

Inside Native Applications

Copyright © 1998 [Mark Russinovich](#)

Last Updated February 8, 1998

Introduction If you have some familiarity with NT's architecture you are probably aware that the API that Win32 applications use isn't the "real" NT API. NT's operating environments, which include POSIX, OS/2 and Win32, talk to their client applications via their own APIs, but talk to NT using the NT "native" API. The native API is mostly undocumented, with only about 25 of its 250 functions described in the Windows NT Device Driver Kit.

What most people don't know, however, is that "native" applications exist on NT that are not clients of any of the operating environments. These programs speak the native NT API and can't use operating environment APIs like Win32. Why would such programs be needed? Any program that must run before the Win32 subsystem is started (around the time the logon box appears) must be a native application. The most visible example of a native application is the "autochk" program that runs chkdsk during the initialization Blue Screen (its the program that prints the "."s on the screen). Naturally, the Win32 operating environment server, CSRSS.EXE (Client-Server Runtime Subsystem), must also be a native application.

In this article I'm going to describe how native applications are built and how they work. I also provide you the source code for an example native application, *Native*, that you can easily install and that will print out a string you specify on the boot-time Blue Screen.

How Does Autochk Get Executed? *Autochk* runs in between the time that NT's boot and system start drivers are loaded, and when paging is turned on. At this point in the boot sequence *Session Manager* (smss.exe) is getting NT's user-mode environment off-the-ground and no other programs are active. The **HKLM\System\CurrentControlSet\Control\Session Manager\BootExecute** value, a MULTI_SZ, contains the names and arguments of programs that are executed by *Session Manager*, and is where *Autochk* is specified. Here is what you'll typically find if you look at this value, where "Autochk" is passed "*" as an argument:

```
Autocheck Autochk *
```

Session Manager looks in the <winnt>\system32 directory for the executables listed in this value. When *Autochk* runs there are no files open so *Autochk* can open any volume in raw-mode, including the boot drive, and manipulate its on-disk data structures. This wouldn't be possible at any later point.

Building Native Applications Microsoft doesn't document it, but the NT DDK *Build* utility knows how to make native applications (and its probably used to compile *Autochk*). You specify information in a SOURCES file that defines the application, the same as would be done for device drivers. However, instead of indicating to *Build* that you want a driver, you tell it you want a native application in the SOURCES file like this:

```
TARGETTYPE=PROGRAM
```

The *Build* utility uses a standard makefile to guide it, \ddk\inc\makefile.def, which looks for a run-time library named nt.lib when compiling native applications. Unfortunately, Microsoft doesn't ship this file with the DDK. However, you can work around this problem by including a line in makefile.def that overrides the selection of nt.lib by specifying Visual C++'s runtime library, msvcr.lib

If you run *Build* under the DDK's "Checked Build" environment it will produce a native application with full debug information under %BASEDIR%\lib\%CPU%\Checked (e.g. c:\ddk\lib\i386\checked\native.exe), and if you invoke it in the "Free Build" environment a release version of the program will end up in %BASEDIR%\lib\%CPU%\Free. These are the same places device driver images are placed by *Build*.

Native applications have ".exe" file extensions but you cannot run them like Win32 .exe's. If you try you'll get the message:

The <Application Name> application cannot be run in Windows NT mode.

Inside a Instead of **winmain** or **main**, the entry point for native applications is **NtProcessStartup**. Also unlike the

Native Application other Win32 entry points, native applications must reach into a data structure passed as its sole parameter to locate command-line arguments.

The majority of a native application's runtime environment is provided by NTDLL.DLL, NT's native API export library. Native applications must create their own heap from which to allocate storage by using **RtlCreateHeap**, a NTDLL function. Memory is allocated from a heap with **RtlAllocateHeap** and freed with **RtlFreeHeap**. If a native application wishes to print something to the screen it must use the function **NtDisplayString**, which will output to the initialization Blue Screen.

Native applications don't simply return from their startup function like Win32 programs, since there is no runtime code to return to. Instead, they must terminate themselves by calling **NtProcessTerminate**.

The NTDLL runtime consists of hundreds of functions that allow native applications to perform file I/O, interact with device drivers, and perform interprocess communications. Unfortunately, as I stated earlier, the vast majority of these functions are undocumented.

An Example Native Application I've created a toy native application that demonstrates how native applications are built and how they work. The program, *Native*, is installed by running the install.bat batch file. The batch file copies native.exe to your <winnt>\system32 directory and adds the following entry to the **BootExecute** Registry value:

```
native Hello world!
```

When you reboot, *Session Manager* executes *Native* after running Autochk. *Native* allocates some heap, locates its command line argument and then prints the argument ("Hello world!") to the Blue Screen, using the functions I described above. If you want it to print something else simply edit the **BootExecute** value under Regedit or Regedt32 and change "Hello world!" to your message.

To uninstall *Native* execute the uninstall.bat batch file. This deletes native.exe from <winnt>\system32 and changes **BootExecute** back to its typical value.

If you want to build *Native* you must have the Windows NT Device Driver Kit. Copy the makefile.def included with *Native's* sources to \ddk\inc and then you can run *Build*.

[Download Native Plus Source \(16KB\)](#)

