq* contours = 0;

void on_trackbar(int pos)
{
    IplImage* cnt_img = cvCreateImage( cvSize(w,w), 8, 3 );
    CvSeq* _contours = contours;
    int _levels = levels - 3;
    if( _levels <= 0 ) // get to the nearest face to make it look more funny
        _contours = _contours->h_next->h_next->h_next;
    cvZero( cnt_img );
    cvDrawContours( cnt_img, _contours, CV_RGB(255,0,0), CV_RGB(0,255,0), _levels, 3, CV_AA,
cvPoint(0,0) );
    cvShowImage( "contours", cnt_img );
    cvReleaseImage( &cnt_img );
}

int ndMain( int argc, char** argv )
{
    int i;
    CvMemStorage* storage = cvCreateMemStorage(0);
    IplImage* img = cvCreateImage( cvSize(w,w), 8, 1 );

    ndEvent_Type MyEvent;

    cvInitSystem (0, 0);

    cvZero( img );
    for( i=0; i < 6; i++ )
    {
        int dx = (i%2)*250 - 30;
        int dy = (i/2)*150;
        CvScalar white = cvRealScalar(255);
        CvScalar black = cvRealScalar(0);
        cvEllipse( img, cvPoint(dx+150, dy+100), cvSize(100,70), 0, 0, 360, white, -1, 8, 0 );
        cvEllipse( img, cvPoint(dx+115, dy+70), cvSize(30,20), 0, 0, 360, black, -1, 8, 0 );
        cvEllipse( img, cvPoint(dx+185, dy+70), cvSize(30,20), 0, 0, 360, black, -1, 8, 0 );
        cvEllipse( img, cvPoint(dx+115, dy+70), cvSize(15,15), 0, 0, 360, white, -1, 8, 0 );
        cvEllipse( img, cvPoint(dx+185, dy+70), cvSize(15,15), 0, 0, 360, white, -1, 8, 0 );
        cvEllipse( img, cvPoint(dx+115, dy+70), cvSize(5,5), 0, 0, 360, black, -1, 8, 0 );
        cvEllipse( img, cvPoint(dx+185, dy+70), cvSize(5,5), 0, 0, 360, black, -1, 8, 0 );
        cvEllipse( img, cvPoint(dx+150, dy+100), cvSize(10,5), 0, 0, 360, black, -1, 8, 0 );
        cvEllipse( img, cvPoint(dx+150, dy+150), cvSize(40,10), 0, 0, 360, black, -1, 8, 0 );
        cvEllipse( img, cvPoint(dx+27, dy+100), cvSize(20,35), 0, 0, 360, white, -1, 8, 0 );
        cvEllipse( img, cvPoint(dx+273, dy+100), cvSize(20,35), 0, 0, 360, white, -1, 8, 0 );
    }

    cvNamedWindow( "image", 1 );
    cvShowImage( "image", img );

    cvFindContours( img, storage, &contours, sizeof(CvContour),
                    CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, cvPoint(0,0) );

    // comment this out if you do not want approximation
    contours = cvApproxPoly( contours, sizeof(CvContour), storage, CV_POLY_APPROX_DP, 3, 1 );

    cvNamedWindow( "contours", 1 );
    cvCreateTrackbar( "levels+3", "contours", &levels, 7, on_trackbar );

    on_trackbar(0);

    MouseControl (&MyEvent);

    return 0;
}



}
```

The result is this:



The differences are these:

1) different #include directives (the reason is the same that we have already said)
2) the nd version includes cvInitSystem (0, 0);
3) the nd version executes a call to *MouseControl (&MyEvent)*. MyEvent is a struct previously defined. This is done in nd version, in way that the mouse pointer appears on the screen. In x86 version this isn't done because the mouse pointer is always present on desktop.

Example 3

This is the third comparison. Let's see a source that uses webcam through OpenCV layer.

```
#include "cv.h"
#include "highgui.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include <float.h>
#include <limits.h>
#include <time.h>
#include <ctype.h>

#ifdef _EiC
#define WIN32
#endif

static CvMemStorage* storage = 0;
static CvHaarClassifierCascade* cascade = 0;

void detect_and_draw( IplImage* image );

const char* cascade_name =
    "haarcascade_frontalface_alt.xml";
/*    "haarcascade_profileface.xml";*/

int main( int argc, char** argv )
{
    CvCapture* capture = 0;
    IplImage *frame, *frame_copy = 0;
    int optlen = strlen("--cascade=");
    const char* input_name;

    if( argc > 1 && strncmp( argv[1], "--cascade=", optlen ) == 0 )
    {
        cascade_name = argv[1] + optlen;
        input_name = argc > 2 ? argv[2] : 0;
    }
    else
    {
        cascade_name = "../../data/haarcascades/haarcascade_frontalface_alt2.xml";
        input_name = argc > 1 ? argv[1] : 0;
    }

    cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name, 0, 0, 0 );

    if( !cascade )
    {
        fprintf( stderr, "ERROR: Could not load classifier cascade\n" );
        fprintf( stderr,
        "Usage: facedetect --cascade=\"<cascade_path>\" [filename|camera_index]\n" );
        return -1;
    }

    storage = cvCreateMemStorage(0);

    if( !input_name || (isdigit(input_name[0]) && input_name[1] == '\0') )
        capture = cvCaptureFromCAM( !input_name ? 0 : input_name[0] - '0' );
    else
        capture = cvCaptureFromAVI( input_name );

    cvNamedWindow( "result", 1 );

    if( capture )
    {
        for(;;)
        {
            if( !cvGrabFrame( capture ))
                break;
            frame = cvRetrieveFrame( capture );
            if( !frame )
                break;
            if( !frame_copy )
                frame_copy = cvCreateImage( cvSize(frame->width,frame->height),
                                            IPL_DEPTH_8U, frame->nChannels );
            if( frame->origin == IPL_ORIGIN_TL )
                cvCopy( frame, frame_copy, 0 );
            else
                cvFlip( frame, frame_copy, 0 );

            detect_and_draw( frame_copy );

            if( cvWaitKey( 10 ) >= 0 )
                break;
        }

        cvReleaseImage( &frame_copy );
        cvReleaseCapture( &capture );
    }
    else
    {
        const char* filename = input_name ? input_name : (char*)"lena.jpg";
        IplImage* image = cvLoadImage( filename, 1 );

        if( image )
        {
            detect_and_draw( image );
            cvWaitKey(0);
            cvReleaseImage( &image );
        }
        else
        {
            /* assume it is a text file containing the
               list of the image filenames to be processed - one per line */
            FILE* f = fopen( filename, "rt" );
            if( f )
            {
                char buf[1000+1];
                while( fgets( buf, 1000, f ) )
                {
                    int len = (int)strlen(buf);
                    while( len > 0 && isspace(buf[len-1]) )
                        len--;
                    buf[len] = '\0';
                    image = cvLoadImage( buf, 1 );
                    if( image )
                    {
                        detect_and_draw( image );
                        cvWaitKey(0);
                        cvReleaseImage( &image );
                    }
                }
                fclose(f);
            }
        }

    }

    cvDestroyWindow("result");

    return 0;
}

void detect_and_draw( IplImage* img )
{
    static CvScalar colors[] =
    {
        {{0,0,255}},
        {{0,128,255}},
        {{0,255,255}},
        {{0,255,0}},
        {{255,128,0}},
        {{255,255,0}},
        {{255,0,0}},
        {{255,0,255}}
    };

    double scale = 1.3;
    IplImage* gray = cvCreateImage( cvSize(img->width,img->height), 8, 1 );
    IplImage* small_img = cvCreateImage( cvSize( cvRound (img->width/scale),
                         cvRound (img->height/scale)),
                     8, 1 );
    int i;

    cvCvtColor( img, gray, CV_BGR2GRAY );
    cvResize( gray, small_img, CV_INTER_LINEAR );
    cvEqualizeHist( small_img, small_img );
    cvClearMemStorage( storage );

    if( cascade )
    {
        double t = (double)cvGetTickCount();
        CvSeq* faces = cvHaarDetectObjects( small_img, cascade, storage,
                                            1.1, 2, 0/*CV_HAAR_DO_CANNY_PRUNING*/,
                                            cvSize(30, 30) );
        t = (double)cvGetTickCount() - t;
        printf( "detection time = %gms\n", t/((double)cvGetTickFrequency()*1000.) );
        for( i = 0; i < (faces ? faces->total : 0); i++ )
        {
            CvRect* r = (CvRect*)cvGetSeqElem( faces, i );
            CvPoint center;
            int radius;
            center.x = cvRound((r->x + r->width*0.5)*scale);
            center.y = cvRound((r->y + r->height*0.5)*scale);
            radius = cvRound((r->width + r->height)*0.25*scale);
            cvCircle( img, center, radius, colors[i%8], 3, 8, 0 );
        }
    }

    cvShowImage( "result", img );
    cvReleaseImage( &gray );
    cvReleaseImage( &small_img );
}
```

And now, let's see the equivalent version for Nanodesktop PSP.

A note: under nd, in order to use this program, you must use KSU or CFW mode to enable webcam support.

