

## 2.1. Nebula Device 엔진의 설치.

시스템 요구 사양

Windows 2000, Windows XP 혹은 Windows Vista

Visual Studio 7 이상의 컴파일러

The DirectX 9.0c SDK

TCL 스크립트 (<http://www.tcl.tk>)

파이썬

엔진 SDK를 다운 받기

다운 받을 수 있는 Nebula 엔진은 크게 두 가지 버전이 있습니다. 하나는 Radonlabs 사의 사이트에서 다운 받을 수 있는 SDK이며, 다른 하나는 소스포지(SourceForge) 사이트의 Nebula 프로젝트 페이지에서 다운 받을 수 있는 버전입니다. 여기에서는 소스포지에서 다운 받은 버전을 기준으로 설명하도록 하겠습니다.

Radonlabs의 버전과 소스포지의 버전은 대부분 동일합니다만 몇 가지 부분에서 차이가 있습니다. 우선 두 빌드 방법은 틀립니다. Radonlabs의 SDK는 Tcl을 사용하는 반면에 커뮤니티 버전은 Python을 사용합니다. 또 커뮤니티 버전에는 외부 개발자가 업로드한 사용자 확장 모듈들이 있지만 Radonlabs사의 SDK에는 이 부분이 포함되어 있지 않습니다. 엔진의 소스 코드는 거의 대부분 동일하지만 조금씩 틀린 부분도 존재합니다.

소스포지 사이트의 Nebula 프로젝트 페이지의 주소는 아래와 같습니다.

<http://sourceforge.net/projects/nebuladevice>

커뮤니티의 Nebula 엔진의 모든 소스 코드는 SVN(Subversion)을 사용해서 다운 받을 수 있습니다. 윈도우즈 사용자의 경우 TortoiseSVN을 사용하면 쉽게 다운받을 수 있습니다. TortoiseSVN은 아래에서 다운 받을 수 있습니다.

