Sources 파일에 대해서 - Tip

이번 강의에서는 Sources 파일을 사용하기 위한 규칙과 DDK에서 설명하지 않은 유용한 매크로에 대해서 알아 보도록 하자.

1. BUILD의 제한과 규칙

BUILD는 다음과 같은 규칙을 따라야만 한다.

1. 소스 폴더에서 하나의 DLL이나 라이브러리만이 생성된다. DLL을 생성할 때, DLL과 동일한 이름으로 DLL 내부에서 사용하기 위한 라이브러리(import library)와 다른 DLL에서 사용하기 위한 파일이(export file) 생성된다.
2. 생성된 라이브러리와 DLL을 연결할 목적으로 하나의 소스 파일을 구성하는 파일들이 있다면, 다수의 실행 파일이 생성될 수 있다.
3. DLL을 생성할 때 .def 파일에 있는 LIBRARY 매크로에 서술된 내용과 Sources 파일에 있는 TARGETNAME 매크로에 서술된 내용은 동일해야 한다. 두 매크로에 서술된 값은 대소문자를 구분한다.
4. DLL이 다른 라이브러리와 연결 되어야 한다면, 연결될 라이브러리들은 Sources 파일에 있는 TARGETLIBS 매크로에 선언되어야 한다. 이러한 라이브러리들은 UMLIBS에서 이미 설명한 "₩*₩" 표기를 가지고 선언한다.
5. BUILD 키워드와 "=" 표기 사이에는 공백을 두지 않아야 한다. 예를 들면,

   SOURCES= file.c

   는 사용이 가능하지만, 다음과 같은 경우는 컴파일 에러가 발생한다.

   SOURCES = file.c
6. BUILD는 NMAKE 매크로나 조건 구문을 처리하지 않는다. 따라서 Sources 파일에 있는 BUILD를 위한 키워드는 텍스트 문자로만 구성되어야 한다. 또한 BUILD는 NMAKE 매크로를 참조 할 수 없다.

2. Free 버전에 Debugging 정보 포함

드라이버는 Checked 버전과 Free 버전의 두 가지 형태 중 하나로 컴파일 된다. Checked 버전은 DBG 컴파일러 상수가 1로 설정되어 있기 때문에 #if DBG - #endif 구조로 둘러 쌓인 모든 코드가 소스 코드에 포함된다. 따라서 Checked 버전은 드라이버를 디버깅하기 위한 버전에 해당한다.

그러나 드라이버의 Free 버전은 DBG 값이 0으로 설정되어 있기 때문에 #if DBG - #endif 구조로 둘러 쌓인 코드가 드라이버에 포함되지 않는다. 그러나 Free 버전은 생성된 바이너리 파일에 함수 이름만을 포함한 최소한의 symbolic 정보를 가지고 있다.

Free 버전의 디버깅은 어셈블리 코드를 통해서만 가능하다(처음 내가 소프트아이스라는 디버깅 프로그램을 가지고 드라이버를 테스트 할 때 이러한 현상 때문에 어려운 점이 있었다. 그러나 마이크로소프트에서 제공하는 WinDbg와 같은 디버깅 프로그램의 경우 Free 버전으로 필드 하더라도 환경 설정만 맞으면 소스 레벨로도 디버깅이 가능한다).
그러나 드라이버 파일을 테스트하다 보면 어떤 환경에서는 Checked 버전은 잘 동작하지만 Free 버전에서 문제가 발생하는 경우가 있다. 이런 상황에서는 어셈블리 코드로 디버깅 하기가 어렵기 때문에 Free 버전에 간단한 디버깅 정보를 포함하여 소스 레벨로 디버깅을 하는 것이 때로 유용하다.
Free 버전에 디버깅 정보를 포함하기 위한 매크로를 살펴보기 전에, 먼저 Checked 버전과 Free 버전의 컴파일러 옵션의 차이를 살펴보도록 하자. 각 컴파일러 옵션은 드라이버 생성시 만들어지는 .log 파일을 확인하면 된다. 여기서 제시되는 사항은 [초보자 강의]의 컴파일러 환경 설정에서 사용되는 샘플 드라이버의 빌드 옵션이다.

```
cl -nologo -Ii386₩ -I. -IC:₩DDK₩NTDDK₩inc -IC:₩DDK₩NTDDK₩inc₩ddk -I..
-IC:₩DDK₩NTDDK₩inc -IC:₩DDK₩NTDDK₩inc -IC:₩DDK₩NTDDK₩inc₩ddk -
IC:₩DDK₩NTDDK₩inc₩ddk₩wdm -IC:₩DDK₩NTDDK₩private₩inc -
IC:₩DDK₩NTDDK₩inc -D_X86_=1 -Di386=1 -DSTD_CALL -
DCONDITION_HANDLING=1 -DNT_UP=1 -DNT_INST=0 -DWIN32=100 -
D_NT1X_=100 -DWINNT=1 -D_WIN32_WINNT=0x0500 -DWINVER=0x0500 -
D_WIN32_IE=0x0501 -DWIN32_LEAN_AND_MEAN=1 -DDBG=1 -DDEVL=1 -
DFPO=0 -DNDEBUG -D_DLL=1 /c /Zel /Zp8 /Gy -cbstring /W3 /Gz
/QIfdiv- /QIf /QI0f /GB /Gi- /Gm- /GX- /GR- /GF -Z7 /Od /Oi /Oy- -
FIC:₩DDK₩NTDDK₩inc₩warning.h .₩htsys.c
htsys.c
```

