

The NTK Project
for
xHarbour

GETTING STARTED

©2001-2006 - Jn Dechereux.

Summary

I	Introduction.	5
0.1	'The NTK Project', what is it exactly?	7
0.2	Who are people concerned by The NTK Project?	10
II	The NTK Project's architecture.	11
0.3	The NTKCore layer.	15
0.4	The NTKDll layer.	16
0.5	The NTKRad layer.	18
III	Prerequisite.	19
0.6	Step 1 - Borland CPP: Setup and Configuration.	22
0.7	Step 2 - xHarbour: Setup and Configuration.	23
0.8	Step 3 - NTK Project: Setup and Configuration.	24
0.8.1	- XHRBENV.BAT the NTK Project's Var environment.	24
IV	How to make a NTK Project application.	27
0.9	- MK.BAT the NTKCore's Makefile.	29
0.10	- MKRAD.BAT the NTKRad's MakeFile.	29
0.11	- Creating my own custom MakeFile using MAKE.BAT.	30
V	How to create my first NTK application.	31
0.12	Using NTKRad way of programming.	33
0.13	Using NTKCore 'traditional manner' of programming.	33

VI	Where can I find NTK Project documentation.	35
-----------	--	-----------

VII	Acknowledgments.	39
------------	-------------------------	-----------

VIII	Special notes of the author.	43
-------------	-------------------------------------	-----------

Part I
Introduction.

0.1 'The NTK Project', what is it exactly?

The NTK (New Technology Kit) **Project**, is a professional framework designed for [x]Harbour, providing a set of tools that allow xBase developers (Clipper, [x]harbour, clip-4-Win and others) to create modern Win32 GUI programs.

NTK Project propose two main approaches of programming MS-Windows applications:

- The first method consists in using NTK's built-in RAD (Rapid Application Development) API, which is the more modern, the more easy and rapid way to create robust apps. From the syntax point of view, it is also more xBase like...

- The second technic is the CORE approach, also called 'Traditional manner', because it lets the developer free to directly access to the Win32's API functions by handling and processing messages, as it uses to be done in low-level languages like C/C++. But don't worry, unlike C, NTK is far more friendly !

In other words, the NTK programmer will mostly use the RAD API to develop efficiently and concentrate its efforts on the business logic, but sometimes will require NTK's CORE API for special or complex tasks.

NTK Project for xHarbour, it's also :

- A powerful DLL call engine (dynamic wrapper) built-in to the CORE layer.
- A syntax backward compatibility with most of Clip-4-Win 16 bit functions.
- A syntax backward compatibility with a lot Clipper-Dos commands and functions.

(e.g. SET COLOR TO R+/W, @ y,x GET ..., @ y,x SAY..., NtkINKEY()¹ or NtkACHOICE() are understood!).

NTK Project for xHarbour will enable you to :

- Easily develop true 32 bit GUI applications for MS-Windows environments, the way you want: Core, Rad or Both...
- Provide fast and powerful GUI features in your [x]Harbour programs without know anything about C/C++ language.

i.e. .32 bit common dialogs (Open/Save/Color/Font/Dir/Folder,...)

.32 bit controls (MaskEdit, Gauge, Listview, Sliders, Tooltips, Treeview, etc)

.Bitmap APIs manipulation (included support of bmp, ico, gif and jpeg formats)

.Tons of functions from different themes (Accelerators, Bit manipulation, Buttons, Caret, Cursor, Clipboard, Dialogs, Drawing,

¹the only restriction is to set the 'Ntk' prefix before each function.

.Files, GDI, INI files and RegEdit, Menus, Keyboard, Mouse, Run and ShellExecute, Disks and Directories, and so on...)

.Lot of Contribs, Core and Rad samples...

- Keep on using your favorite xBase RDBMS knowledge.
- Migrate the major part of you old Dos-Clipper apps with minor code changes.
- Port a lot of your old Win16 Clip-4-Win code with less changes, saving the essential of the main business logic.
- Access MS-Windows APIs via NTK's DLL call engine without having to write a single line of C code.
- And far more...