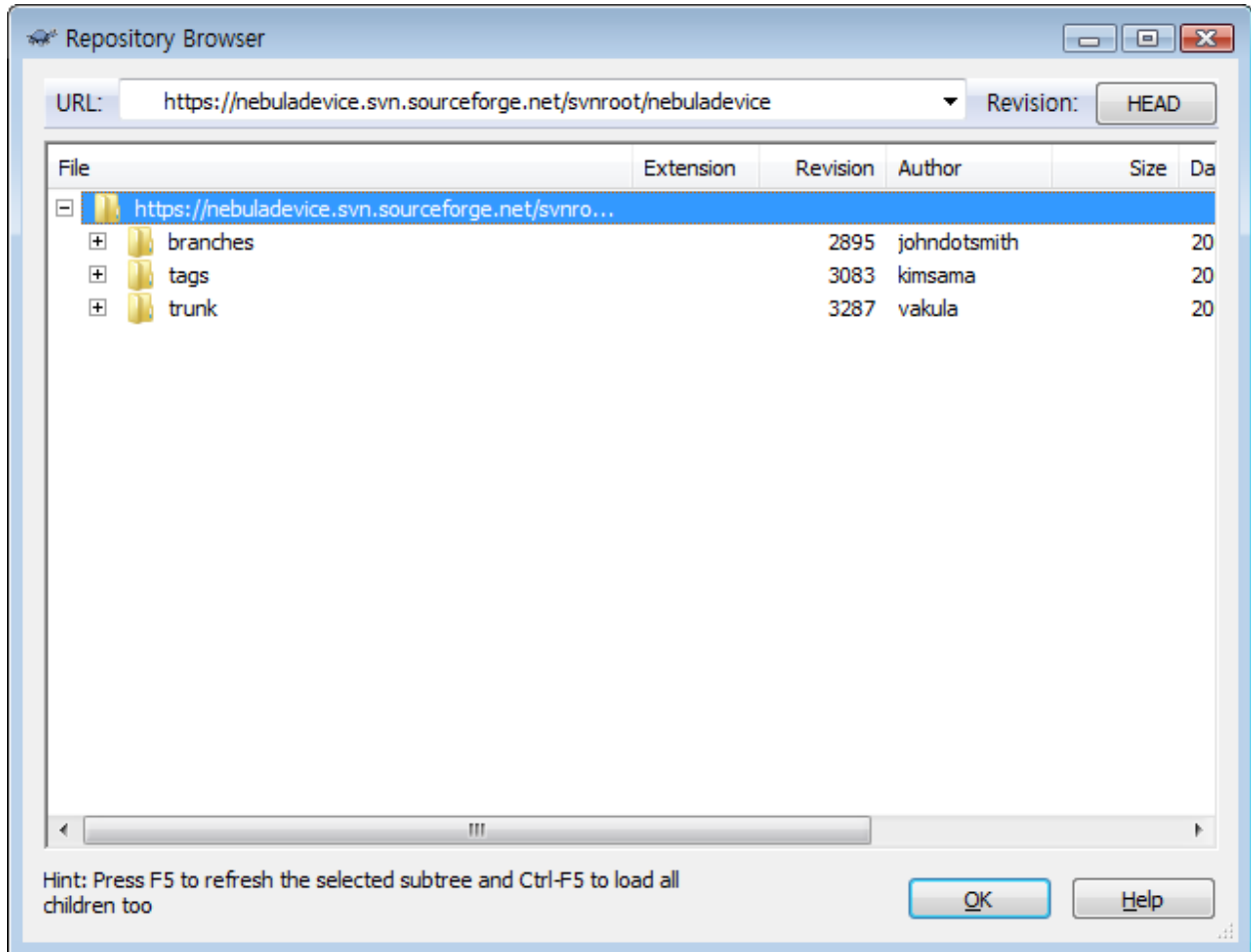
<http://tortoisesvn.tigris.org/>

설치 후에 다운 받고자 하는 폴더에서 오른쪽 마우스를 클릭해서 트랙(Track) 메뉴를 출력한 다음 'TortoiseSVN' > 'Repo-browser' 메뉴 아이템을 선택합니다.

Repository Browser 다이얼로그의 URL에 아래의 주소를 입력합니다.

<https://nebuladevice.svn.sourceforge.net/svnroot/nebuladevice>

SF(SourceForge) 사이트의 Nebula2의 SVN 저장소의 소스 트리(source tree)는 다음과 같습니다.



최신 개발 버전을 다운받고자 하는 경우에는 **trunk**를 다운 받으면 됩니다. 다른 브랜치(branch) 버전은 개발 과정에서 머지(merge)등을 위해 생성한 것들이므로 **trunk**로부터 다운 받을 것을 권합니다.

다운 받은 폴더의 구조

모두 다운 받은 다음 다운 받은 폴더는 다음과 같습니다.

이름	수정한 날짜	유형
appwiz	2008-04-03 오후 2:...	파일 폴더
bin	2008-04-03 오후 2:...	파일 폴더
buildsys3	2008-04-03 오후 2:...	파일 폴더
code	2008-04-03 오후 2:...	파일 폴더
data	2008-04-03 오후 2:...	파일 폴더
doc	2008-04-03 오후 2:...	파일 폴더
export	2008-04-03 오후 2:...	파일 폴더
tools	2008-04-03 오후 2:...	파일 폴더
utils	2008-04-03 오후 2:...	파일 폴더
work	2008-04-03 오후 2:...	파일 폴더
appwizard	2008-04-03 오후 2:...	Python File
build.cfg	2008-04-03 오후 2:...	Python File
readme	2008-04-03 오후 2:...	텍스트 문서
update	2008-04-03 오후 2:...	Python File

이중에서 몇 가지 중요한 폴더를 언급하자면 우선 첫 번째로 'bin' 폴더는 실행 파일들이 있는 폴더입니다. 윈도우즈 환경에서는 이 bin 폴더 내의 win32 폴더 안에 실행 파일들이 위치하게 됩니다. 두 번째로는 'code' 폴더입니다. 이 폴더 안에는 실제로 Nebula 엔진과 Mangalore의 소스 코드와 외부 개발자들이 커밋(commit)한 소스 코드들이 위치하는 'contrib' 폴더가 있습니다. 세 번째는 'export' 폴더입니다. 이 폴더에는 Nebula 응용 프로그램 실행시 필요한 메쉬 및 애니메이션 데이터와 텍스처 파일들이 위치해 있습니다. \$nebuladir 폴더에 위치해 있는 'update.py' 파일은 파이썬 파일로 비주얼 스튜디오의 솔루션 파일과 같이 빌드에 필요한 파일을 생성하기 위한 것입니다.