<List A> Checked 버전의 컴파일러 옵션

```
cl -nologo -Ii386₩ -I. -IC:₩DDK₩NTDDK₩inc -IC:₩DDK₩NTDDK₩inc₩ddk -I..
-IC:₩DDK₩NTDDK₩inc -IC:₩DDK₩NTDDK₩inc -IC:₩DDK₩NTDDK₩inc₩ddk -
IC:₩DDK₩NTDDK₩inc₩ddk₩wdm -IC:₩DDK₩NTDDK₩private₩inc -
IC:₩DDK₩NTDDK₩inc -D_X86_=1 -Di386=1 -DSTD_CALL -
DCONDITION_HANDLING=1 -DNT_UP=1 -DNT_INST=0 -DWIN32=100 -
D_NT1X_=100 -DWINNT=1 -D_WIN32_WINNT=0x0500 -DWINVER=0x0500 -
D_WIN32_IE=0x0501 -DWIN32_LEAN_AND_MEAN=1 -DDEVL=1 -DFPO=1 -
DNDEBUG -D_DLL=1 /c /Zel /Zp8 /Gy -cbstring /W3 /Gz /QIfdiv- /QIf /QI0f
/GB /Gi- /Gm- /GX- /GR- /GF -Z7 /Oxs /Oy -
FIC:₩DDK₩NTDDK₩inc₩warning.h .₩htsys.c
```

htsys.c

<List B> Free 버전의 컴파일러 옵션

두 컴파일러 옵션에는 몇 가지 차이가 있다. 먼저 Checked 버전에는 DBG가 1로 설정되어 있지만 Free 버전에는 이 값이 정의 되어 있지 않는다. 두 번째로 드라이버 버전에 따른 컴파일러 옵션의 차이다. Checked 버전에는 컴파일러 옵션이 -Z7 /Od /Oi /Oy-로 설정되어 있으며, Free 버전에는 -Z7 /Oxs /Oy로 설정되어 있다. 각 컴파일러 옵션에 대해서 간단히 알아보자.

i. -Z7

이 옵션은 전체 코드 라인수와 함께 디버거에서 사용하기 위한 Symbolic 디버깅 정보를 가진 .OBJ 파일과 .EXE 파일을 만든다. Symbolic 디버깅 정보에는 함수 및 변수의 이름과 타입, 라인 수가 포함된다.

ii. /Od

이 옵션은 프로그램의 모든 최적화 옵션을 제거하고, 컴파일 속도를 증가시킨다. 이 옵션은 기본이 되는 옵션이며, /Od가 코드의 이동을 제한하기 때문에 디버깅 과정을 간편하게 한다.

iii. /Oi

이 옵션은 어플리케이션이 좀더 빨리 수행되도록 함수의 호출을 원래 가지고 있는 형태나 그렇지 않으면 특별한 형태의 함수 호출로 대체한다. 고유의 함수를 사용하는 프로그램은 함수 호출에 대한 Overhead가 없기 때문에 빠르지만 추가적인 코드를 생성해야 하기 때문에 사이즈는 더 커질 수 있다.

iv. /Oy

이 옵션은 스택을 호출하는데 있어서 프레임 포인터(frame pointer)의 생성을 제한한다. 이 옵션은 프레임 포인터를 설정하거나 제거할 필요가 없기 때문에 함수를 호출하는 것이 더 빠르다. Checked 버전에서는 /Oy-를 사용하여 이 옵션의 기능을 해제한다. 이러한 기능은 아마도 /Od가 동일한 기능을 수행하기 때문에 필요 없을 수 있다.

v. /Oxs

이 옵션은 /Ox와 /Os가 조합된 것으로, 최적화 옵션을 조합하여 가능한 한 빠르게 동작하는 프로그램을 만드는 것이다. 이 옵션은 특히 인라인 함수 확장, 광범위한 최적화, 고유 함수 생성, Fast Code 지원, 프레임 포인터의 생략과 Control Stack 조사를 가능하게 한다. 코드 분할을 가능하게 하는 것은 /Os가 수정하고, Fast Code를 넘어 Small Code Size의 지원과 여분의 코드에 대한 중복은 첫 번째가 아닌 경우 프레임 포인터를 생략을 요청하는 /Oy가 수정한다.