Note:

NTK Project supports a third alternative way of programming, known as 'data driven' application development. That consists to detach the interface code from the engine code.

The program's interface (named 'Resource Dialog') is designed externally from a visual resource editor like Borland Workshop, MS-Visual studio, etc.. then, the resulting .RES file is linked with the program itself, which one, simply calls something like :

```
NTK_DialogBox( hInst , "SAMPLE_DLG" , hWndd )
```

to connect the Dialog to the business logic code during runtime.

The advantage of such a technic is implicitly understandable. Besides, it is a very speed way to develop and maintain screen user interfaces.

Even, some developers (keyboard fanatics) don't use the Workshop at all! As they usually got a pretty good knowledge of the RC syntaxe language, they enjoy to code their screen interfaces directly inside the .RC files and then produce the .RES invoking manually the RC compiler.

But, from another point of view, it appears several pitfalls in which some of us fell into (at least once) when we began MS-Window programming:

- First, The main trap is that EDITS created inside a MS-Dialog doesn't have any ability to be controlled by a picture clause, like GETS are in the xBase world. Further more, keyboard navigation inside the Dialog (from an Edit to another) is a real nightmare! MS does not offer anything more than basic TAB (to go forward) and SHIFT-TAB (to go backward).

Can you imagine a P.O.S (point of sale) application based upon such a GUI user interface?

- Second, From my point of view, an other major drawback is that MS-Windows Dialogs never let you know when user struck a special key like F1 to F12, PgUp, PgDn, etc... This can only be done with extra code intercepting WM_GETDLGCODE message. This is a real waste of time! What you gain in one side, you lose it on the other side. Where is the bargain?

So, to be objective regarding to MS-Dialogs, it is reasonable to say they are productive if you remain in MS imposed limits. In other words, handy, but only for creating standard interfaces.

0.2 Who are people concerned by The NTK Project?

As we said in the previous section, these are mainly developers coming from xBase world like :

- Clipper, [x]Harbour or (other text mode) xBase RDBMS people, who are needing or wanting to develop modern 32 bit-GUI applications or to migrate their old MS-Dos ones...
- Clip-4-Win² programmers who are wanting to continue programming in a very similar way they did these last ten years (with this magic tool!), but now, in a Win32 environment...
- FiveWin users wanting to experiment a close Win32 GUI under xHarbour...
- HWGUI, MiniGUI or Other open source users deciding to combine the best of their favorite library, both with the power of NTKCore and NTKDll layers...

But not only, **here are some other good reasons to use NTK Project for xHarbour:**

- If you're fed up with slowness of bloated classic programming environments of the actual market, usually requiring tons of (always missing) DLLs, without speaking of those that take 600 MB of free disk space just to install, ...
- If you're tired of cool modern languages that could be great if only they were supplied with a true compiler! Runtimes can be handy sometimes, but are definitively not compatible with speed execution nor with easy and low-time deployment., ...
- If you want to make faster apps, but you don't want, or have no time to learn C,...
- If you need 32 bit GUI applications able to have very short access times on (single or network) large databases, ...
- If your are looking for a low-cost 'per user' license network RDBMS,...
- If your are looking for a productive, high speed, and handy tool capable to let you manipulate small or large databases and generate powerful GUI interfaces in very few lines of code, ...
- If, if, ... STOP!

Do not hesitate anymore, dive with us into the fabulous world of **NTK Project for xHarbour!** You will quickly be able to offer your clients, very fast compiled (unique EXE) networking applications with no license fee³ per user.

²At start, late 2001, NTK was initiated as an internal project to port our numerous and huge commercial applications from Clip4Win 16 bit to Win32/xHarbour.

³As Clipper was in an older time, today, xHarbour is also unlimited in terms of networking 'per user' license, unlike many (expensive) other products!

Part II

The NTK Project's architecture.

NTK project is a third party library formed of sub-libraries which ones are themselves composed of layers. There's only two major kind of sub-libraries and layers: Low-Level and Hi-Level ones.