실제 소스 파일이 있는 code 폴더의 서브 폴더에 대해서는 아래에서 좀 더 자세하게 설명하겠습니다.

- code
  - contrib - Nebula 커뮤니티 멤버들이 커밋한 패키지들에 대한 소스 코드들이 위치한다.
- mangalore - 망갈로 게임 플레이 프레임워크의 소스 코드들이 위치합니다. 2부에서 망갈로를 설명하는 부분에서 언급하도록 하겠습니다.
- nebula2
  - bldfiles
  - inc
    - anim2 -
    - application -
    - audio3 -
    - character -
    - deformers -
    - file -

- gfx2 -
  - gui -
  - input -
  - kernel -
  - locale -
  - loki - , Radonlabs 사의 SDK에는 없음.
  - mathlib -
  - microtcl -
  - misc -
  - nature -
  - network -
  - particle -
  - renderpath -
  - resource -
  - scene -
  - script -
  - shadow2 -
  - signals -
  - sql -
  - tinyxml -
  - toolkit -
  - tools -
  - util -
  - variable -
  - video -
  - xml -
- res - nebula2 응용 프로그램에 사용될 아이콘 파일 등이 위치한다.
  - src - 서브 폴더마다 해당하는 .cc 소스 파일들이 위치한다.

## 엔진 SDK 빌드 하기

Nebula2의 빌드 시스템은 파이썬을 이용한 빌드 시스템으로 플랫폼에 따라 빌드에 필요한 솔루션 파일이나 메이크(make) 파일을 자동으로 생성하는 기능을 제공합니다.

현재 Nebula2의 빌드 시스템에서 지원하는 컴파일러는 다음과 같습니다.

- MS VisualC++ 7.0 이상 (윈도우즈 플랫폼)
- GCC/C++ (Linux 및 Mac OS X 플랫폼)

Nebula2의 빌드 시스템을 실행 시키기 위해서는 시스템에 다음의 패키지들이 설치 되어 있어야 합니다.