여기서 한가지 더 알아볼 사항은 Linker가 포함하는 정보이다. Checked 버전과 Free 버전의 링크에는 두 가지 차이가 있다. 우선, Checked 버전의 드라이버는 int64.lib, ntoskrnl.lib와 hal.lib의 checked 버전이 링크되고, Free 버전의 드라이버는 동일한 라이브러리의 Free 버전이 링크된다. Ntoskrnl.lib와 hal.lib는 동적 링크 라이브러리로써 단지 일 부분만이 링크될 뿐이며, 따라서 이러한 라이브러리의 외부 참조에 대한 결정은 드라이버가 커널에 로드될 때 이루어진다. 이것은 ntoskrnl과 hal에 있는 외부에서 참조 가능한 인터페이스(exported interface)가 동일하다면, Checked 드라이버와 Free 드라이버가 동일한 Checked 커널이나 Free 커널에 로드 되는 것을 허락한다. 그러나 ntoskrnl과 hal에 있는 외부에서 참조 가능한 인터페이스(import interface)의 구현은 커널의 Free 버전과 비교해서 Checked 버전의 커널과 완전히 다를 수 있다.
링크 과정에서 두 번째 차이점은 빌드 되는 각 타입을 위한 디버그 세팅이다. Checked 버전과 Free 버전은 /DEBUG 링커 플래그를 설정한다. Checked 버전을 위해서는 -debug:notmapped,FULL로 설정되고, Free 버전을 위해서는 -debug:notmapped,MINIMAL로 설정된다. Checked 버전과 Free 버전은 또한 /DebugType 플래그를 설정하며, Checked 버전을 위해서는 -debugtype:both가 설정되고, Free 버전을 위해서는 -debugtype:coff가 설정된다. 여기서 MINIMAL과 COFF 플래그 값은 본질적으로 파일에서 광범위하게 사용되는 Symbolic 정보를 줄인다. 이와 반대로 FULL과 BOTH 옵션은 라인 수, 지역 변수와 다른 디버깅 정보를 제공한다.
DDK 문서에는 BUILD를 사용하여 이미지 파일에 line-number에 대한 정보를 얻는 것이 가능하다고 언급하고 있다. 그러나 이러한 방법은 기본 옵션에 해당하지 않기 때문에 NTDEBUG 환경 변수를 다음과 같이 설정해야 한다 설명하고 있다.

```
set NTDEBUG = ntsd
```

이러한 환경 변수를 설정하는 것은 전체 심볼릭 디버깅 정보를 가진 이미지 파일과 .obj 파일을 생성할 수 있다. 그러나 이러한 설정은 제대로 동작하지 않는다.
NTDEBUG를 Sources 파일에서 “ntsd”로 설정하면, DBG 상수가 설정되고 코드 최적화가 제거된 드라이버의 checked 버전을 얻는다. 대신 Sources 파일에서 NTDEBUG를 “ntsdnodbg”로 설정하면 항상 코드에 대한 최적화를 완료한 Free 버전을 생성할 수 있다.
다음과 같은 코드를 살펴보자.

```
!IF “$(DDKBUILDENV)” != “checked”

NTDEBUG=ntsdnodbg
NTDEBUGTYPE=both
```

```
USE_PDB=1
MSC_OPTIMIZATION=/Odi

!ENDIF
```

독립적인 환경 변수인 DDKBUILDENV를 조사함으로써 Sources 파일은 드라이버의 Checked 이나 Free 버전에 영향을 미칠 수 있도록 조건적으로 플래그를 설정할 수 있다. 또한 NTDEBUGTYPE 플래그는 Checked 버전과 동일한 디버그 형식을 Free 버전의 드라이버에서 적용하고자 할 때 사용한다. 이러한 플래그는 링커가 debugtype을 설정하는데 사용된다.

Sources 파일에서 USE_PDB 플래그를 설정하였다. 만약 이 플래그가 설정되지 않으면, Symbolic 정보는 드라이버의 실행 가능한 이미지 파일에 포함된다. 이것은 결과적으로 동일한 표준이 되는 Free 버전보다 두 세배 정도 큰 실행 가능한 이미지를 생성한다.