The main library and master component is NTKCORE.LIB. It contains the three basic layers of NTK Project that are: NTKCore, NTKDLLs and NTKRad.

NTKCORE.LIB gives the opportunity to create Win32 programs using two main ways: R.A.D development (very easy) or Core development (more powerful and close to C).

As we can see looking at the following sketch, NTK Project is supplied with other 'high level' layer libraries, like:

- NTKBRWSE.LIB, which is a GUI implementation and adaptation of the TBrowse CLASS object.
- NTKEXRAD.LIB is the official EXTended RAD layer of NTKRad, providing many other high Level functions treating various themes:
 - .DBF and Network manipulations
 - .Alert, aChoice and other Dialogs
 - .Calling and running external apps
 - .Etc, etc...
- AND so on...

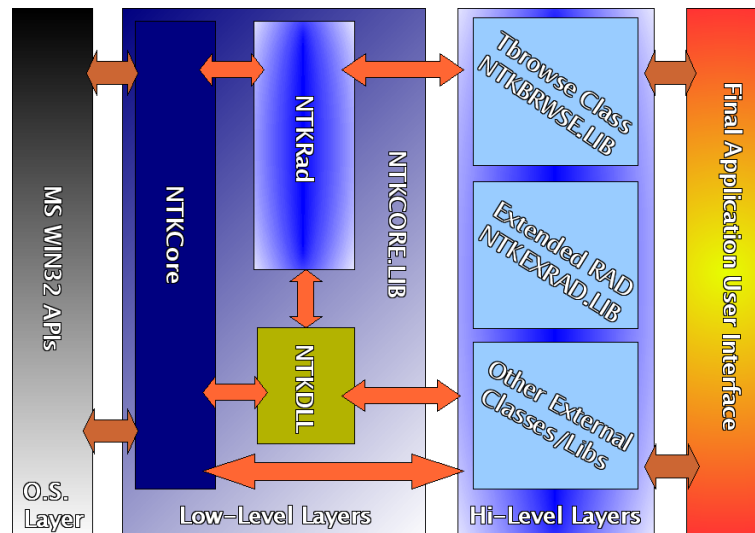


Figure 1: NTK's Architecture Diagram

Now, it also becomes clear that NTK Project is open enough and can support external user defined libraries via the contribution way...

0.3 The NTKCore layer.

NTKCore is the 'low-level' engine of NTK Project. We use to call it 'The Magic Blue Box'!
All other (actual and future) layers and libraries remain on it and are (and will be) architected around it.

NTKCore layer is a set of low level functions allowing to perform direct calls to the Win32 API. NTKCore, it's about 400 functions that are same or very close to original provide by the Win32 API. Some have been slightly adapted to be more friendly or more xBase compliant...

e.g.

In its extreme generosity, MS Win32 API provides us the following C function:

```

BOOL GetMessage( LPMSG lpMsg,           // address of the MSG structure.
                 HWND hWnd,           // handle of window
                 UINT wParamFilterMin, // first message to filter
                 UINT wParamFilterMax // last message to filter
                );                    // The Return value is boolean: 0 or 1

```

In NTKCore, it gives something like this:

```

NTK_GetMessage( aMsg      , ; // array of 7 items each one represents a MSG member
               hWnd      , ; // numeric: handle of window
               nMsgFilterMin, ; // numeric: first message to be filtered
               nMsgFilterMax ; // numeric: last message to be filtered
               )          // The Return value is logic: .F. or .T.

```

In other words, with NTKCORE, the developer (you) will be able to build 32 bits GUI applications using the MS-Windows 'traditional' manner. That is to say : events or messages handling and processing. By the way, you will provide fast and powerful GUI features to your [x]Harbour programs without know anything about C/C++ language and its constraints.

So, just by adding NTKCORE.LIB to your application MAKEFILE, the magic world of Win32 APIs will be yours!

Though, practice of this kind of programming is beyond the scope of this manual, feel free to learn more about the MS Win32 API programming, reading the famous 'Petzold book'. It's really worth while having a look to it. It certainly would be a great and serious benefit!