- 파이썬(Python) 2.4 혹은 그 이상의 버전, <http://www.python.org>

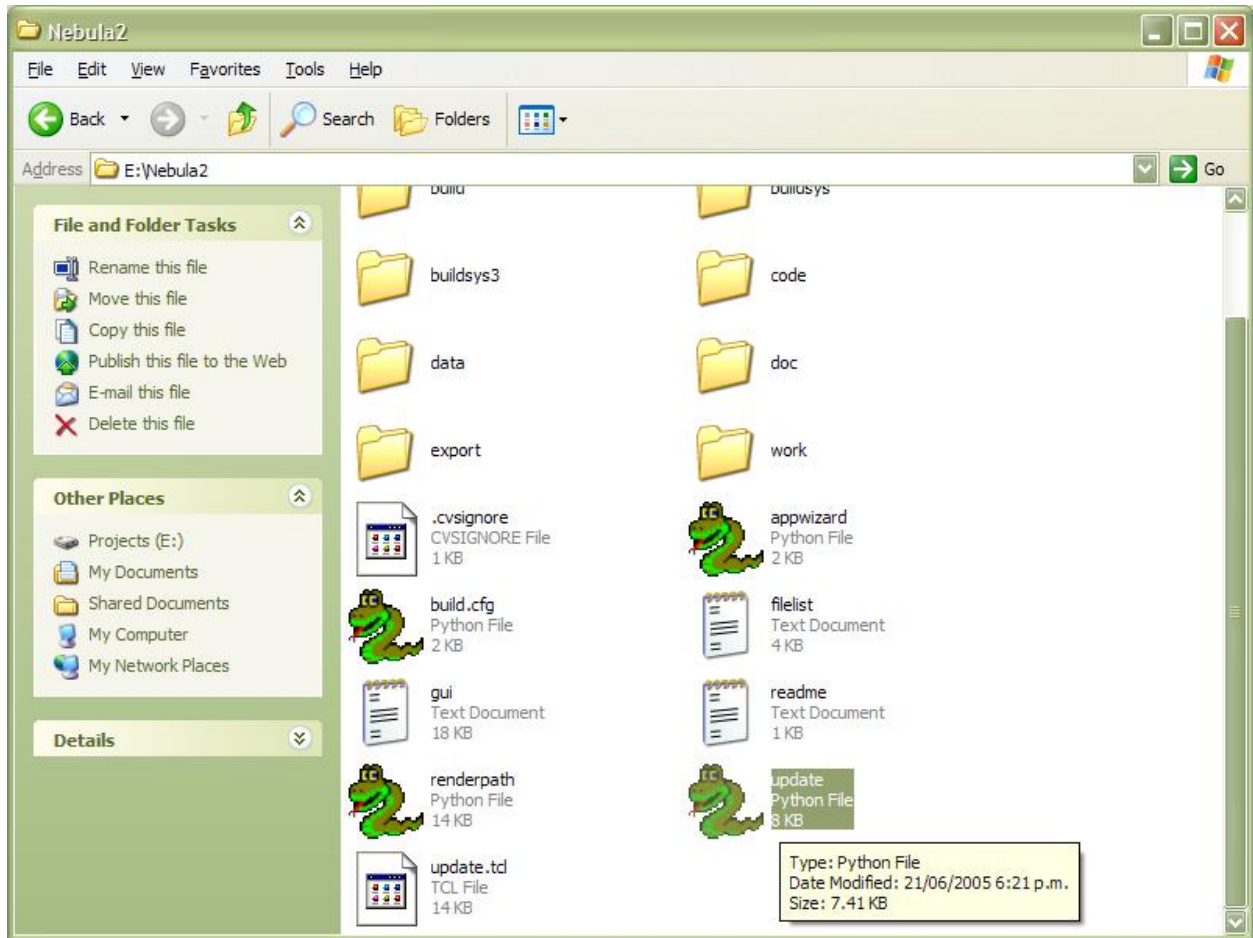
- wxPython 2.6 혹은 2.8 버전을 권장, <http://www.wxpython.org>

Radonlabs 사에서는 Tcl을 사용하지만 SourceForge의 오픈 소스 커뮤니티에서는 Python을 사용해서 만든 빌드 스크립트를 사용하고 있습니다. 이 빌드 시스템은 front-end GUI를 제공하는데, 이 GUI 툴이 wxPython으로 작성이 되어 있기 때문에 wxPython 패키지의 다운로드도 필요합니다. (wxPython은 GUI 빌드 시스템을 사용하는 경우에만 필요한 패키지입니다. 커맨드 라인에서 빌드를 선호하는 경우라면 꼭 설치할 필요는 없습니다.) Python와 wxPython은 위의 링크된 사이트에서 다운 받을 수 있습니다.

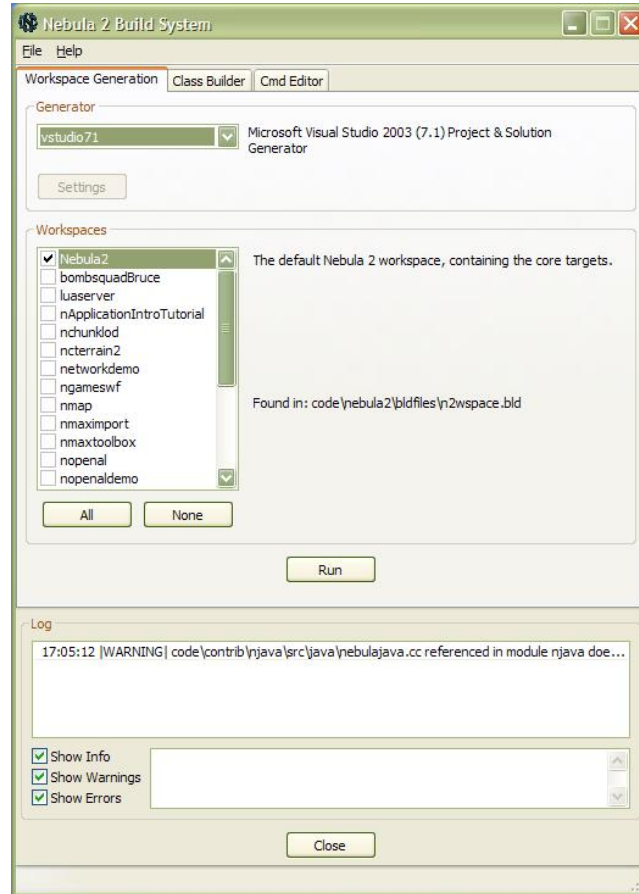
Nebula에서 VisualStudio 등을 사용해서 프로젝트 파일을 생성하는 것이 아니라 별도의 프로젝트 파일을 생성하는 스크립트 파일을 사용하는 이유는 모듈별 관리를 용이하게 하기 위해서입니다.