```
#include <nanodesktop.h>
#include <ndhighgui.h>

static CvMemStorage* storage = 0;
static CvHaarClassifierCascade* cascade = 0;

const char* cascade_name =
    "ms0:/DEMOPACK/CVDEMO8/haarcascade_frontalface_alt.xml";



void detect_and_draw( IplImage* img )
{
    static CvScalar colors[] =
    {
        {{0,0,255}},
        {{0,128,255}},
        {{0,255,255}},
        {{0,255,0}},
        {{255,128,0}},
        {{255,255,0}},
        {{255,0,0}},
        {{255,0,255}}
    };

    double scale = 1.3;

    IplImage* small_img = cvCreateImage( cvSize( cvRound (img->width/scale),
                         cvRound (img->height/scale)), 8, 1 );
    int i;

    cvResize( img, small_img, CV_INTER_LINEAR );
    cvEqualizeHist( small_img, small_img );
    cvClearMemStorage( storage );

    if( cascade )
    {
        //double t = (double)cvGetTickCount();
        CvSeq* faces = cvHaarDetectObjects( small_img, cascade, storage,
                                            1.2, 2, 0,
                                            cvSize(40, 40) );

        /*
        t = (double)cvGetTickCount() - t;
        printf( "detection time = %gms\n", t/((double)cvGetTickFrequency()*1000.) );
        */

        for( i = 0; i < (faces ? faces->total : 0); i++ )
        {
            CvPoint pt1, pt2;

            CvRect* r = (CvRect*)cvGetSeqElem( faces, i );
            pt1.x = r->x*scale;
            pt2.x = (r->x+r->width)*scale;
            pt1.y = r->y*scale;
            pt2.y = (r->y+r->height)*scale;

            cvRectangle( img, pt1, pt2, CV_RGB(255,0,0), 3, 8, 0 );
        }
    }

    cvShowImage( "result", img );
    cvReleaseImage( &small_img );
}



int ndMain ()
{
    CvCapture *capture;
    IplImage *frame, *frame_copy = 0;

    cvInitSystem (0, 0);

    ndHAL_Delay (10);
    ndHAL_EnableMousePointer ();

    cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name, 0, 0, 0 );

    storage = cvCreateMemStorage(0);

    cvNamedWindow( "result", 1 );

    capture = cvCaptureFromCAM (-1);

    cvSetCaptureProperty (capture, CV_CAP_PROP_FRAME_WIDTH,  208);
    cvSetCaptureProperty (capture, CV_CAP_PROP_FRAME_HEIGHT, 192);
    cvSetCaptureProperty (capture, CV_CAP_PROP_TRASMISSION_MODE, 8);

    if( capture )
    {
        for(;;)
        {

            if( !cvGrabFrame( capture ))
                break;

            frame = cvRetrieveFrame( capture );

            if( !frame )
                break;

            if( !frame_copy )
                frame_copy = cvCreateImage( cvSize(frame->width,frame->height),
                                            IPL_DEPTH_8U, frame->nChannels );

            if( frame->origin == IPL_ORIGIN_TL )
                cvCopy( frame, frame_copy, 0 );
            else
                cvFlip( frame, frame_copy, 0 );

            detect_and_draw( frame_copy );


        }

        cvReleaseImage( &frame_copy );
         cvReleaseCapture( &capture );
    }

    ndDelay (10);
    ndHAL_SystemHalt (0);

    return 0;

}
```

Here there are many differences. Let's see which are the most important.

a) the x86 example uses the CLI (command line interface) to determine the name of the file haar requested. Under nd, a CLI-like passage of parameters could happen using pseudoExec function, but this source doesn't use this function. The nd program simply defines the name of chosen haar in its source code (in this way, you cannot choose the name from CLI: the name is defined in C code and it isn't customizable anymore).

b) the x86 version of HighGUI supports grab from AVI. This function doesn't exist in ndHighGUI, so the section of the code that allows the grab from AVI video has been deleted in nd source.

c) the nd version of the program uses these calls:

```
cvSetCaptureProperty (capture, CV_CAP_PROP_FRAME_WIDTH,  208);
cvSetCaptureProperty (capture, CV_CAP_PROP_FRAME_HEIGHT, 192);
cvSetCaptureProperty (capture, CV_CAP_PROP_TRASMISSION_MODE, 8);
```

This function allows the user to set the width, height and transmission mode of an image grabbed by the camera.

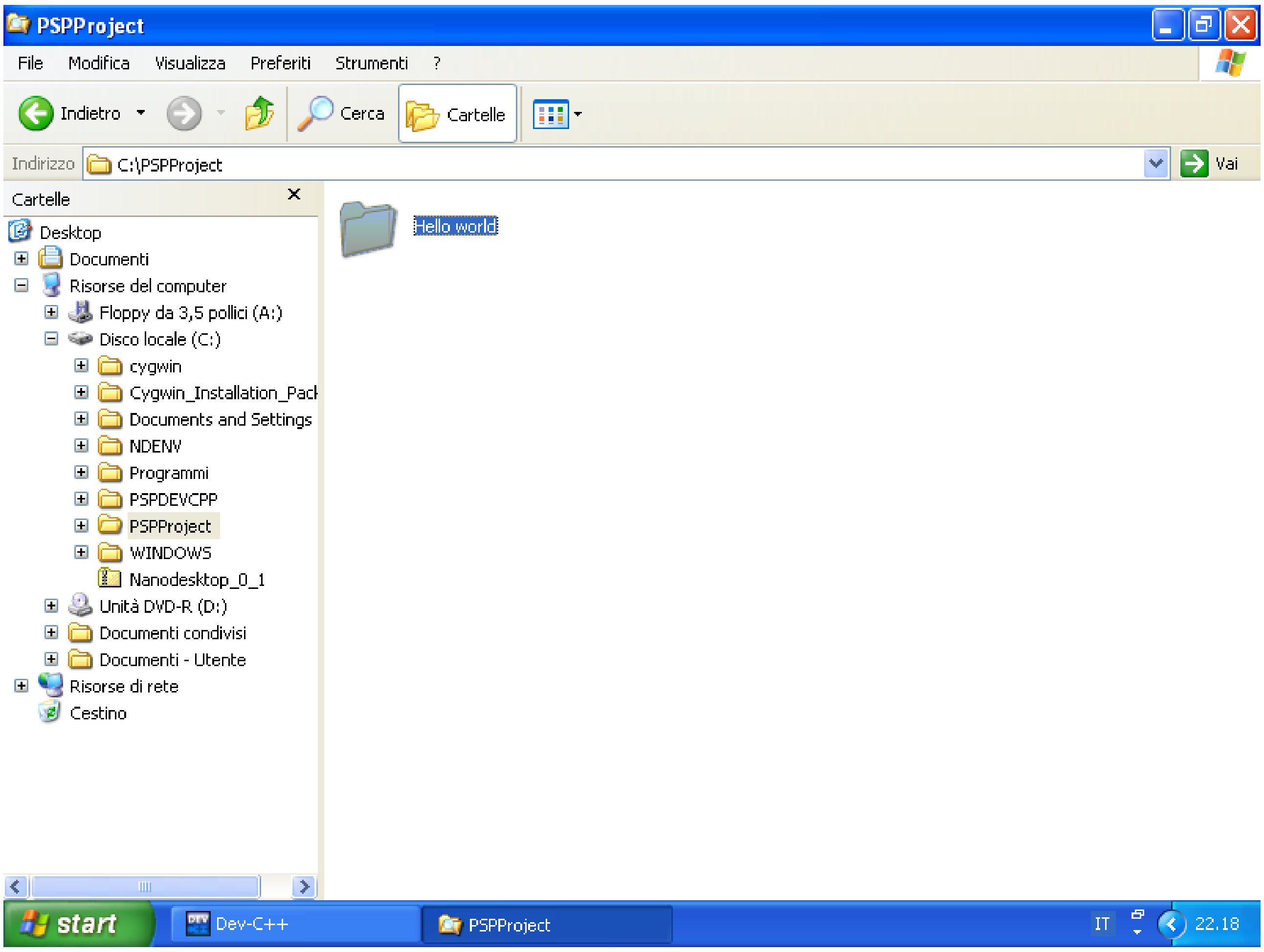
Chapter 30

Your first program in C++

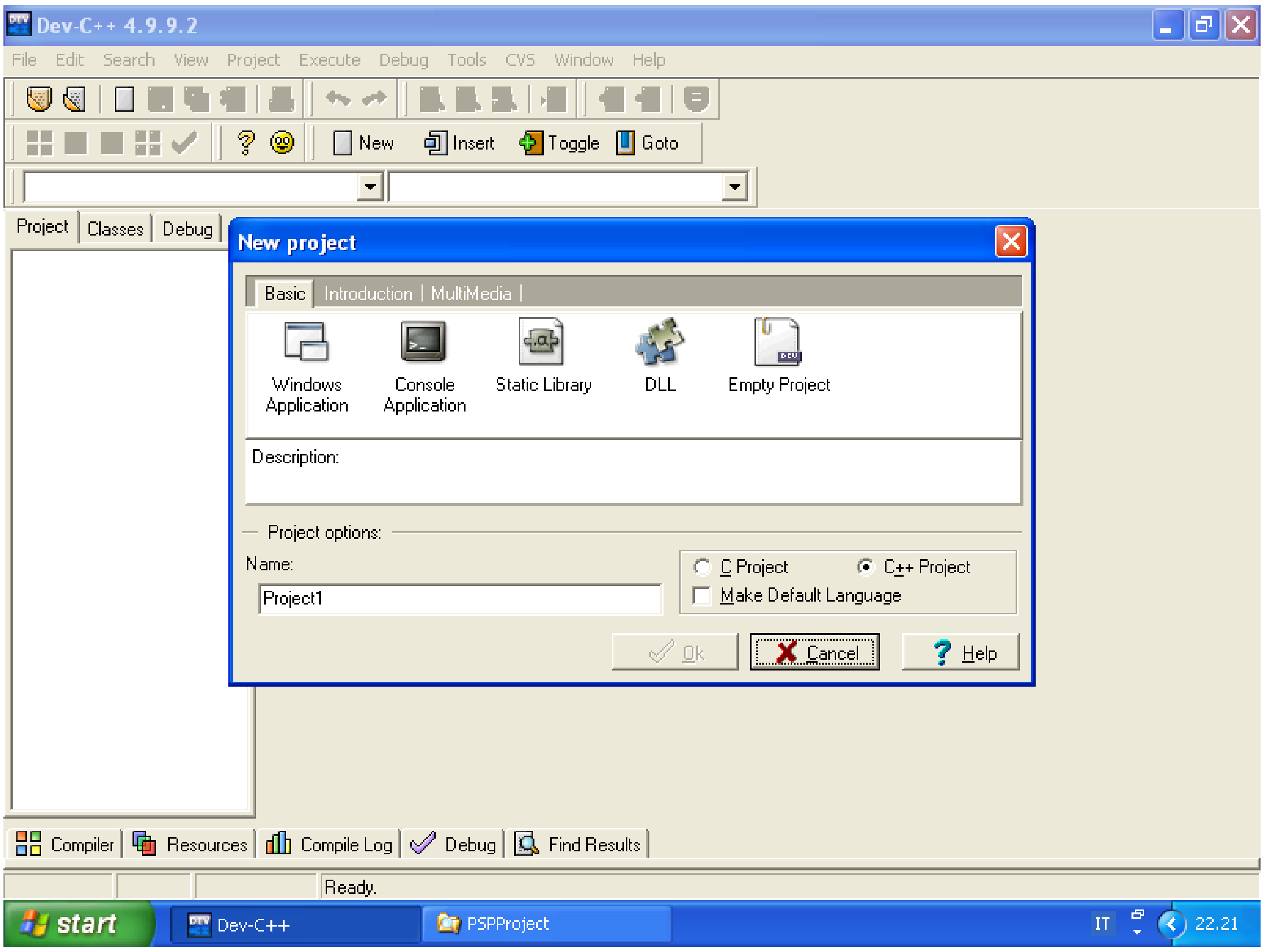
At beginning from Nanodesktop 0.3.4 Visilab has added the support for C++ language. The support is experimental and everybody that finds bugs or troubles is invited to signal it (or to send a patch) to pegasus2000@email.it.

Now, let's see how to create a simple C++ program under Nanodesktop.

Prepare a new folder for your C++ project



Now, open Dev-C++ and choose *File/New* and select *New Project:*


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
Compiler Resources Compile Log Debug Find Results
Ready.
start Dev-C++ PSPProject IT 22.21

New project

Basic | Introduction | MultiMedia

Windows Application, Console Application, Static Library, DLL, Empty Project

Description:

Project options:

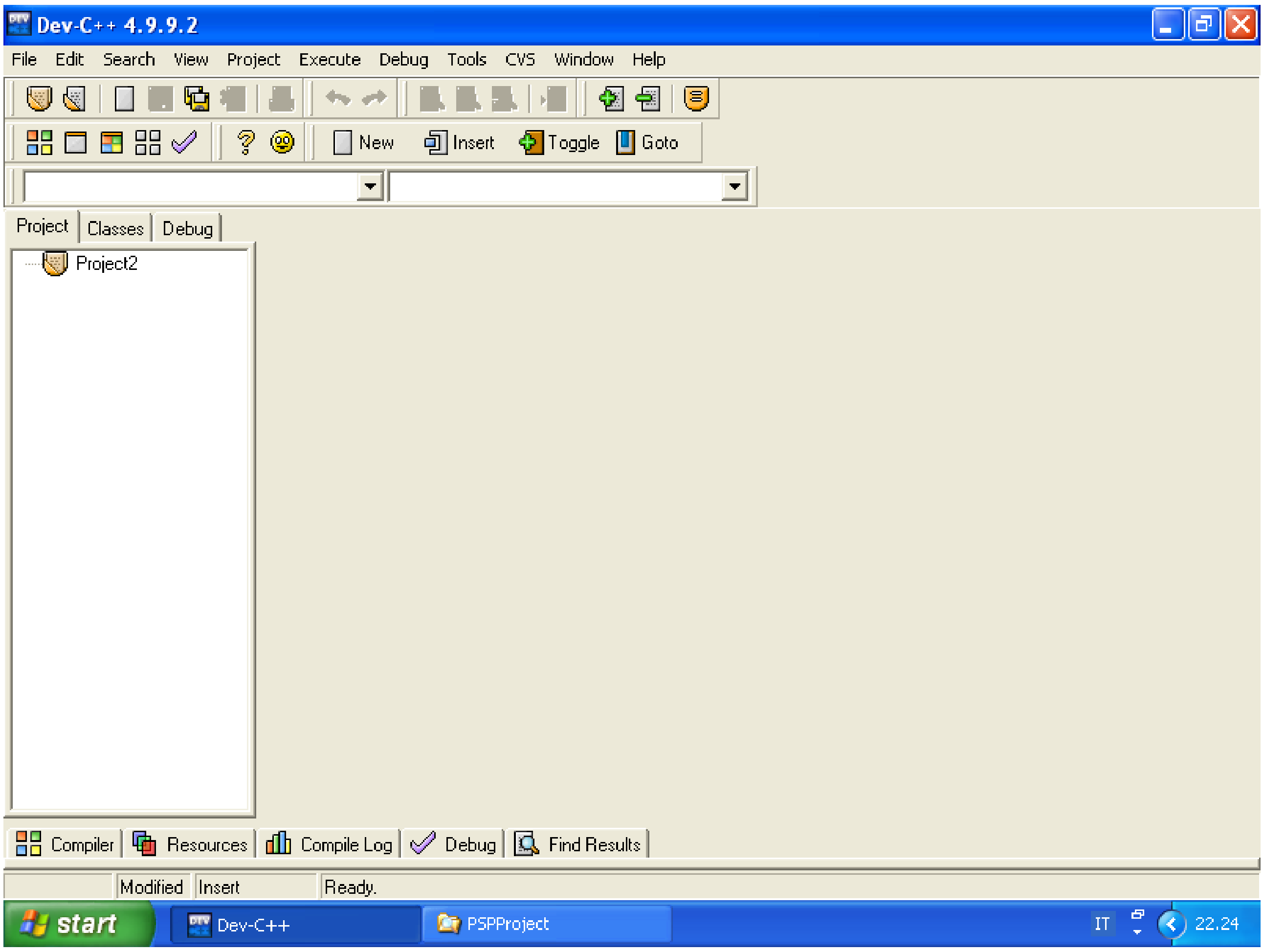
Name: Project1

( ) C Project [✓ C++ Project]

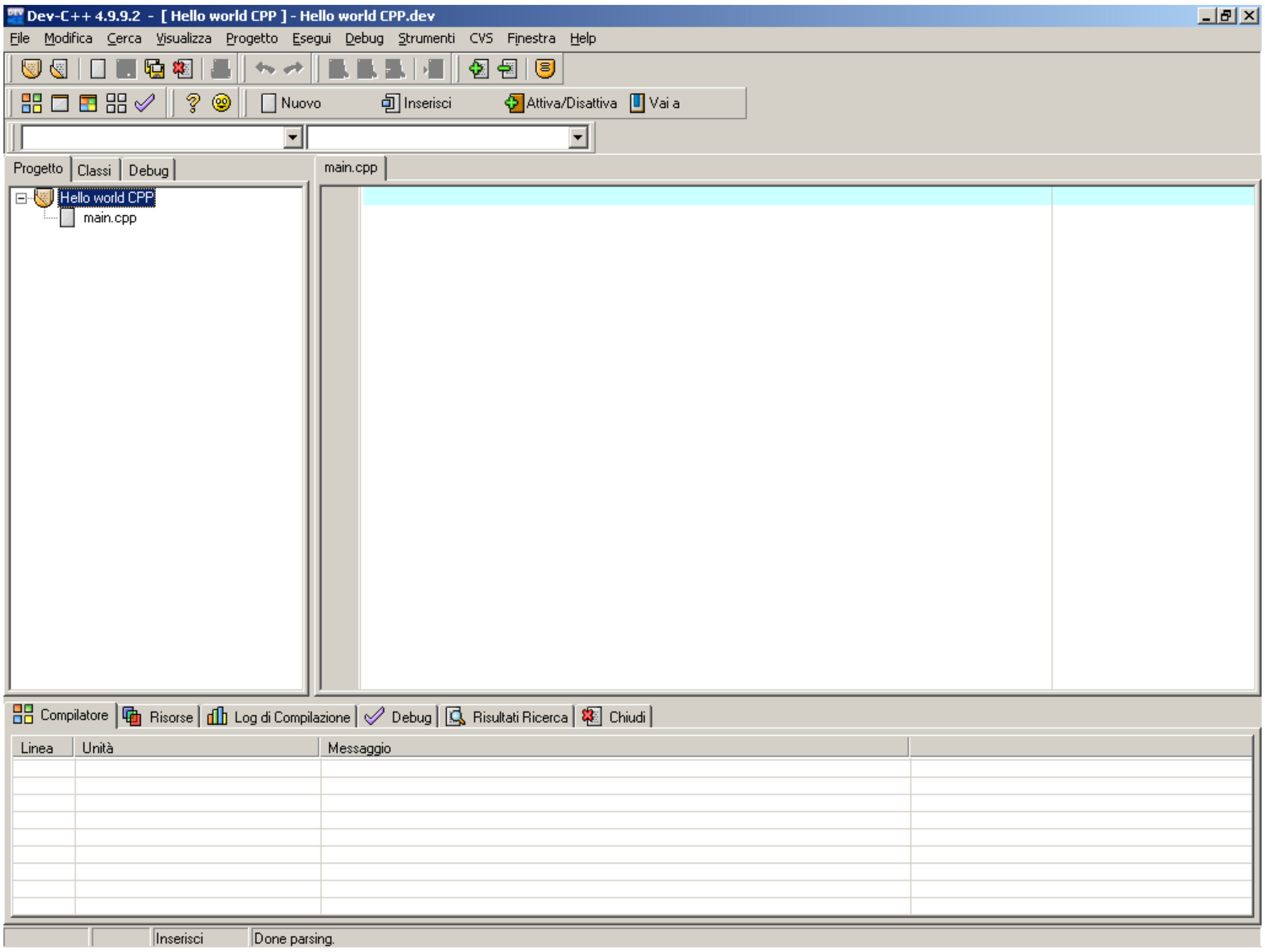
[ ] Make Default Language

Ok | Cancel | Help

Choose *Empty Project:*


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
Project2
Compiler Resources Compile Log Debug Find Results
Modified Insert Ready.
start Dev-C++ PSPProject IT 22.24

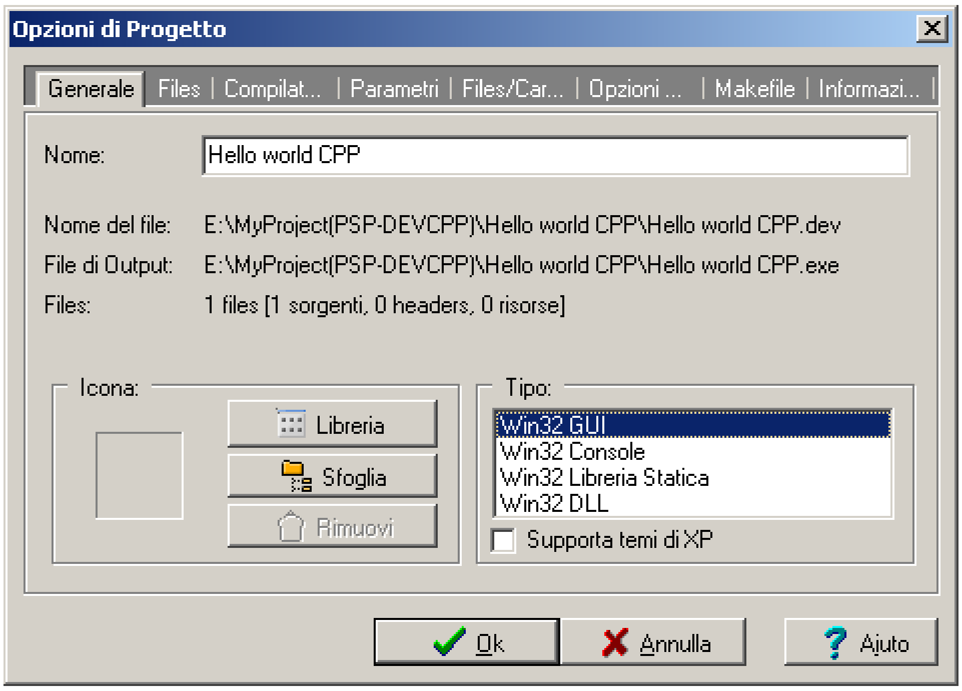
Ok, now add a new C++ file to your project. Call it ***main.cpp***


Dev-C++ 4.9.9.2 - [Hello world CPP] - Hello world CPP.dev
File Modifica Cerca Visualizza Progetto Esegui Debug Strumenti CVS Finestra Help
Nuovo Inserisci Attiva/Disattiva Vai a
Progetto Classi Debug
Hello world CPP
main.cpp
Compilatore Risorse Log di Compilazione Debug Risultati Ricerca Chiudi
Linea Unità Messaggio
Inserisci Done parsing.

You can insert your first program:

```
#include <iostream>

#include <nanodesktop.h>


int main()
{
    ndInitSystem ();
    using namespace std;

    cout << "Hello World!";
    return 0;
}
```

Now save your project and prepare to build it. Go to Project and select *Options:*


Opzioni di Progetto
Generale | Files | Compilat... | Parametri | Files/Car... | Opzioni ... | Makefile | Informazi...
Libreria
Sfoglia
Rimuovi
Ok
Annulla
Aiuto

Nome: Hello world CPP

Nome del file: E:\MyProject(PSP-DEVCPP)\Hello world CPP\Hello world CPP.dev

File di Output: E:\MyProject(PSP-DEVCPP)\Hello world CPP\Hello world CPP.exe

Files: 1 files [1 sorgenti, 0 headers, 0 risorse]

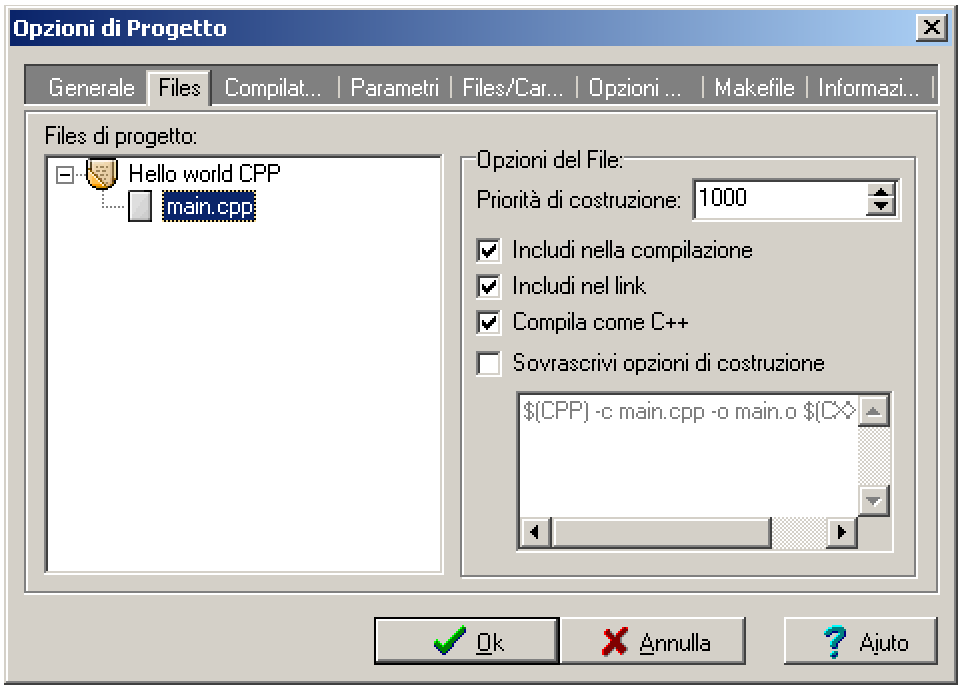
Icona:

Tipo:
- [✓ Win32 GUI]
- Win32 Console
- Win32 Libreria Statica
- Win32 DLL

☐ Supporta temi di XP

BE CAREFUL:

'Type' MUST BE Win32 GUI. If you choose a different type, like Win32 Console, Dev-C++ will pass to psp-gcc the parameter –windows and you'll obtain an error during compilation.


Opzioni di Progetto
Generale | Files | Compilat... | Parametri | Files/Car... | Opzioni ... | Makefile | Informazi...
Ok
Annulla
Aiuto

Files di progetto:
- Hello world CPP
  - main.cpp

Opzioni del File:

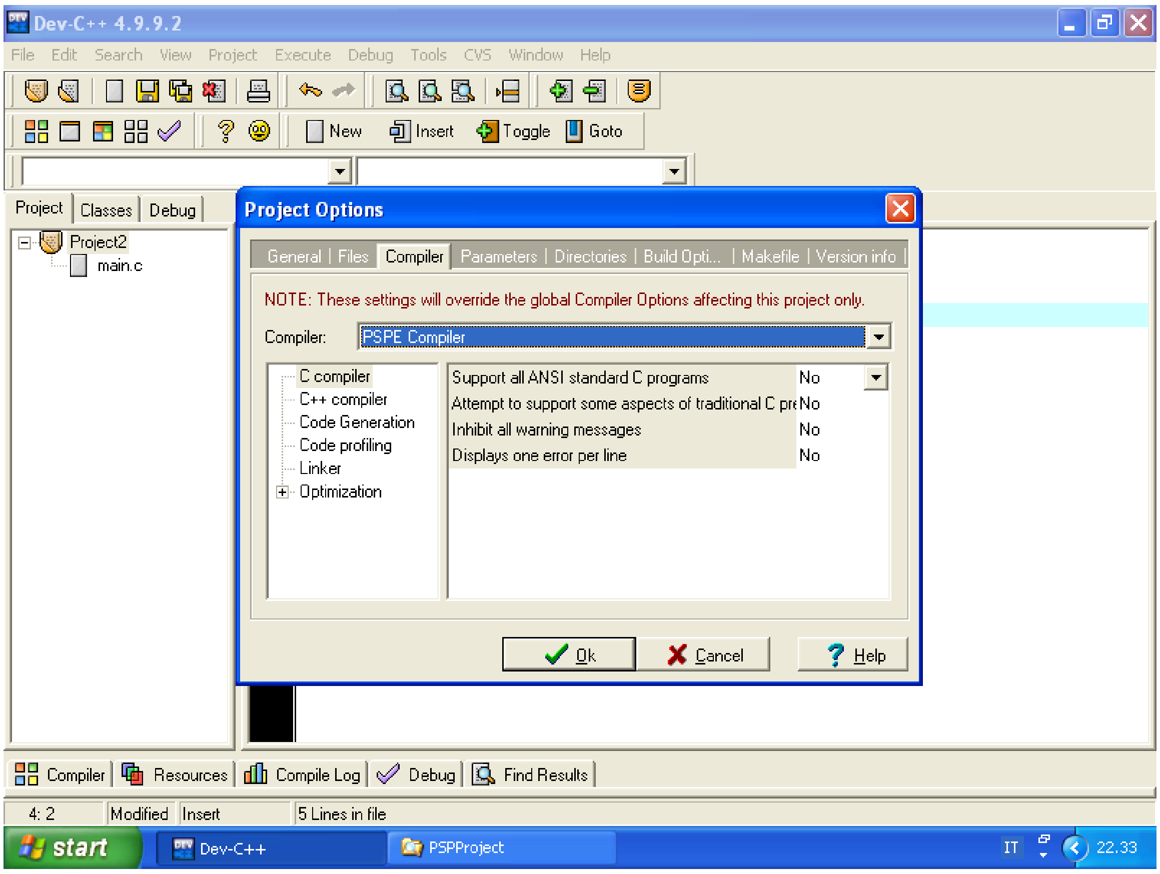
Priorità di costruzione: 1000

☑ Includi nella compilazione
☑ Includi nel link
☑ Compila come C++
☐ Sovrascrivi opzioni di costruzione

