# Chapter 7

# TACO for Windows *by W-D.Klotz and A.Götz*

## 7.1 Introduction

TACO for Windows exists since November 1997. It is being used more and more to interface detector systems, OPC based SCADA system and other software running under Windows. The original Windows port was done using the version V5.15 of the DSAPI of TACO. It has been recently updated with the latest versions of all the libraries - DSAPI=V8.29, DBAPI=V6.12, and DSXDR=V5.20. The main difference is that TACO on Windows now supports asynchronism and events. It is therefore fully compatible with the most recent versions of TACO for Windows. A new port of TACO C++ device server library to Windows is on the way and will hopefully be finished before the end of November 2001. This chapter describes the V8.29 port of the C version DSAPI and associated TACO libraries to Windows. It supports writing TACO device servers and clients under Windows 95/98/NT and 2000. A port of the ndbm version of the TACO database server and manager exists but is not available as package. It can be downloaded on request (send an email to `taco@esrf.fr`). The following text describes how to download and install the Windows port of TACO and an example device server.
This document describes how to install and use the device server libraries (libdsapi.lib, libdbapi.lib, libdsxdr.lib) and the ONC RPC (oncrpc.lib) on Windows NT and Windows 95/98. Once installed on your target computer, you can develop TACO device servers and TACO clients written in ANSI C.
The libraries have been developed and tested on Windows 95/98 and Windows NT4.0. With the exception of the ONC RPC library, libraries are created as static libraries. All libraries have been compiled with MS Visual C++ Version 6.0, and are delivered as Release versions.
The libraries are distributed in two packages:

1. Binary, comprising the libraries, header files and a sample client/server application with sources, project- and makefiles.

2. Source, comprising in addition to the binary distribution all source code, that allows you to rebuild the libraries yourself on your target system.

All executables and libraries in this distribution have been compiled as Win32-Release versions. For details on compiler flags etc. you have to look into the corresponding makefiles, i.e. files with the extension .mak.
The current libraries are based on DSAPI revision 8.29

The zip file for the TACO Windows release can be found on our ftp server[1]. Additional archives mentioned below are also available on our ftp server[2]. Additional products that you need The Taco device server libraries are based on SUN's ONC RPC. This RPC has been ported to 32 bit Windows starting from the original source code, which is freely available for UNIX from SUN. The library is named oncrpc.lib and linked into a dynamic link library oncrpc.dll. It is based on the Windows Socket definition from MicroSoft. You can get the tested ONC RPC library from our ftp server. The C/C++ compiler used to build this release was Microsoft Visual C++ version 6.0. For the Text-to-Speech sample server you need Microsoft's Speech SDK and an additional DDE server (TTSApp.exe). Both are available from our ftp server.

## 7.2 Installation

All files are bundled in a single zip archive (16 MB). You have to extract files from the archive. During extraction files and directories will be created relative to the directories /taco/dbase and /taco/dserver. If you do not extract to the root directory of your current disk, you have to change path-specifications in the corresponding makefiles or project files. To avoid this work, it is strongly recommended to install everything in the root directory! Doing so you will get the following directory tree structure:

```
C:\TACO\DBASE
+---res
    +---clnt
    +---svc
    +   +---rtdb
    +---win32
    +   +---Debug
    +   +---Release
    +---include

C:\TACO\DSERVER
+---dev
+   +---classes
+   +   +---main
+   +   +   +---src
+   +   +   +---include
+   +---system
+   +   +---api
+   +   +   +---admin
+   +   +   +   +---include
+   +   +   +---apilib
+   +   +   +   +---include
+   +   +   +   +---src
+   +   +   +   +---win32
+   +   +   +   +   +---Debug
+   +   +   +   +   +---Release
+   +   +   +---cmds_err
+   +   +       +---include
+   +   +       +---res
```

[1] ftp://ftp.esrf.fr/pub/cs/taco/taco_win32_v8_29.zip
[2] ftp://ftp.esrf.fr/pub/cs/taco/dsapiNT/

```
+   +   +       +---src
+   +   +---xdr
+   +   +   +---include
+   +   +   +---src
+   +   +   +---win32
+   +   +       +---Debug
+   +   +       +---Release
+   +   +---dc
+   +   +   +---include
+---include
+---classes
+   +---powersupply
+   +   +---ag
+   +   +   +---include
+   +   +   +---src
+   +   +   +---win32
+   +   +       +---ps_menu
+   +   +       +   +---Release
+   +   +       +---Release
+   +   +---src
+   +   +---include
+   +---TextTalker
+       +---Release
+       +---src
+       +---include
+       +---TextTalker_menu
+           +---Release
+---lib
+   +---win32
+       +---Debug
+       +---Release
```

## 7.3 Binary distribution

If you plan to develop new device servers and client applications only, you should use the binary distribution. To get the binary distribution, you have to extract the following directories with all their subdirectories:

- taco/dserver/classes
- taco/dserver/include
- taco/dserver/lib

  In the directory /taco/dserver/lib/win32/Release you will find the files:

- DSMain.res a resource file that has(!) to be linked with every device server (do not modify it!);
- libdbapi.lib a library to access the static data base;
- libdsapi.lib the main DSAPI library;
- libdsxdr.lib a library with XDR filter routines.
- libtts.lib a library used by the TextTalker sample application.