0.4 The NTKDll layer.

This layer has been especially created to offer the possibility to reach directly from a NTK/xHarbour program, most functions⁴ of the Win32 API that would not be included in NTKCore, or functions that would be stored into other OEM manufacturer DLL.

Thus, if the wished function is embedded inside an external DLL, the developer can wrap it dynamically with NTKDll's call engine. This can be done both ways:

- Using the combination of NTKcore's functions (NTK_LoadLibrary, NTK_GetProcAddress NTK_CallDll and NTK_FreeLibrary).

Or more easily

- Using the NTKRad's command (DECLARE FUNCTION Myfunc LIB MyDllName.Myfunc ...).

e.g. #1

```

/*
*** ShellAbout() WRAPPER using the RAD command: Declare function or _DLL function.
*** Usage      : ShellAbout( hWnd, cTitleBarText, cInboxText, hIcon ) -> nRet
*** Returns    : A numeric value. 1=Success OR 0=fail.
*** Included:  In Shell32.DLL
*/

// ----- NTK declaration style...
Declare function ShellAbout Lib Shell32.ShellAboutA ;
                ( hWnd As HWND, cApp As string,;
                  cOtherStuff As string, hIcon As HICON ) AS int

// The following syntax works same as the above. Do not hesitate to try it !
// It has been created for NTK's backward compatibility with 16 bit Clip4Win apps.
// Just Look at NTKDLL.CH for further details...

// ----- C4W like declaration style... For Backward compatibility!
// _DLL function ShellAbout(hWnd AS HWND ;
//                          cApp AS string,;
//                          cOtherStuff AS string,;
//                          hIcon AS HICON) AS int Pascal:Shell32.ShellAboutA

```

e.g. #2 Have a look to CRW32.PRG in the contribs folder of NTK Project. You will see a fully functional set of wrapped functions for the Crystal Report API printing engine !

⁴Complex C++ functions or classes cannot be wrapped with NTKDLL. This will certainly require writing of some extra C code to be done.

0.5 The NTKRad layer.

Based upon NTKCore layer, NTKRad is brought to the xBase developer, with the main aim to hide the hard stuff that is handling and processing messages when you're programming MS-Windows 'the traditional' manner.

So, developing applications using NTKRad is just kidding for an xBase programmer, because the most common xbase commands and functions are available for GUI mode. In fact, developer only have to take care about two important things.

- First, keep in mind a NTKRad application always start with at least the minimum following declaration command: `CREATE WINDOW hWnd TITLE "My title..."`

- Second, as it is well known, in a graphical environment, functions or commands like `SAVE SCREEN` or `RESTORE SCREEN` do not work anymore. So, this not absolutely required, but it is strongly recommended to gather all `@ y,x SAY ...` commands and other screen output commands inside a unique place where MS-Windows will come to peek, and then, take order of what he has to do when the current working window need to be painted/repainted.

To do so, NTKRad gives you the opportunity to attach a 'ON PAINT' clause to the `CREATE WINDOW` command. Thus, it becomes possible to write something like this:

```
CREATE WINDOW hWndDemo      ;
      TITLE cWinTitle      ; // Minimum declaration
      AT 0,0 SIZE 320,200  ;
      ON PAINT DoRePaint()
```

As you can suppose it, the `DoRePaint()` procedure/function will have to receive all the `@ y,x SAY ...` and all other screen output orders. Then, NTKRad does ALL THE JOB FOR YOU and tells MS-Windows what it has to do, each time a portion or the whole window context need to be refreshed!

Part III
Prerequisite.

At this stage **NTK project** is only available for **xHarbour** using the latest version of **Borland CPP**. So, before to build our first NTK GUI program, it is necessary to correctly configure the programming environment. To do so, the following sections propose a quick and useful help guide.

To be clear and accurate:

- NTK Project is a third party library working for and with xHarbour RDBMS.
- NTK Project and xHarbour require a C compiler to produce executable applications.
- Without a GUI library like 'The NTK Project', xHarbour can only create Win32 Console (Text mode) executables.
- Actually, NTK Project doesn't support other C compiler than Borland CPP.