※ 출처 : NT Insider 1998년 판 "Build Tricks : Checked and Free Revisited"

http://www.osronline.com/article.cfm?id=68

XP DDK 문서

3. Visual Studio에서 구조체 정보 연결 방법

Visual Studio의 유용한 기능 중 한가지는 파일이나 함수 내부에서 선언된 데이터 구조체를 클릭하면, 이러한 구조제가 정의된 헤더 파일로의 이동이 가능하다는 것이다. 이것은 데이터 구조체가 사용된 부분에서 마우스의 오른쪽 클릭을 하면 나타나는 서브메뉴 중 "Go to Definition of …"의 메뉴를 선택하면 된다.

그러나 현재 여러분이 드라이버를 컴파일 하기 위해 생성한 "MakeFile" 형식의 프로젝트에서는 사용할 수 없다. 어플리케이션을 작성할 때는 Visual Studio에서 기본적인 환경을 설정해 주기 때문에 가능하지만, 드라이버는 Visual Studio에 설정된 정보들을 사용하는 것이 아니라 Sources 파일을 사용하기 때문에 데이터 구조체의 선언 부분을 찾는 것이 불가능 하다.

그러면 왜 이러한 기능이 필요할까? 일반적으로 개발자들이 프로젝트에 사용되는 데이터 구조체를 일관성 있게 작성하거나 프로젝트에 포함되는 헤더 파일이 적을 경우 문제가 덜한 편이다. 하지만, 하나의 프로젝트에 다수의 헤더 파일이 있다고 생각해 보자. 프로젝트를 진행하다 보면 필요에 따라서 데이터 구조를 변경하거나 멤버 변수를 추가하거나 삭제할 가능성이 있다. 이때 선언된 헤더 파일이 소수이거나 수정해야 할 데이터 구조체가 정확히 어떤 헤더 파일에 선언되어 있는지 알고 있다면 문제가 되지 않는다. 그러나 사용되는 헤더 파일이 증가하게 되고 여러 구조체를 사용할 경우, 수정해야 할 데이터 구조체가 어디에 선언되어 있는지 잊어 먹는 경우가 있다. 이러한 상황에서

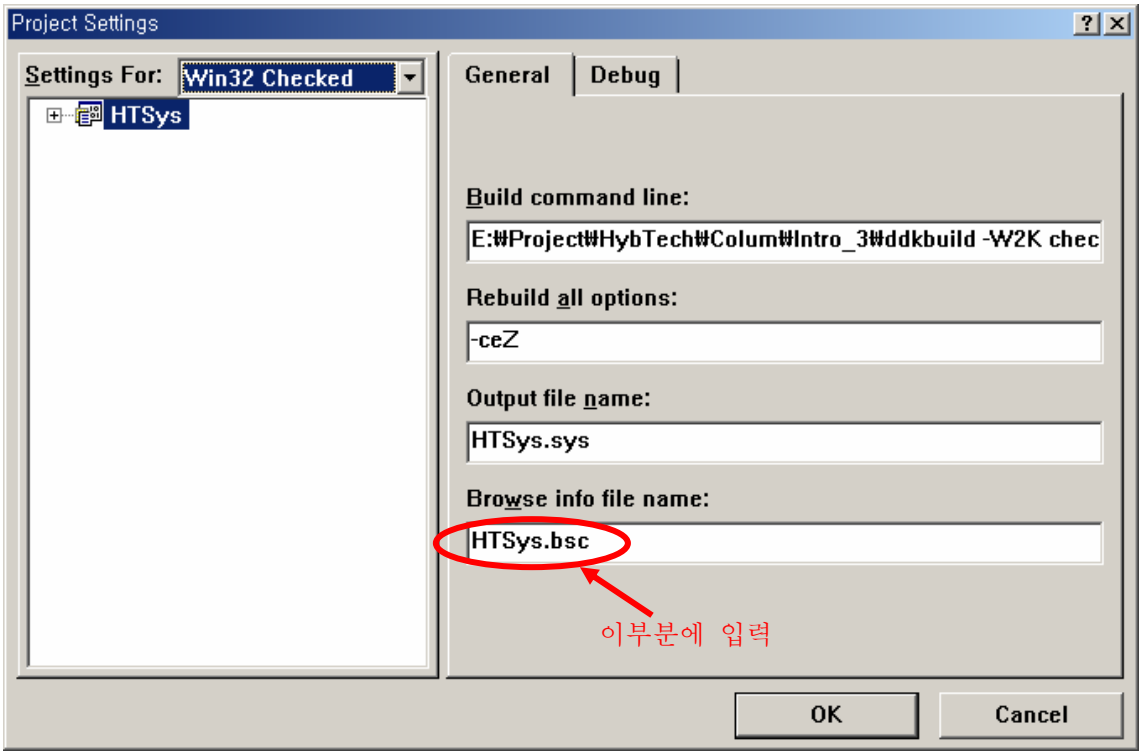
는 직접 데이터 구조체를 검색하여 선언된 헤더 파일을 찾아야 하는 번거로움이 있다. 이러한 번거로움 때문에 데이터의 선언 부분을 바로 찾을 수 있는 기능은 꽤 유용하다. 드라이버를 컴파일 하기 위한 "MakeFile" 프로젝트에서 이러한 기능을 추가시키기 위해서는 몇 가지 작업을 수행해야 한다.