- oncrpc.lib the ONC RPC import library.
- oncrpc.dll the ONC RPC dll.

In the directory /taco/dserver/lib/win32/Debug you find the same files as WIN32 Debug versions. The directory /taco/dserver/include comprises all .h header files of this release. Building the AGPowersupply sample client/server pair. In the directory dserver/classes/powersupply/ag/win32 you will find

- Agpsds.mak the common makefile , and
- Agpsds.dsw the common project file
- Agpsds.hpj the help project file to genereate Agpsds.hlp.

for the ag-powersupply device server and the ag-powersupply menu client.
Look into the Win32 Release configuration of the makefile if you want to understand how to set compiler flags when you build a server or a client. When you compile the source files on NT or Windows 95/98 you have to define the preprocessor macros _NT, WIN32,_WINDOWS.
In the directories

- taco/dserver/classes/powersupply/ag/win32/Release
- taco/dserver/classes/powersupply/ag/win32/ps_menu/Release

you will find a ready to run server - AGpsds.exe, and a client that knows this server's commands - ps_menu.exe.
Hint: To satisfy the precompiler on my machine, I had to set the /I compiler directive as follows: /I "../include" /I "../../include" /I "/taco/dserver/include" /I/ "/taco/oncrpc/win32/include"
Hint: To satisfy the linker, we had to link with the following libraries:

```
libdsapi.lib libdsxdr.lib libdbapi.lib oncrpc.lib version.lib
wsock32.lib kernel32.lib user32.lib gdi32.lib winspool.lib
comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib
uuid.lib odbc32.lib odbccp32.lib comctl32.lib
```

and set the library path to:

```
/libpath:"/taco/dserver/lib/win32/Release"
```

Hint: Use the find facility to search for the correct location of header files and libraries if you don't succeed to compile and link the samples. And study the makefiles!

### 7.3.1 Building the Text-to-Speech example

In the directory taco/dserver/classes/TextTalker you will find

- TextTalker.mak the common makefile for server and client
- TextTalker.dsw the common project file for server and client
- TextTalker.hpj the help project file to generate TextTalker.hlp.

Look into the Win32 Release configuration of the makefile if you want to understand how to set compiler flags when you build a server or a client. When you compile the source files on NT or Windows 95/98 you have to define the preprocessor macros _NT, WIN32,_WINDOWS.
To link the server we had to set the following linker options:

```
libdsapi.lib libdbapi.lib libdsxdr.lib libTTS.lib oncrpc.lib
version.lib wsock32.lib kernel32.lib user32.lib gdi32.lib
winspool.libcomdlg32.lib advapi32.lib shell32.lib ole32.lib
oleaut32.lib uuid.libodbc32.lib odbccp32.lib comctl32.lib
/subsystem:windows /incremental:no/ /pdb:"$(OUTDIR)/TextTalkerds.pdb"
/debug /machine:I386/ /out:"$(OUTDIR)/TextTalkerds.exe"
/libpath:"/taco/dserver/lib/win32/Release"
```

The server (TextTalkerds.exe) needs a slave server (TTSApp.exe) to run correctly. TTSApp.exe is a DDE server, that receives requests either interactively from it's GUI or through DDE messages. TextTalkerds.exe uses these DDE messages as interface to Microsoft's text to Speech engine. That means that you have to install Microsoft's Speech SDK and the TTSApp.exe server before you can use TextTalkerds.
You find Microsoft's Speech SDK as a self-extracting archive on our ftp server called sdk30s.exe (11,799KB) and the TTSApp project called ttsapp.zip. (166KB)
To start TextTalkerds.exe, you first start manually TTSApp.exe, and then TextTalkerds.exe, or you copy TTSApp.exe into your system PATH. TextTalkerds will launch TTSApp automatically if it is in the system PATH.