So, those three products have to be downloaded, installed and correctly configured to work properly together, as expected.

Note:

As said in the above, Borland CPP compiler is required. DON'T WORRY - Using The NTK Project, you'll never have to deal with it. The C compiler is just invoked in a transparent way, during compilation and linkage phases, to transform all your .PRG files into a unique .EXE (executable) file.

For further details, have look at the '**Creating my own custom MakeFile for a specific NTK application**' section.

0.6 Step 1 - Borland CPP: Setup and Configuration.

Borland CPP 5.5.1 is actually freely available at

<http://edn.embarcadero.com/article/20633>

- 1 - Download it.
- 2 - Install it.
- 3 - Just enter 'C:\BCC55'⁵ for main installation folder.
- 4 - Now, the last thing to do is to check correctly the two configuration files BCC32.CFG and ILINK32.CFG. Both are stored in the BIN subfolder.

If, as shown above, you choose '\BCC55' for installation folder, the respective content of each of those files should look like as follow:

```
[Content of BCC32.CFG]
-I"c:\Bcc55\include"
-L"c:\Bcc55\lib;c:\Bcc55\lib\psdk"
-lj"c:\Bcc55\lib;c:\Bcc55\lib\psdk"
```

```
[Content of ILINK32.CFG]
-L"c:\Bcc55\lib;c:\Bcc55\lib\psdk"
```

So, if you experiment compilation or linkage problems, feel free to add or to adapt your CFG files according to your needs.

⁵Of course, the 'C:' expression designates the default installation volume, but you might choose 'D:'. In such a case, you have to replace all 'C:' strings of .CFG files with 'D:'.

0.7 Step 2 - xHarbour: Setup and Configuration.

If you are not comfortable nor experimented with compiling xHarbour sources, and over all, to avoid useless waste of time, we recommend you to directly download and install xHarbour binaries.

xHarbour is an open source product, you can download it at xharbour.org, from

http://www.xharbour.org/index.asp?page=download/windows/binaries_win

Just be aware to select ”**xHarbour Binaries for Borland C++ 5.5.1**”.

or directly at sourceforge, from:

<http://prdownloads.sourceforge.net/xharbour/xharbour-0.99.60.bin.w32.bcc32.zip?>

1-Download it.

2-Run Setup and select C:\XHARBOUR for main folder installation.

3-OK, that’s all for xHarbour installation. Now go to STEP 3.

0.8 Step 3 - NTK Project: Setup and Configuration.

- 1-Download it.
- 2-Run Setup and keep the default selection to C:\WNTKCORE as main folder installation.
- 3-OK, that's all for NTK installation. Now look at XHRBENV.BAT settings...

0.8.1 - XHRBENV.BAT the NTK Project's Var environment.

XHRBENV.BAT is mainly provide for the developer's convenience. It is located in C:\WNTK4HRB\main folder. It is only used to initialize and correctly set all memory environment variables needed by NTK and xHarbour, during a programming session. And, for this reason, it has to be launched from the Shell-console, at start of each new development session.

But before using XHRBENV.BAT, you'll have to configure it correctly once for all!

The first setting you'll have to do, is to make MS-Windows able to auto-start XHRBENV.BAT whatever directory you're working in. To do so, you're facing a binary choice:

- a) Either, you copy XHRBENV.BAT into your favorite and usual batch directory, which one, should already be declared into the MS-Windows PATH environment var!
- b) Or, directly, you add the following string ';C:\WNTKCORE\' to the existing MS-Windows PATH environment var.

(Remember: Right click on MyComputer - Advanced - Environment vars - Click 'Path var' item, then select EDIT button.)

The second and ultimate required setting is to check inside XHRBENV.BAT, if all parameters are good enough, according to the previous installations made in previous steps.