먼저 컴파일 하고자 하는 드라이버의 Sources 파일에 다음과 같은 코드를 추가한다.

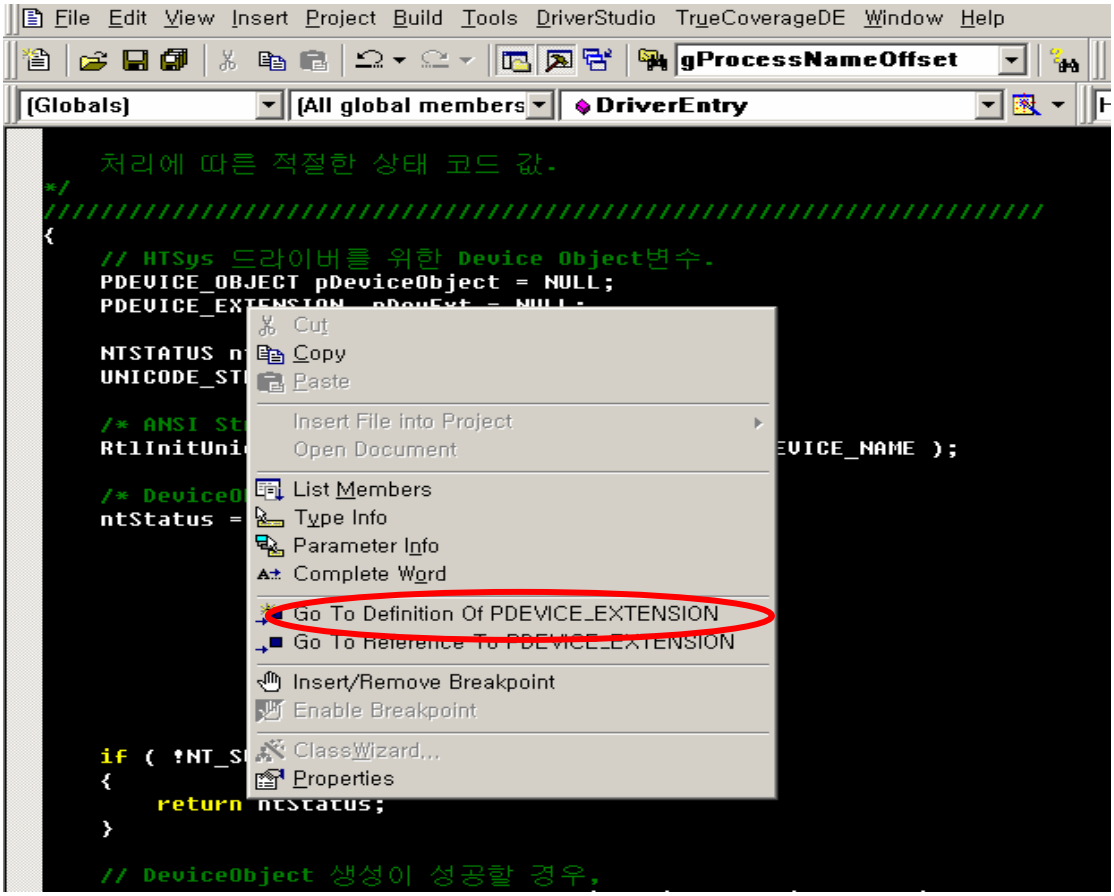
```
TARGETNAME=HTSys
TARGETPATH=obj
TARGETTYPE=DRIVER

INCLUDES=$(BASEDIR)₩inc;$(BASEDIR)₩inc₩ddk;..₩

SOURCES=HTSYS.C

// 새로 추가된 내용이다.
BROWSER_INFO=1
BROWSERFILE=HtSys.bsc -n
```