## 7.4 Source distribution

If you want to rebuild the libraries, you have to use the source distribution. When you extract the directories

- taco/dbase
- taco/dserver/dev

with all their subdirectories from the archive, you get the source distribution.
Look into the files

- taco/dbase/res/win32/dbapilib.mak
- taco/dserver/dev/system/api/apilib/win32/libdsapi.mak
- taco/dserver/dev/system/xdr/win32/xdrlib.mak

for the makefiles of these libraries.
Look into the files

- taco/dbase/res/win32/dbapilib.dsw
- taco/dserver/dev/system/api/apilib/win32/libdsapi.dsw
- taco/dserver/dev/system/xdr/win32/xdrlib.dsw

for the project files for Microsoft Visual C++ 6.0.
Compiling the libraries is harder. You should first have successfully compiled the sample applications in the binary distribution, before attempting that.

## 7.5 The ONC RPC library

The ONC RPC library is packaged in the same zip file. When you unzip this archive in the root directory of your hard disk you get the following directory structure:

```
C:\TACO\ONCRPC
+---win32
    +---drivers
    +   +---etc
    +---librpc
    +   +---lib
    +       +---Release
    +       +---Debug
    +---rpcgen
    +---rpcinfo
    +---service
    +---test
    +---wintest
    +   +---vers1
    +       +---Release
    +---bin
    +---include
        +---rpc
```

In the project file for Microsoft Visual C++ 6.0[3] oncrpc.lib and oncrpc.dll are found in the Release and Debug directories, respectively. All .h header files are placed in: /oncrpc/win32/include.

You have to copy oncrpc.lib to /taco/dserver/lib/win32/Release to build the Release versions of the DSAPI libraries or sample applications. You also have to copy ./Release/oncrpc.dll to the Windows system directory c:/winnt/system32, before running the Release versions. The same holds for the corresponding files in ./Debug if you want to build and run the Debug versions.

If you want to build the oncrpc library yourself, you have to define the preprocessor macro _X86_ on Intel platforms.

### 7.5.1 Portmapper.exe

Portmapper has to run on your computer before you start any device servers. The makefile and sources for portmapper are in /taco/oncrpc/win32/service. The makefile or project file creates two applications portmap.exe and inst_pm.exe. For W/NT portmapper has to be started as a system service. To register portmapper as system service you use the helper inst_pm. With the ControlPanel/Services utility you can define the statup mode of portmapper. On Windows 95/98 portmap.exe is a different executable!! You have to start portmap.exe on reboot by an entry in 'autoexec.bat' When you rebuild portmap.exe you have to modify the makefile according to your system. Read the first lines of the makefile! You have to set OS either to _NT or _W95.

### 7.5.2 Rpcinfo.exe

rpcinfo.exe is a utility known to UNIX users. It allows you to interrogate portmapper's port tables on you local or any remote host that runs portmapper. In the makefile you have to set OS either to _NT or _W95 if you want to build the executable from scratch.

[3] /taco/oncrpc/win32/librpc/lib/oncrpc.dsw

### 7.5.3 Rpcgen.exe

rpcgen.exe is the RPC IDL compiler. It generates C-stub source code and xdr-filter source code according to your protocol definition in your IDL-file. In the makefile you have to set OS either to _NT or _W95 if you want to build the executable from scratch. Don't mind the many warning messages during compilation, rpcgen.exe works nevertheless!

### 7.5.4 RPC sample programs

cou_svc.exe and do_cou.exe are a server/client pair to test the ONC RPC library. They are both simple console applications. cou_svc.exe in the directory wintest/vers1 is the same RPC server as a Windows application.

## 7.6 Tips and Tricks for developers

### 7.6.1 Printing debug messages

The standard text output I/O library functions printf, fprintf, ? etc do not work on Windows. TextOut(0 is Windows way to display a string in a window's client area at specified coordinates. Apart from that Windows provides only minimal support for text output to the client area of a window. To simplify this problem, the DSAPI library provides a set of functions similar to printf.
The library provides a global integer that can take values between 0 - 4 to describe different debug levels:

```
0     no debug output, like standard printf
1     level adds error messages,
2     level adds trace messages,
3     level adds more details on trace and errors,
4     level adds dumps of data.

extern int giDebugLevel;     // 0 is default
```

The library provides two functions to manipulate this global:

```
extern void SetDebugLevel(int i);
extern int GetDebugLevel();
```

The library provides a replacement function to printf that accepts as the first argument a format string compatible with the formats printf uses, followed by a variable list of arguments:

```
extern void cdecl DbgOut(LPSTR lpFormat, ...);
```

Instead of calling DbgOut directly, you should use one of the following macros in your code for the corresponding debug level as stored in giDebugLevel.

```
#define dprintf                          DbgOut
#define dprintf1 if (giDebugLevel >= 1) DbgOut
#define dprintf2 if (giDebugLevel >= 2) DbgOut
#define dprintf3 if (giDebugLevel >= 3) DbgOut
#define dprintf4 if (giDebugLevel >= 4) DbgOut
```

There is another helpful macro defined in the header file macros.h:

```
#ifdef _NT
:
#ifdef WIN32
#define PRINTF(a)      MessageBox(NULL,a,NULL,MB_OK|MB_ICONASTERISK);
#endif

#else   /* not _NT */
#define PRINTF(a)      printf(a)
:
#endif  /* _NT */
```

To make printf(char *format,?) compatible with Windows, you should use sprintf(buff, char* format,?) first and PRINTF(buff) afterwards instead. This provides the standard printf functionality on UNIX, but pops up a MessageBox on Windows instead.