Here is how to proceed:

- 1- From Explorer or from a running Shell command, move to C:\WNTK4HRB\folder.
- 2- With Notpad or your favorite text editor, open/Modify XHRBENV.BAT.
- 3- Now, here is approximately what you can see, and what you are supposed to verify and to adapt or not:

[Content of XHRBENV.BAT]

```
REM
REM --- MINIMAL ENVIRONMENT TO USE NTK FOR xHARBOUR WITH BCC 5.5.1
REM --- YOU CERTAINLY HAVE TO MODIFY IT FOR YOUR OWN CONVENIENCE...
REM
REM --- THEREFORE , WE'RE ASSUMING EITHER YOUR FAVORITE BATCH FOLDER IS
REM --- ALREADY KNOWN FROM %PATH%
REM --- OR C:\WNTK4HRB\ HAS BEEN DECLARED INTO %PATH%
```

REM

```
rem PATH%PATH%;C:\BATCH;C:\TOOLS;C:\BCC55\BIN;C:\BCC55\INCLUDE;C:\BCC55\LIB;C:\XHARBOUR\BIN;C:\BCC55\LIB;C:\XHARBOUR\BIN;c:\DJGPP\BIN
SET OBJ=C:\XHARBOUR\B32\OBJ
SET LIB=C:\BCC55\LIB;C:\XHARBOUR\LIB;
SET INCLUDE=C:\BCC55\INCLUDE;C:\XHARBOUR\INCLUDE
```

```
IF EXIST C:\DJGPP\DJGPP.ENV SET DJGPP=C:\DJGPP\DJGPP.ENV
IF NOT EXIST C:\DJGPP\DJGPP.ENV SET Bison_Simple=C:\DJGPP\share\bison\bison.simple
```

```
SET HB_ARCHITECTURE=w32
SET HB_COMPILER=bcc32
```

```
SET C_USR=
SET L_USR=
```

```
SET BCCDIR=C:\BCC55
SET HRBDIR=C:\XHARBOUR
SET HB_ARCHITECTURE=w32
SET HB_COMPILER=bcc32
rem SET BISON_SIMPLE=\BCC55\BIN\BISON.SIM
```


Part IV

How to make a NTK Project application.

Making a NTK Project application can be done three different ways, depending on what kind of application you're developing.

- If you're creating a pure NTKCore application composed of a single PRG module,
Use: MK.BAT
- If you're creating a NTKRad application composed of a single PRG module,
Use: MKRAD.BAT
- If you're creating a bigger application (mixing both NTKcore and NTKrad commands and functions) composed of a several PRG modules,
Use: your own specific Makfile derivate from our MAKE.BAT template.

0.9 - MK.BAT the NTKCore's Makefile.

MK.BAT is an fast and handy builder, specially designed to create NTKCore applications composed of an unique PRG module.

For example, all demonstration PRGs supplied in \WNTKHRB\NTKCORE can be built using the following syntax: **MK appname** (remember: do not add .prg extension)

To learn more about MK.BAT, do not hesitate to edit it from NTK project's main directory (\WNTK4HRB) and a have a look to its architecture...

0.10 - MKRAD.BAT the NTKRad's MakeFile.

MKRAD.BAT is an fast and handy builder, specially designed to create NTKRad applications composed of an unique PRG module.

For example, all demonstration PRGs supplied in \WNTKHRB\NTKRAD can be built using the following syntax: **MKRAD appname** (remember: do not add .prg extension)

To learn more about MKRAD.BAT, do not hesitate to edit it from NTK project's main directory (\WNTK4HRB) and a have a look to its architecture...

0.11 - Creating my own custom MakeFile using MAKE.BAT.

If you want to develop applications composed of more than one unique PRG module and/or requiring some extra libraries to be linked with, you'll have to make your own specific MAKE file.

To do so, we provide **MAKE.BAT** , which is an handy template builder. Use it in three steps:

Step 1 - Copy it into your application folder.

Step 2 - Edit, customize and adapt it, according to your application modules, executable name, needed libraries, etc...

Step 3 - Just type MAKE at shell prompt. Voila, your app is now created!

Part V

How to create my first NTK application.