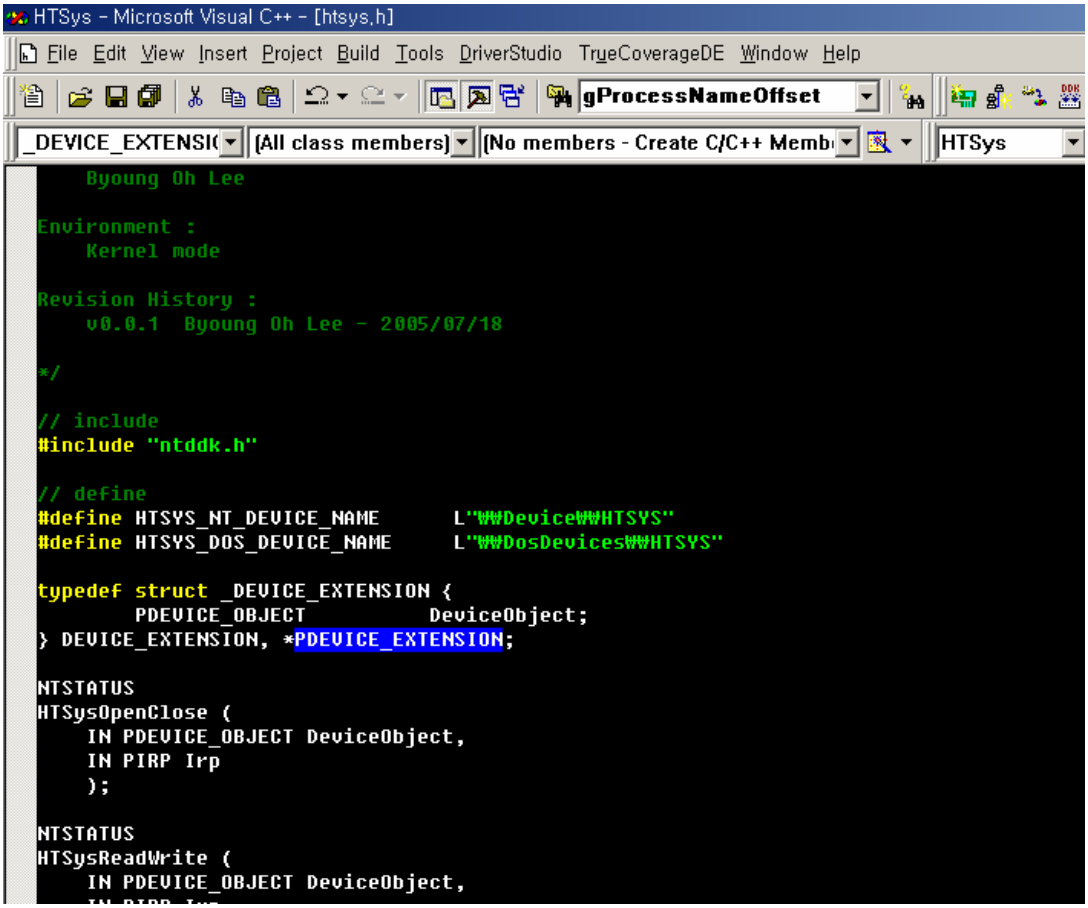
여기서 사용한 Sources 파일은 드라이버의 초보 강의에 해당하는 "컴파일 환경 설정"에서 사용한 샘플 소스에 있는 Sources 파일을 수정하였다.

다음으로 처리할 사항은 "Project->Setting"을 선택하여 나타나는 다이얼로그의 마지막 부분인 "Browse Info file name :"란에 HTSys.bsc 파일이 정의되어 있지 않으면 다음 그림과 같이 정의한다.