### 7.6.2 The startup.c file

The developer can assign two function pointers in the server's startup routine. One to perform delayed actions during startup and the other for clean server shut down. Since the server's main calls the startup routine to initialize TACO's RPC services before initializing Windows and creating window handles, you have the possibility to continue the startup after the creation of window classes and main window handles. The function pointer (*DelayedStartup)() will be invoked by the DSAPI library after Windows has finished it's initialization.
If the application uses other Windows services like OLE, DDE or whatever, it has to shut them down in a clean manner. For that purpose the developer can assign the function pointer (*OnShutDown)(), which will be invoked when the main window receives a WM_CLOSE windows message.
Here the definitions in DevServer.h:

```
/* Function called from 'libdsapi' for delayed startup. Useful for
 * Windows applications to perform startup operations when Window's
 * GUI has been initialized. If function pointer is NULL, no delayed
 * startup will take place.
 */
extern long (*DelayedStartup)();
/*
 * Function called from 'libdsapi' for clean shutdown. Useful for
 * Windows applications to perform shutdown operations before the
Window's
 * process is shutdown. If function pointer is NULL, no delayed
 * startup will take place.
 */
extern void (*OnShutDown)();
```

There is the possibility to pass some lines of text to the application's startup. This text will be displayed in the main Windows's backdrop and can be used to inform the user of the server's identity and version.
Here the definition of the corresponding structure in DevServer.h:

```
/* an array of strings to be displayed on the main window backdrop */
typedef struct {
int lines;
char **text;
} MainWndTextDisplay;
extern MainWndTextDisplay gMWndTxtDisplay;
```

If you want text to appear in the main window you have to place something similar like that into the startup routine:

```
:
:
/*
 * Here is the place to define what to put into
 * the main window's backdrop.
 */
static char* info[]= {
{"TACO Server that speaks ASCII text"},
{"32 bit Version rev. 1.0 for Windows 95/98/NT, Oct 2001"},
{"ESRF, BP 220, 38043 Grenoble, France"}
};
:
:
/*
 * Here is the place to assign what to put into
 * the main window's backdrop.
 */
gMWndTxtDisplay.lines= 3;
gMWndTxtDisplay.text= info;
:
:
```

### 7.6.3 Important Window handles

If you want to extend the server's GUI, you need to know the following handles which are declared as globals in NT_debug.h:

```
extern HWND ghWndMain;           // the main window handle
extern char* gszAppName;         // the application's name
extern HINSTANCE ghAppInstance;  // the application's module handle
```

## 7.7 Limitations

The libraries do not provide asynchronous calls nor do they provide calls to the TACO Data Collector nor device servers in C++. The ONC RPC library has been tested at it's best, but one never knows.. If you encounter any bug, try to fix it, and please let me know it! The device server comprises two threads now. A main thread that handles GUI- and Window- events, and a worker thread, that runs the svc_run() function, i.e. dispatches all RPC requests. There is no thread synchronization for the time being. Therefore, if you call RPC-service routines from the main thread, for example as a result of an interactive user input via the GUI, you may run into troubles. Closing remarks In Windows jargon, the sample client application ps_menu is a so called console application. That means that the MFC Framework supplies its own WinMain function, upon which you have no influence what so ever. Apparently that does not conflict with the fact, that the DSAPI library contains also a WinMain entry point, i.e. the server's main. We hope (we haven't tested it yet) that this will stay like that, if you write a standard Windows client, i.e. when you provide your own WinMain for your client, or when you write a non console client with the MFC Framework.
The Device Server's WinMain function has been rewritten, and is much cleaner now. It takes note of small differences between W/NT and W/95. With the new

ONC RPC library, we have now better control on the interplay of Windows events and RPC requests. The device server handles all Window events in a main thread, that updates the GUI, whereas a second worker thread handles the svc_run() loop for RPC requests.
Both sample device servers have now their own help support.
The next step will be to support C++ device servers on Windows.
In case of problems or requests/proposals for modifications contact `klotz@esrf.fr` or `götz@esrf.fr`.