```
$(CPP) -c main.cpp -o main.o $(CX
```

Note that we have selected the option "Compile as C++".

Now choose the PSPE Compiler set:


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
Project2
main.c
Project Options
General Files Compiler Parameters Directories Build Opti... Makefile Version info
Ok Cancel Help
Compiler Resources Compile Log Debug Find Results
4: 2 Modified Insert 5 Lines in file
start Dev-C++ PSPProject IT 22.33

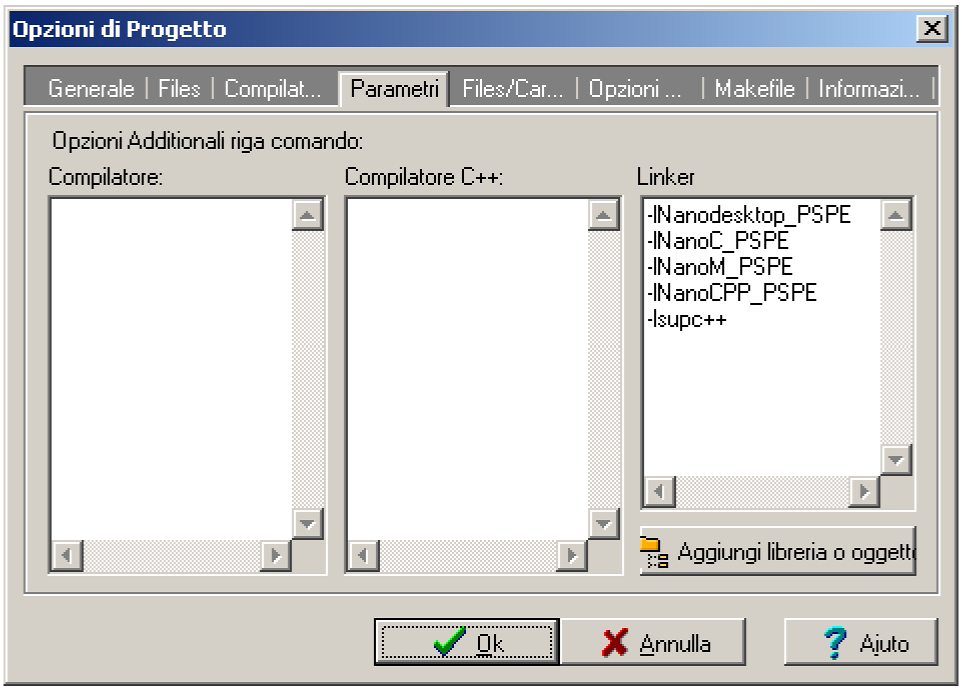
NOTE: These settings will override the global Compiler Options affecting this project only.

Compiler: PSPE Compiler

- C compiler
- C++ compiler
- Code Generation
- Code profiling
- Linker
- Optimization

Support all ANSI standard C programs	No
Attempt to support some aspects of traditional C pre	No
Inhibit all warning messages	No
Displays one error per line	No

Now, choose the required libraries. For now, you can choose **–lNanodesktop_PSPE**, **-lNanoC_PSPE**, **-lNanoM_PSPE, -lNanoCPP_PSPE, -lsupc++** (-l means -ELLE)


Opzioni di Progetto
Generale Files Compilat... Parametri Files/Car... Opzioni ... Makefile Informazi...
Aggiungi libreria o oggett
Ok Annulla Aiuto

Opzioni Additionali riga comando:

Compilatore:

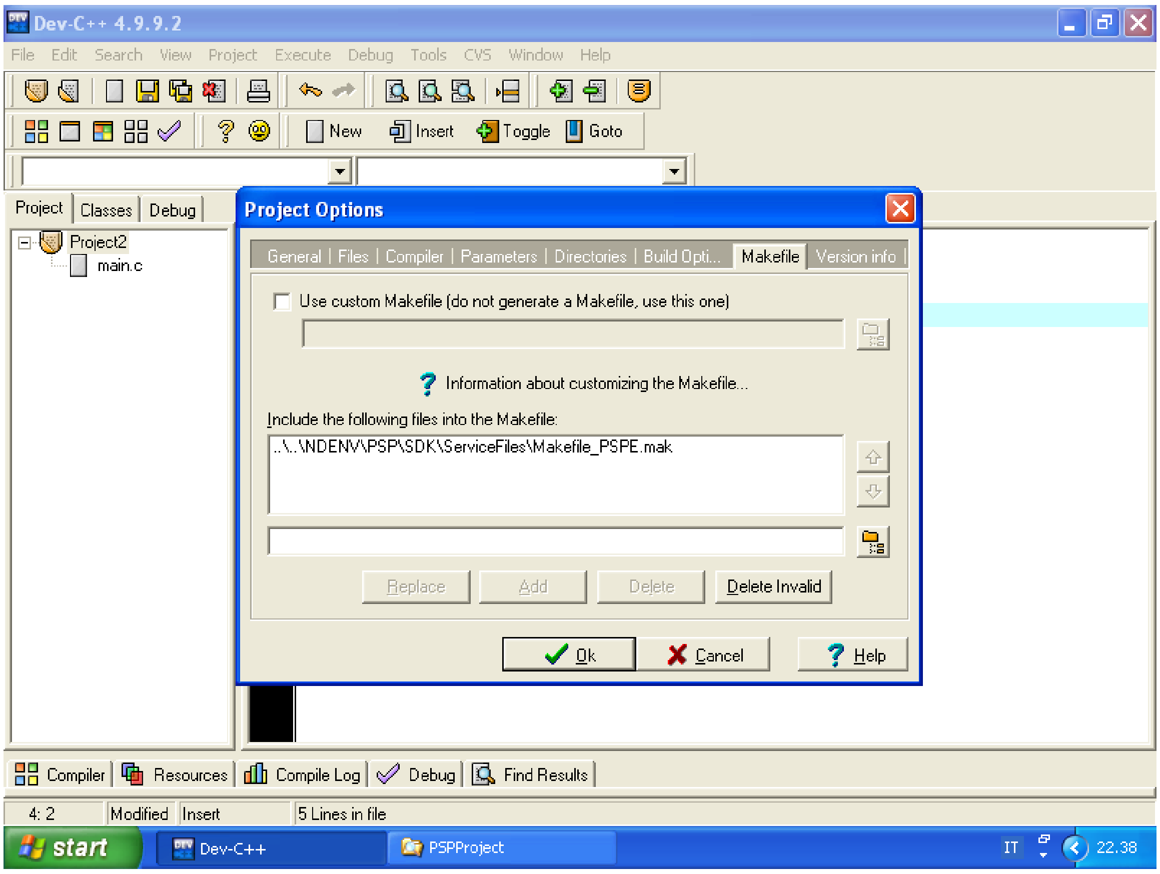
Compilatore C++:

Linker

```
-lNanodesktop_PSPE
-lNanoC_PSPE
-lNanoM_PSPE
-lNanoCPP_PSPE
-lsupc++
```

Now this is the most important stage: you must tell Dev-CPP to use a custom Makefile for PSPE.

This Makefile is in **<ndenv folder>\PSP\SDK\ServiceFiles\Makefile_PSPE.mak**.


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
Project2
main.c
Project Options
General Files Compiler Parameters Directories Build Opti... Makefile Version info
Information about customizing the Makefile...
Replace Add Delete Delete Invalid
Ok Cancel Help
Compiler Resources Compile Log Debug Find Results
start Dev-C++ PSPProject IT 22.38

☐ Use custom Makefile (do not generate a Makefile, use this one)

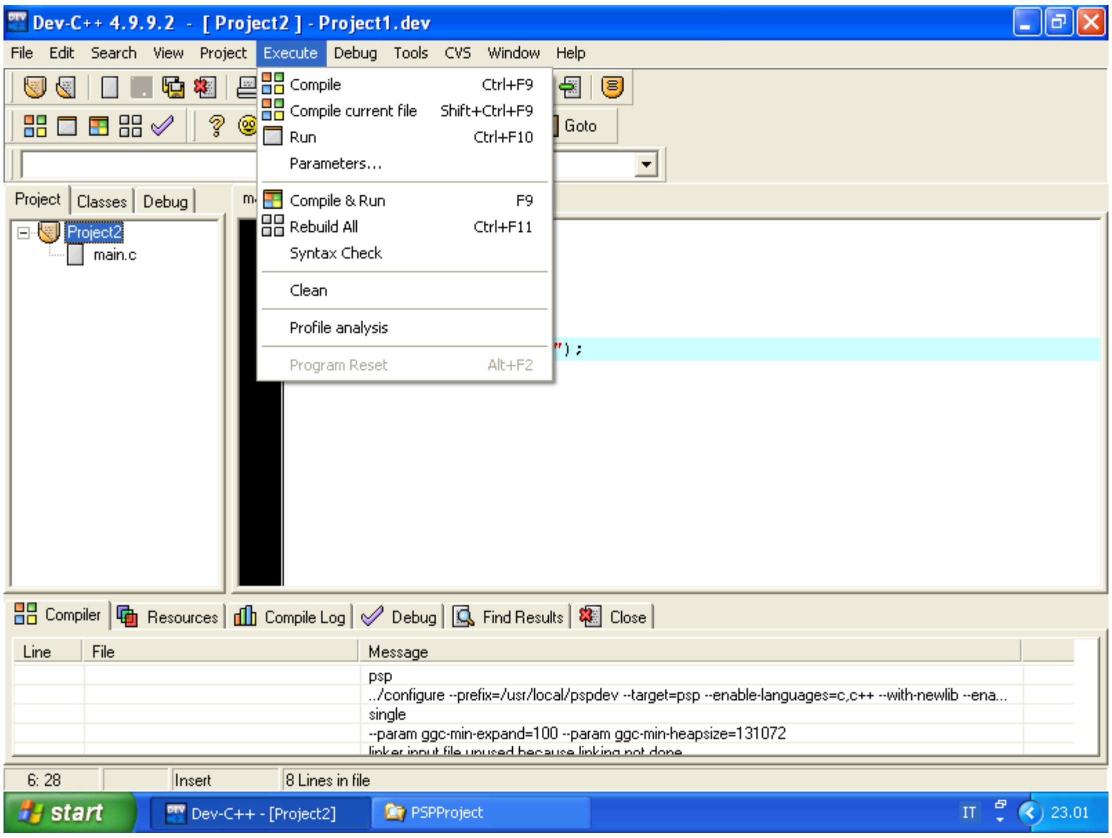
Include the following files into the Makefile:

```
..\..\NDENV\PSP\SDK\ServiceFiles\Makefile_PSPE.mak
```

```
4: 2    Modified  Insert    5 Lines in file
```

This file will recall the PSPE emulator. When you want to recompile for the real Playstation Portable, you will have to change the libraries (**_PSP** instead of **_PSPE**) and the makefile (**Makefile_PSP.mak** instead of **Makefile_PSPE.mak**).

We are now ready to compile. Choose Execute/Rebuid All:

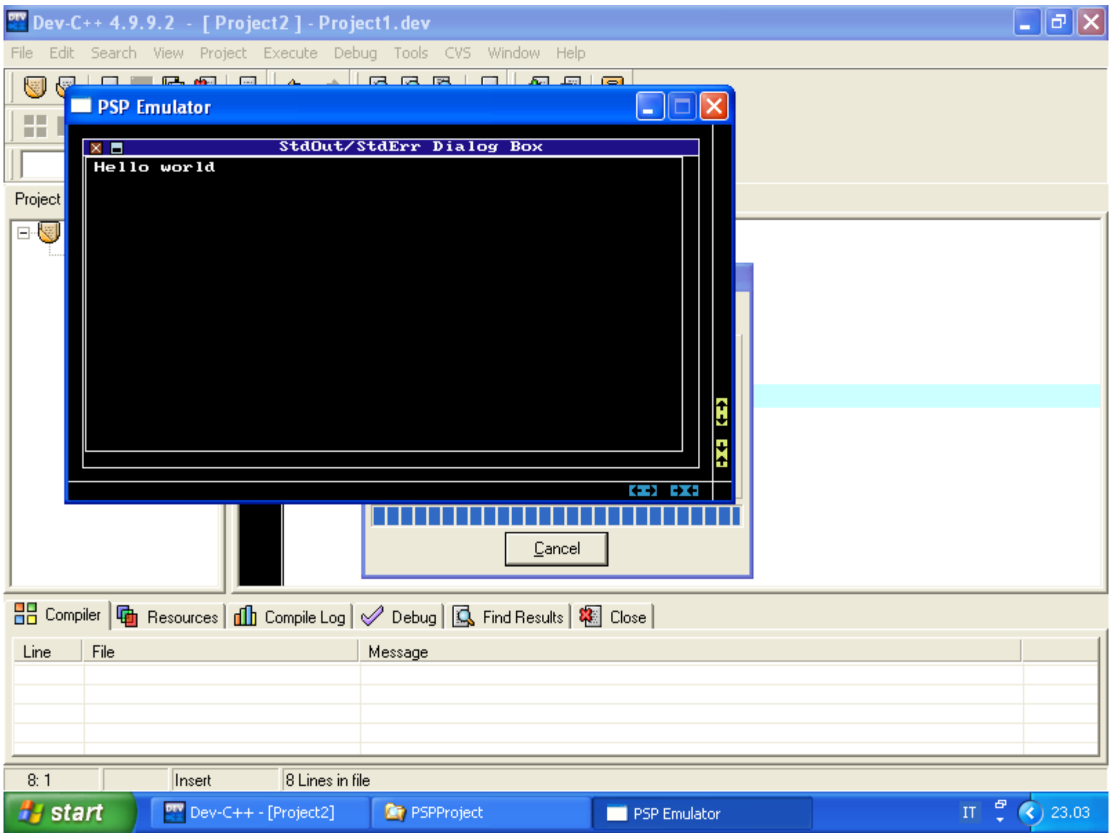

Dev-C++ 4.9.9.2 - [Project2] - Project1.dev
File Edit Search View Project Execute Debug Tools CVS Window Help
Compile Ctrl+F9
Compile current file Shift+Ctrl+F9
Run Ctrl+F10
Parameters...
Compile & Run F9
Rebuild All Ctrl+F11
Syntax Check
Clean
Profile analysis
Program Reset Alt+F2
Goto
Project Classes Debug
Project2
main.c
Compiler Resources Compile Log Debug Find Results Close
start Dev-C++ - [Project2] PSPProject IT 23.01

```
Line  File  Message
            psp
            ../configure --prefix=/usr/local/pspdev --target=psp --enable-languages=c,c++ --with-newlib --ena...
            single
            --param ggc-min-expand=100 --param ggc-min-heapsize=131072
            linker input file unused because linking not done
```

```
6: 28        Insert    8 Lines in file
```

Dev-CPP will compile your source.

The system will create a subfolder in the project folder named *pbp_for_pspe:* this will contain the executable EBOOT_PSPE.PBP. The system will also be able to run PSPE automatically:



```
Hello world
```

Line	File	Message

8: 1 Insert 8 Lines in file

If you want to recompile your program for **PSP**, **PSP KSU mode** or **PSP CFW mode**, you can follow the same instructions seen in **chapter 4**.

Appendix A

ndHighGUI prototypes

Dario Ignazio
Filippo Battaglia

Here we can see the ndHighGUI prototypes for ndOpenCV programming.

Section A: Initialization

cvInitSystem

Initializes HighGUI

```
int cvInitSystem( int argc, char** argv );
```

argc

Number of command line arguments.

argv

Array of command line arguments

The function cvInitSystem initializes ndHighGUI. The parameter argc and argv are ignored (put them to 0,0).

In Nanodesktop environment, cvInitSystem internally recalls ndInitSystem. This reinitializes the Nanodesktop graphical subsystem, and also provides to reset other system variables that are necessary for the correct working of ndHighGUI.

Section B: Managing window

cvNamedWindow

Create an OpenCV window

```
int cvNamedWindow( const char* name, ndint64 flags );
```

name

Name of the window which is used as identifier and that appears in the window caption.

flags

Flags of the window. It is a 64-bit parameter, bit-mapped in way of mantaining compatibility with Intel x86 OpenCV code. Using this parameter and the right keys, you can pass some informations to ndHighGUI without breaking the compatibility with Intel prototypes.

If you don't specify any option, when you'll use cvShowImage to show an image, the system will provide to move and resize the target window, in way to best fit the screen and to show correctly the desidered image.

This behaviour can be changed using the right 64-bit keys or constants, that have to be or-ed together in *flag* parameter.

If you specify the constant **CV_USE_NDWSWND,** Nanodesktop will use the window space (see page 79) to show the image in a space that is partially over the ROI bounds of the window. A window that has been created using this flag, cannot be associated with a trackbar.

If you specify the constant **CV_FORCE,** the system won't try to move/resize the window before visualizing the image. It will use the size and the position that has been stated when the window has been created by cvNamedWindow. This position and size can be defined using CVKEY_SETPOS and CVKEY_SETSIZE: if the user doesn't specify these informations, the system will use automatically the content of the system variables HGUI_DefaultLenX, HGUI_DefaultLenY for size and nd PosGen generator for the position.

The option **CV_MENUSUPPORTED** creates a window with Nanodesktop menu support.

The option **CV_WINDOW_AUTOSIZE** is mantained only for compatibility reason

The user can set the position and the size of the window using the two 64-bit keys: **CVKEY_SETPOS** (PosX, PosY) and **CVKEY_SETSIZE** (LenX, LenY).

The color of the window is defined by internal system variables. The user can change the colors of the next window using the routine cvSetNextColorWnd or similar (see further).

If a window with such name already exists, the function does nothing.

The routine return 1 on success and 0 on failure.

Examples:

```
cvNamedWindow( "OpenCV result", 0 );
```

Create a window. The system will resize/move the window when you'll try to use cvShowImage in order to show an image inside it.

```
cvNamedWindow( "OpenCV result", CV_USE_NDWSWND );
```

Create a window that will use WS space in order to show images that are greater than the window dimensions.

```
cvNamedWindow ("Example", CVKEY_SETPOS (0, 0) | CVKEY_SETSIZE (100, 100) | CV_MENUSUPPORTED )
```

Create a window called Example, with position (0,0), size (100,100) and with support for nanodesktop menu.

```
cvNamedWindow ("Example", CVKEY_SETPOS (0, 0) | CVKEY_SETSIZE (100, 100) | CV_FORCE )
```

Create a window called Example, with position (0,0), size (100,100). The window won't be resized/moved by nd when the user will try to show a cvimage in it.

cvDestroyWindow

Destroy a window

```
void cvDestroyWindow( const char* name );
```

name

Name of the window to be destroyed.

The function cvDestroyWindow destroys the OpenCV window with a given name. It ignores all windows that haven't be created using cvNamedWindow.

cvDestroyAllWindows

Destroy all HighGUI windows

```
void cvDestroyAllWindows(void);
```

The function cvDestroyAllWindows destroys all opened OpenCV windows. The windows that haven't been created in OpenCV context (i.e through cvNamedWindow) aren't affected by the routine.

cvResizeWindow

Resize an OpenCV window

```
void cvResizeWindow( const char* name, int width, int height );
```

name

Name of the window to be resized.

width

New width

height

New height

The function cvResizeWindow changes the dimensions of the OpenCV window.

cvMoveWindow

Move an OpenCV window to a new position

```
void cvMoveWindow( const char* name, int x, int y );
```

name

Name of the window to be resized.

x

New x coordinate of top-left corner

y

New y coordinate of top-left corner

The function cvMoveWindow changes position of the window.

cvGetNdWindowHandler, cvFindWindowByName

Gets window handle by name (the two functions have the same behaviour)

```
char cvGetNdWindowHandler (char *name);

char cvFindWindowByName (char* NameWindowToSearch);
```

name

Name of the window.

The function cvGetNdWindowHandler returns Nanodesktop handle associated to the window

cvGetNdWindowTitle

This routine returns a pointer to the name of an OpenCV window, associated with a given nd handle.