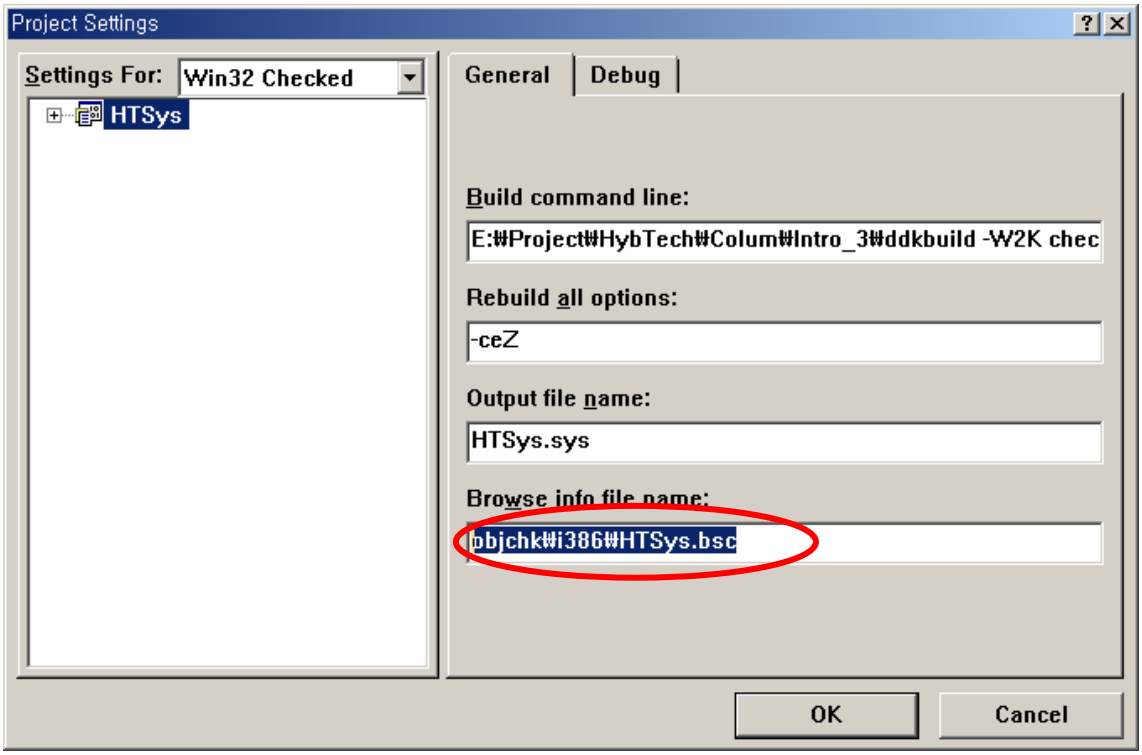
이와 같은 작업을 한 후, 다시 드라이버를 컴파일 하여 데이터 구조체가 사용된 곳에서 마우스를 클릭하여 데이터의 구조체가 선언된 헤더 파일을 쉽게 찾아 갈 수 있다. 아래 그림은 샘플 드라이버인 “HTSys.sys” 드라이버를 수정하여, PDEVICE_EXTENSION 구조체를 선언한 하고 DriverEntry() 함수에서 이 구조체를 사용하여 재 컴파일 하였다. 이후 PDEVICE_EXTENSION 구조체가 사용된 부분에서 마우스 오른쪽 클릭 했을 때, 나타나는 메뉴와 함께, 실제 PDEVICE_EXTENSION 구조체가 선언된 부분으로 이동한 화면이다.


File Edit View Insert Project Build Tools DriverStudio TrueCoverageDE Window Help
gProcessNameOffset
(Globals)
(All global members
DriverEntry
처리에 따른 적절한 상태 코드 값.
*/
{
// HTSys 드라이버를 위한 Device Object변수.
PDEVICE_OBJECT pDeviceObject = NULL;
NTSTATUS n
UNICODE_ST
/* ANSI St
RtlInitUni
/* DeviceO
ntStatus =
if (!NT_S
{
return ntStatus;
}
// DeviceObject 생성이 성공할 경우,
Cut
Copy
Paste
Insert File into Project
Open Document
List Members
Type Info
Parameter Info
Complete Word
Go To Definition Of PDEVICE_EXTENSION
Go To Reference To PDEVICE_EXTENSION
Insert/Remove Breakpoint
Enable Breakpoint
ClassWizard...
Properties



여기서 한가지 알아 두어야 할 사실은 .bsc 파일의 생성은 DDK가 아닌 Visual Studio의 기능이라는 것이다. 따라서 개발자들은 현재 사용하고 있는 Visual Studio에 포함된 bscmake.exe를 사용해야 한다.

위와 같은 매크로를 사용할 때 한가지 더 알아 두어야 할 사실은 생성될 .bsc 파일의 위치를 명확히 기술해야 한다는 것이다. 만약 “MakeFile”로 생성된 프로젝트의 세팅 환경에서 다음 그림과 같이 선언되었다고 하다.



이와 같이 선언되었을 경우 Sources 파일에서 .bsc 파일의 정확한 위치를 정의해야 한다. 만약 정확한 위치가 정의되어 있지 않으면 다음과 같은 다이얼로그가 나타난다.

다음은 Sources 파일 내에서 이를 수정한 것이다.

```
TARGETNAME=HTSys
TARGETPATH=obj
TARGETTYPE=DRIVER

INCLUDES=$(BASEDIR)₩inc;$(BASEDIR)₩inc₩ddk;..₩

SOURCES=HTSYS.C
```

```
BROWSER_INFO=1
BROWSERFILE=objchk₩i386₩HtSys.bsc -n
```