0.12 Using NTKRad way of programming.

To understand basic concepts of programming a NTKRad application, we invite you to move into \WNTK4HRB\NTKRAD directory, and first, edit and look carefully at HELLO.PRG. Then, you can continue with HELLO1, HELLO2 and HELLORAD.PRG...

Remember, to build a NTKRad program, just type **MKRAD appname** at shell prompt.

To dive more in deep with NTKRad programming, we recommend you to explore bigger applications such as FANCYGETS, APPDEMO or SAVE'n'RESTORE. These three programs are samples of business applications composed of multiple modules. The source code of the PRG modules has been intentionally over doc'ed to provide you some good tutorials, able to help you to figure out easily how work NTKRad fundamentals. Thus, you will be quickly productive and capable to write your own applications.

Note:

Do not hesitate to dig supplied contributions stored into \WNTK4HRB\CONTRIB directory. Though, they are provided as it, without guarantee of any kind ('use it at your own risk' they said...), they're often very good technical supports too!

0.13 Using NTKCore 'traditional manner' of programming.

To get a pretty good overview of the powerful potential of NKCore APIs, we invite you to move into \WNTK4HRB\NTKCORE directory and give a try to all demonstration samples. Looking at HELLO.PRG and HELLOEX.PRG should be a good start...

Note:

This part of NTK project required knowledges about programming MS Windows using the 'traditional manner', that are beyond the scope of this manual. So from now, we're assuming that you've got the sufficient abilities to understand concepts described in the tutorial samples provided in NTKCORE's folder.

If it is not the case and if you're interested by this way of doing things, we strongly recommend you to get the following materials:

- The famous book "PROGRAMMING WINDOWS FIFTH EDITION" from Charles Petzold
- A copy of WIN32 SDK .Hlp file and a Internet access to MSDN ON LINE
- A lot of spare time...

Part VI

**Where can I find NTK Project
documentation.**

NTK Project manuals are localized into \WNTK4HRB\DOC directory.

You will find:

- GSTARTED.PDF - This GETTING STARTED manual
- NTKCORE.HLP - Electronic manual describing the whole NTKCore APIs of NTKCORE.LIB
- NTKRAD.PDF - Electronic manual describing basic NTKRad commands and functions embedded into NTKCORE.LIB
- FLIST.TXT - A quick memo list of all NTKCore APIs resuming function parameters and return value.

Note:

As you can understand, writing accurate documentation for such a big project as NTK is an heavy task and can not be done just blinking. So before you complain for a leak of documentation, here are some little tips that can help you:

- Take a look at FLIST.TXT
- Use GREP tool or Windows Search to seek the wanted function into all PRG samples throught \WNTK4HRB folder and sub-folders. Keep in mind that PRG provided ARE TUTORIALS!
- Use GREP tool or Windows Search to seek the wanted string into all .CH localized into \WNTK4HRB\INCLUDE folder
- If you're looking for information about NTKRad commands, try exploring NTKCMD.CH. A lot of RAD commands are stored inside.
- Post a message on the forum. Maybe someone of NTK's community can point you the right direction.
- Then, only if you really don't find anything, post a message to our technical support.

Part VII

Acknowledgments.

First, I wish to thank all members of my crew (Andre, Sylvain, Marco, Jne, Abbougaga Jr and all other contributors, testers, ... they will recognize themselves.) having helped me and supported during this 5 last years to make possible that NTK Project is now alive!

Then, I would like also to give a special acknowledgment to all members of Harbour and xHarbour development team. Those guys made a really great job!

Part VIII

Special notes of the author.

I have to confess that English is not my mother tongue. So, in case of typo or misspelling, don't be too angry. Just send me a little feedback. Then, i will correct the problem and will try to make things better next time...

You know, i'll consider any comment, correction, suggestion with great pleasure and high kindness. Just be polite! This remark is also available for bug notifications or any other kind of contributions that can improve NTK Project. So, enjoy and have fun using NTK Project for xHarbour!

Sincerely yours,

Jn Dechereux.