```
char *cvGetNdWindowTitle (char WndHandle);
```

cvShowImage

Shows the image in the specified window

```
void cvShowImage( const char* name, const CvArr* image );
```

name

Name of the window.

image

Image to be shown.

The function cvShowImage shows the image in the specified OpenCV window. If the user has specified a ROI (Region of Interest) in an OpenCV image, ndHighGUI will show only the ROI and not the entire image.

The behaviour of the routine depends also by the parameters of the window, specified when it has been created using cvNamedWindow routine. In general, nd will resize/remove the window in order to adapt it to the best visualization possibility for the image.

If the window has been created with CV_USE_NDWSWND option disabled, the routine will perform preliminary calculations in order to determine how much space you can give to the image, viewing it together with other trackbars eventually associated to the window, and without using the scroll bars. When the system has completed the computations, the image is shown, eventually with the opportune scaling or resizing.

If the window has been created using CV_USE_NDWSWND option enabled, the system will use window space to show the image. If the image has dimensions that are greater than the window dimensions, Nanodesktop will show the image using window space (a part of the image will be visualized in over-screen, and the scroll bars will appear in the screen).

No attempt to resize/move the window is done when it has been created using CV_FORCE option. In this case, the image will be visualized using only an opportune scaling.

cvCreateTrackbar

Creates the trackbar and attaches it to the specified window

```
int cvCreateTrackbar( const char* trackbar_name, const char* window_name,
                      int* value, int count, CvTrackbarCallback on_change );
```

trackbar_name

Name of created trackbar.

window_name

Name of the window which will be used as a parent for created trackbar.

value

Pointer to the integer variable, which value will reflect the position of the slider. Upon the creation the slider position is defined by this variable.

count

Maximal position of the slider. Minimal position is always 0.

on_change

Pointer to the function to be called every time the slider changes the position. This function should be prototyped as

```
void CvTrackbarCallback(int pos)
```

Can be NULL if callback is not required.

The function cvCreateTrackbar creates the trackbar (a.k.a. slider or range control) with the specified name and range, syncronizes the control variable with trackbar position and specifies callback function to be called when the position of the trackbar changes.

The created trackbar is displayed at the top of the specified window.

The routine is emulated in a way almost identical to what is done on the Intel x86 systems, except for the following restrictions :

-The trackbar cannot be associated if the window has been created using cvNamedWindow with the option CV_USE_NDWSWND.

-The new trackbar will appear only when you'll update the window contents with a new call to cvShowImage. The user can force this mechanism by calling **cvForceTrackbarRender**, but this is risky.
In fact, it is necessary that the OpenCV image that has been visualized in the OpenCV target window, are still in memory when you call cvForceTrackbarRender. If, instead, you have deallocated the image associated with a window (for example, using cvReleaseImage) and in a second moment you try to use cvForceTrackbarRender, the system will crash.

-The name of trackbar cannot be longer than 31 characters: the characters in excess are truncated .

The routine returns 1 if successful, and 0 in case of failure.

cvGetTrackbarPos

Retrieves trackbar position

```
int cvGetTrackbarPos( const char* trackbar_name, const char* window_name );
```

trackbar_name

Name of trackbar.

window_name

Name of the window which is the parent of trackbar.

The function cvGetTrackbarPos returns the current position of the specified trackbar.

cvSetTrackbarPos

Sets trackbar position

```
void cvSetTrackbarPos( const char* trackbar_name, const char* window_name,
int pos );
```

trackbar_name

Name of trackbar.

window_name

Name of the window which is the parent of trackbar.

pos

New position.

The function cvSetTrackbarPos sets the position of the specified trackbar.

cvSetMouseCallback

Not supported yet

cvWaitKey

Waits for a pressed key

```
int cvWaitKey( int delay=0 );
```

delay

Delay in nanoseconds.

The function cvWaitKey waits for key event infinitely (delay<=0) or for "delay" milliseconds. Returns the code of the pressed key or -1 if no key were pressed until the specified timeout has elapsed.

Section C: Load/save/convert images

cvLoadImage

Loads an image from file

```
IplImage* cvLoadImage( const char* filename, int iscolor);
```

filename

Name of file to be loaded.

iscolor

Specifies the use of color in loaded image:
if it is 1, the returned image will be in BGR format, also if the image on the disk is in gray-tones. The eventual conversion is done in a way trasparent to the user.

if it is 0, the returned image will be in gray tones, also if the image on the disk is RGB.

The function cvLoadImage loads an image from the specified file and returns the pointer to the loaded image.
Internally, this routine uses Nanodesktop API for image opening. So, the formats that can be opened are the same supported by the version of nd in use.

If you are using a version of Nanodesktop without support for Dev-IL, you will only be able to load BMP image files.

If you are using a version of Nanodesktop with Dev-IL support enabled, you will able to load all formats that are supported by ndDEVIL (BMP, GIF, JPG, PNG, PNM, PGM, PSD, PCX, ICO, TGA, SGI, TIF, XPM).

cvSaveImage

Saves an image to the file

```
int cvSaveImage( const char* filename, const CvArr* image );
```

filename

Name of the file.

image

Image to be saved.

The function cvSaveImage saves the image to the specified file.

The image format is chosen depending on the filename extension - see cvLoadImage. Also this routine internally uses Nanodesktop API so, you will be able to save the image in a format different than BMP only if you are using a version of Nanodesktop with Dev-IL support enabled.

Only 8-bit single-channel or 3-channel (with 'BGR' channel order) images can be saved using this function. If the format, depth or channel order is different, use cvCvtScale and cvCvtColor to convert it before saving.

The image on disk will *always be* in 3 channel formats, aswell as if the saved image was in gray-tones.

cvConvertImage

Converts one image to another with optional vertical flip

```
void cvConvertImage( const CvArr* src, CvArr* dst, int flags=0 );
```

src

Source image.

dst

Destination image. Must be single-channel or 3-channel 8-bit image.

flags

The operation flags:

CV_CVTIMG_FLIP	flips the image vertically
CV_CVTIMG_SWAP_RB	swap red and blue channels. In OpenCV color images have BGR channel order, however on some systems the order needs to be reversed before displaying the image (cvShowImage does this automatically).

The function cvConvertImage converts one image to another and flips the result vertically if required.

cvConvertNDItoIPLImage

Converts a Nanodesktop image into an OpenCV image

```
IplImage *cvConvertNDItoIPLImage (struct ndImage_Type *MyImage, int ColorDepth,
int NrChannels, int Magn);
```

MyImage

Pointer to the struct ndImage_Type that contains the informations about the target nd image

ColorDepth

The color depth of the destination OpenCV image.

NrChannels

The nr. of channels of the destination OpenCV image (3 for RGB images and 1 for gray-tones images)

Magn

Scale factor (in integer). If you set it to 1, the image won't be scaled; if you set it to x, the image will be magnified by a x factor in length and in width

ndIMG_ConvertIPLtoNDIImage

Converts an OpenCV image into a Nanodesktop image

```
char ndIMG_ConvertIPLtoNDIImage (struct ndImage_Type *MyImage, IplImage
*MyIPLImage, float ScaleX, float ScaleY, char IsColor, char ColorFormat)
```

In closed terms, this is a Nanodesktop routine and not a ndHighGUI routine. So, the error code is returned to the call routine, without graphical notification. Let's see the significant of the parameters:

IplImage	The image to be converted
MyImage	Pointer to a struct ndImage_Type that will store the information about the new image
ScaleX,ScaleY	Scale factor
IsColor	Set it to 0 if you want a gray tones image, 1 for coloured image
ColorFormat	ColorFormat of the nd destination image (NDRGB or NDMGKNB).

The allocation of the nd image is executed internally in the routine.

Section D: Webcam support

CvCapture

Video capturing structure

```
typedef struct CvCapture CvCapture;
```

The structure *CvCapture* does not have public interface and is used only as a parameter for video capturing functions.

CvCaptureFromFile

Initializes capturing video from file

```
CvCapture* cvCaptureFromFile( const char* filename );
```

filename

Name of the video file.

At present ndOpenCV doesn't support the capture from an AVI file. This routine is maintained here, only for compatibility reason)

The routine always returns 0 (in the original OpenCV this corresponds to an error)

cvCaptureFromCAM

Initializes capturing video from camera

```
CvCapture* cvCaptureFromCAM( int WebCamID );
```

WebCamID

> 32 bit identification code of the camera to be used. If there is only one camera or it does not matter what camera has to be used -1 may be passed.

The function cvCaptureFromCAM allocates and initializes the *CvCapture structure* for reading a video stream from the camera. The routine uses internally **ndHAL_CAM_ActivateCamera (WebCamID)**;

To release the structure, use **cvReleaseCapture**.

This routine reinitializes the webcam defined by WebCamID, and prepares the system to receive the data. If WebCamID is set to -1, the system will use the first camera available. The programmer can also select the camera that has to be used, through appropriate WebCam IDs. Actually, the codes that are recognized are the following:

ND_USE_PSP_GOCAM	Use the GoCam
ND_USE_PSP_EYESVR	Use Eyeserver

So, for example, cvCaptureFromCAM (ND_USE_PSP_EYESVR), forces the system to use Eyeserver webcam.

If the operation is terminated without errors, the routine returns the address of the struct CvCapture that has been allocated automatically by the system and associated to the camera.

If, instead, the system has encountered errors during the operation, the routine returns 0, and sets the system variable **HGUI_SystemError** to a 32-bit code indicating the origin of the trouble.

cvReleaseCapture

Releases the CvCapture structure

```
void cvReleaseCapture( CvCapture** capture );
```

capture

> pointer to video capturing structure.

The function cvReleaseCapture releases the CvCapture structure allocated by cvCaptureFromFile or cvCaptureFromCAM.

cvGrabFrame

Grabs frame from camera or file

```
int cvGrabFrame( CvCapture* capture );
```

capture

video capturing structure.

The function cvGrabFrame grabs the frame from camera or file. The grabbed frame is stored internally. To retrieve the grabbed frame, cvRetrieveFrame should be used.

cvRetrieveFrame

Gets the image grabbed with cvGrabFrame

```
IplImage* cvRetrieveFrame( CvCapture* capture );
```

capture

video capturing structure.

The function cvRetrieveFrame returns the pointer to the image grabbed with cvGrabFrame function. The returned image should not be released or modified by user.

cvQueryFrame

Grabs and returns a frame from camera or file