드라이버를 컴파일 할 때 대부분 Free 버전과 Checked 버전에서 컴파일 된 드라이버 이미지 파일은 서로 다른 위치에 놓이게 된다. 이 경우 Free 버전과 Checked 버전에서 .bsc 파일이 서로 다른 위치에 있다면 두 버전을 수정할 때마다 매번 .bsc 파일의 위치를 변경 시켜야 한다. 이러한 불편을 피하기 위해서 마지막으로 수정된 Sources 파일의 내용은 다음과 같다.

```
TARGETNAME=HTSys
TARGETPATH=obj
TARGETTYPE=DRIVER

INCLUDES=$(BASEDIR)₩inc;$(BASEDIR)₩inc₩ddk;..₩

SOURCES=HTSYS.C

BROWSER_INFO=1

!IF "$(DDKBUILDENV)" != "checked"
BROWSERFILE=objfre₩i386₩HtSys.bsc -n
!ELSE
BROWSERFILE=objchk₩i386₩HtSys.bsc -n
!ENDIF
```

위와 같이 Sources 파일을 수정하게 되면, 드라이버의 Free 버전이나 Checked 버전에 상관없이 .bsc 파일을 참조할 수 있다.

※ 출처 : NT Insider 2002년 판 “If You Build It – Visual Studio and Build Revisited”
http://www.osronline.com/article.cfm?id=104

4. Optimize 해제 방법

Windows 2000은 기본적으로 코드에 대한 최적화 기능(Optimization)이 설정되어 있다. 이렇게 코드에 대한 최적화를 실행하는 것은 프로그램 실행 시 좀더 따르게 동작할 수 있도록 하기 위함이다.

코드에 대한 최적화가 설정되어 있으면, 드라이버 코드를 소스 레벨로 디버깅 시 이상한 점을 발견할 수 있다. 분명 소스 레벨에서 한 단계씩 실행을 하면서 디버깅을 시도하지만 현재 디버깅 하고자 하는 위치를 정확히 가리키지 못하거나, 하나의 함수에서 사용되는 지역 변수 등을 확인하지 못하는 경우가 종종 있다. 이러한 현상은 코드에 대한 최적화 옵션이 설정되어 있기 때문이다.
만약 개발자 여러분이 정확하게 소스 레벨로 디버깅을 처리하면서 원하는 동작을 확인하고자 한다면, 컴파일 과정 시 설정된 최적화 옵션을 해제해 주어야 한다.
코드 최적화의 해제는 크게 세가지 방식으로 처리할 수 있다. 먼저, 드라이버의 소스 코드 내에서 설정하는 것이다. 이 방식은 optimize pragma를 사용하는 것으로 소스 코드에서 헤더 파일을 선언한 후에 다음과 같은 형식으로 사용하면 된다.

```
#pragma optimize("", off)
```

두 번째 방법은 드라이버를 컴파일 하기 위해서 사용되는 환경 변수인 "setenv.bat" 파일에서 코드에 대한 최적화 옵션을 해제하는 것이다. 처음 DDK를 설치하게 되면 최적화 옵션이 설정된 상태이기 때문에, 최적화 옵션을 해제하도록 처리한다.
"setenv.bat" 파일을 조사하면 중간 부분에 다음과 같은 코드가 나타날 것이다.

```
:free

rem set up an NT free build environment
set BUILD_ALT_DIR=fre
set NTDBGFILES=1
set NTDEBUG=ntsdnodbg
set NTDEBUGTYPE=windbg
set MSC_OPTIMIZATION=

goto done

:checked

rem set up an NT checked build environment

set BUILD_ALT_DIR=chk
set NTDBGFILES=1
set NTDEBUG=ntsd
```

```
set NTDEBUGTYPE=windbg
set MSC_OPTIMIZATION=/Od /Oi  // 이 부분에서 최적화를 옵션을 해제한다.
```

“setenv.bat” 파일을 조사하면, Free 버전과 Checked 버전 둘 다 “set MSC_OPTIMIZATION=”와 같이 선언되어 있을 것이다. 이 부분에서 코드에 대한 최적화을 해제하기 위한 옵션을 선택하면 된다. 여기서 사용되는 최적화 해제 옵션은 “/Od /Oi”이다.

마지막 방법은 드라이버를 컴파일 하기 위해 반드시 필요한 파일인 Sources 파일을 이용하는 것이다. Sources 파일의 첫 번째 과정에서 말했던 것처럼, Sources 파일에는 드라이버를 컴파일 하기 위한 정보들이 포함되어 있다. 이 파일 내부에서 코드에 대한 최적화 옵션을 해제하면 된다. “setenv.bat” 파일에서 나타난 것처럼 최적화를 설정하는 매크로는 “MSC_OPTIMIZATION” 매크로이다. 따라서 다음과 같이 Sources 파일 내부에 선언하면 된다.

```
MSC_OPTIMIZATION=/Odi
```

여기서 사용된 “/Odi” 옵션은 “/Od /Oi” 옵션과 동일하다.

이것으로 Sources 파일에 대한 강좌를 마무리 한다.