```
IplImage* cvQueryFrame( CvCapture* capture );
```

capture

video capturing structure.

The function cvQueryFrame grabs a frame from camera or video file, decompresses and returns it. This function is just a combination of cvGrabFrame and cvRetrieveFrame in one call. The returned image should not be released or modified by user.

cvGetCaptureProperty

Gets video capturing properties

```
int cvGetCaptureProperty( CvCapture* capture, int property_id );
```

capture

video capturing structure.

property_id

ID that defines the requested information

This function has been implemented only partially. There are different constant identifiers that you can use to indicate which information do you want. The routines that would have to return these informations, however, haven't been all implemented, so some ID will return -1 (an error condition).

A list of recognized ID is the following:

CV_CAP_PROP_TRASMISSION_MODE:

returns the trasmission mode for Eyeserver or GoCam (8 per gray tones trasmission mode, 16 per RGB555 mode and 24 per RGB color mode)

CV_CAP_PROP_POS_MSEC:

not implemented yet

CV_CAP_PROP_POS_FRAMES:

not implemented yet

CV_CAP_PROP_POS_AVI_RATIO:

not implemented yet

CV_CAP_PROP_FRAME_WIDTH:

returns the width of captured video frame

CV_CAP_PROP_FRAME_HEIGHT:

returns the height of captured video frame

CV_CAP_PROP_FPS:

returns the number of frame captured for second. It is unimplemented (to do), so it returns always 8.

CV_CAP_PROP_FOURCC:

not implemented yet

cvSetCaptureProperty

Set a video capturing property

```
int cvSetCaptureProperty( CvCapture* capture, int property_id, double Value );
```

capture

video capturing structure.

property_id

ID that defines the requested information

Value

the new value for the target parameter

This function has been implemented only partially. There are different constant identifiers that you can use to indicate which information you want to modify. Some identifiers are mantained only for compatibility with x86 OpenCV, but in reality the respective functionalities have no point under nd, so this ID will return -1 (an error condition).

A list of recognized ID is the following:

CV_CAP_PROP_TRASMISSION_MODE:

sets the trasmission mode for Eyeserver or GoCam (8 per gray tones trasmission mode, 16 per RGB555 mode and 24 per RGB color mode)

CV_CAP_PROP_POS_MSEC:

not implemented yet

CV_CAP_PROP_POS_FRAMES:

not implemented yet

CV_CAP_PROP_POS_AVI_RATIO:

not implemented yet

CV_CAP_PROP_FRAME_WIDTH:

sets the width of captured video frame

CV_CAP_PROP_FRAME_HEIGHT:

sets the height of captured video frame

cvCreateVideoWriter

Not supported yet

cvReleaseVideoWriter

Not supported yet

cvWriteFrame

Not supported yet

Section E: Error notification

HGUI_EnableErrorNotification, HGUI_DisableErrorNotification

Enable or disable notification of the errors determinated by ndHighGUI.

```
void HGUI_EnableErrorNotification (void);
void HGUI_DisableErrorNotification (void);
```

When an error occurs during elaboration, ndHighGUI emits a notification of the error for the user. The programmer can disable this notification. In this case, the system won't notify the error during the execution of the program. In any case, the programmer can always check the error type, in any moment, using the system variable HGUI_SystemError (values different by 0 indicate a trouble).

HGUI_NotifyError

Force the notification of an error

```
void HGUI_NotifyError ( int code, char code2, const char *func_name, const char
*err_msg)
```

Section F: Operations on pixels

NdHighGUI provides some routines for quick access to the single pixel of an image. These routines use a struct called *struct HGUIPixelType* in order to pass informations to the ndHGUI routines.

```
struct HGUIPixelType
{
    unsigned short PosX, PosY;    // Coordinate del punto
    unsigned char NrChannels;     // Posto a 1 per immagini a tono di grigio,
                                   // posto a 3 per immagini a colori RGB
    int ColorDepth;               // Posto a 8,16,32 a seconda della profondità di colore

    float C1;                     // Per immagini RGB, si ha sempre la convenzione
    float C2;                     // C1=R, C2=G, C3=B.
    float C3;

    // Per immagini a toni di grigio, il campo C1 contiene
    // il valore del colore, mentre C2 e C3 sono posti a 0.
};
```

cvRGBToMagicNumber

Convert a RGB value to RGB555 MagicNumber

```
TypeColor cvRGBToMagicNumber (struct HGUIPixelType *MyHGUIPixel )
```

This routine takes care of converting the color of a pixel, expressed in gray tones or through its RGB components in a value said MagicNumber. This number is the equivalent in RGB555 mode. These magic numbers are very important because nd accepts only MagicNumber (or RGB value) as input for its routine.

The necessary data is passed through HGUIPixelType structure, as follows:

PosX and PosY are ignored;

ColorDepth indicates the color depth of the pixel that we want to convert (colors in 256 tones, colors in float ?)

NrChannels indicates the number of channels (1 for pixels in tones of gray, 3 for pixels whose colors are expressed through the RGB components)

C1	is the color R (or the value of gray imaging GRAY)
C2	is the color G
C3	is the color B

Example:

```
struct HGUIPixelType HguiPixel;

HguiPixel.ColorDepth = 16;
HguiPixel.C1 = 0xFFFF;        // Color red
HguiPixel.C2 = 0x0;           // Color green
HguiPixel.C3 = 0xFFFF;        // Color blue

printf ("Magic number returned %X \n", cvRGBToMagicNumber (&HguiPixel));
```

cvConvertPixel

Convert the RGB color values of a pixel in a different format or color depth

```
void cvConvertPixel (struct HGUIPixelType *Record1, struct HGUIPixelType
*Record2)
```

This routine provides to convert a pixel with a given color depth / nr. channels, in another pixel with a different color depth / nr. channels. The result is stored in the C1,C2,C3 registers of *Record2* structure.

CvPutPixelToImage

Put a pixel into an OpenCV image

```
void cvPutPixelToImage (IplImage *MyIPLImage, struct HGUIPixelType
*MyHGUIPixel )
```

This routine puts a pixel into an OpenCV image. The image must be already allocated using cvCreateImage routine. The positions X and Y of the pixel are indicated by the field PosX and PosY of the struct MyHGUIPixel pointed by the routine. The color (in RGB values) and the color depth are indicated by MyHGUIPixel fields too.

The routine doesn't return values: any error code is returned through the system variable HGUI_SystemError. This is a list of the possible error codes:

NDHGUI_PIXEL_OUT_OF_IMAGE the pixel is outside the image
NDHGUI_IMAGE_NOT_INITIALIZED the opencv image isn't initialized
NDHGUI_WRONG_NR_CHANNELS wrong number of channel
NDHGUI_WRONG_COLOR_DEPTH wrong color depth
NDHGUI_UNKNOWN_ERROR unknown error

cvGetPixelFromImage

Get a pixel from an OpenCV image

```
void cvGetPixelFromImage   (IplImage *MyIPLImage, struct HGUIPixelType
*MyHGUIPixel )
```

This routine allows you to extract the features (color values, nr. channels, color depth) of a single pixel of a single OpenCV image. The image must already be initialized using cvCreateImage.

The retrieved data is stored by the routine in the fields of struct MyHGUIPixel pointed by the routine.

The routine doesn't return any value: any error code is returned through the system variable HGUI_SystemError. This is a list of the possible error codes:

NDHGUI_PIXEL_OUT_OF_IMAGE the pixel is outside the image
NDHGUI_IMAGE_NOT_INITIALIZED the opencv image isn't initialized
NDHGUI_WRONG_NR_CHANNELS wrong number of channel
NDHGUI_WRONG_COLOR_DEPTH wrong color depth
NDHGUI_UNKNOWN_ERROR unknown error

Section G: Generation of the new OpenCV windows and color control

cvChangeDefaultWindowDim

Change the default width and height values that the system uses automatically when it creates a new OpenCV window.

```
void cvChangeDefaultWindowDim (unsigned short NewLenX, unsigned short NewLenY);
```

When the user calls the routine **cvNamedWindow** with the option FORCE but without specifying which are the X-Y sizes of the new window through the key CVKEY_SETSIZE, the system must choose automatically the dimensions to use. In these cases, Nanodesktop uses two internal system variables, called **HGUI_DefaultLenX** and **HGUI_DefaultLenY**. The values of these variables can be changed using **cvChangeDefaultWindowDim**.

Using this routine, the user can set the dimensions of the next window that will be opened.

The routine doesn't accept values for NewLenX or NewLenY that are lesser than 64. If you'll try to enter a value that doesn't respond to this requisite, the routine will generate an error NDHGUI_WRONG_WINDOW_DEFAULT_LENXY

cvSetDefaultColorWnd

This routine changes the colors automatically used by the system when it creates a new OpenCV window.

```
void cvSetDefaultColorWnd (TypeColor ColorTitle, TypeColor ColorBGTitle,
                           TypeColor ColorBGWS,  TypeColor ColorBorder)
```

cvSetDefaultColorTitle

This routine changes the colors of the title that is automatically used by the system when it creates a new OpenCV window.

```
void cvSetDefaultColorTitle (TypeColor ColorTitle)
```

cvSetDefaultColorBGTitle

This routine changes the colors of the title background that is automatically used by the system when it creates a new OpenCV window.

```
void cvSetDefaultColorBGTitle (TypeColor ColorBGTitle)
```

cvSetDefaultColorBGWS

This routine changes the color of the window background that is automatically used by the system when it creates a new OpenCV window.

```
void cvSetDefaultColorBGWS (TypeColor ColorBGWS)
```

cvSetDefaultColorBorder

This routine changes the color of the border that is automatically used by the system when it creates a new OpenCV window.

```
void cvSetDefaultColorBorder (TypeColor ColorBorder)
```

cvSetNextColorWnd

This routine sets a new custom set of colors for the next window that will be created.

```
void cvSetNextColorWnd (TypeColor ColorTitle, TypeColor ColorBGTitle,
                        TypeColor ColorBGWS, TypeColor ColorBorder)
```

These settings will override the normal settings defined by cvSetDefaultColorWnd, but only for the *first* window that will be created after that cvSetNextColorWnd has been called. The further windows will return to be created in base of the default colors.