Nebula 엔진의 특징 중 하나는 모듈러 프로그래밍을 지향한다는 점인데 코드 뿐만 아니라 프로젝트의 관리 역시 모듈 별로 구분해서 쉽게 관리할 수 있는 방법을 제공하기 위해서 이렇게 프로젝트 파일을 생성하는 스크립트를 제공하고 있습니다. 개별 모듈에 대한 프로젝트(솔루션) 파일은 빌드 파일(.bld) 파일을 작성하여 생성하게 됩니다. 만약에 필요 없는 모듈이 있는 경우 빌드 파일에서 해당 모듈에 대한 부분만 삭제하면 필요한 프로젝트 파일을 구성할 수 있으므로 모듈 개수가 많은 경우에도 쉽게 관리할 수가 있습니다.

먼저 update.py 파일을 실행합니다.



아래와 같이 Nebula2 Build System 다이얼로그를 나타내는 것을 확인할 수 있습니다.



Generate 콤보 박스에서 사용하는 킴파일러를 선택한 다음 **Workspace**에서 생성하고자 하는 모듈을 모두 선택합니다. 선택이 끝난 다음에 'Run' 버튼을 누르게 되면 자동으로 빌드에 필요한 파일들이 생성됩니다.

빌드 시스템의 작동 원리는 주어진 하위 디렉토리들을 검사하여 .bld 빌드 파일이 존재하면 해당 빌드 파일에 대한 솔루션 파일(윈도우즈 플랫폼이라고 가정하는 경우)을 생성하는 것입니다. 빌드 시스템이 .bld 파일의 유무를 검사하는 디렉토리는 다음과 같습니다.

- \$home/code/nebula2/bldfiles and its subdirectories
- \$home/code/XXX/bldfiles
- \$home/code/contrib/XXX/bldfiles

Nebula 엔진은 다른 외부 라이브러리들을 사용하기 때문에 엔진을 빌드하기 위해서는 이들 외부 라이브러리의 헤더 파일과 라이브러리 파일이 필요합니다. 필요한 외부 라이브러리는 다음과 같습니다.

- ogg

- `sqlite`
- `vorbis`

빌드에 필요한 외부 라이브러리는 [SourceForge Nebula](#) 페이지에서 다운로드할 수 있습니다. 다운로드 받은 파일은 압축을 해제한 다음 `code/nebula2` 디렉토리에 복사하면 됩니다.

SDK의 빌드는 이상과 같이 할 수 있지만 사용자가 프로젝트를 생성하려면 **Nebula**의 빌드 시스템에 맞게 직접 빌드 파일(`.bld` 파일)을 작성해야 합니다. 그러면 이번에는 직접 빌드 파일을 작성하는 방법에 대해서 살펴 보도록 하겠습니다.

## Nebula 빌드 파일 작성하기

Nebula2의 빌드 파일은 크게 다음의 세가지로 구성되어 있습니다.

- 모듈(*Module*)? - 헤더 파일과 소스 파일을 정의합니다.
- 타겟(*Target*)? - 모듈들의 집합을 정의하며, 이들 집합의 결과인 실행 파일, 정적/동적 라이브러리에 대한 여러가지 설정들을 정의합니다.
- 워크스페이스(*Workspace*)? - 보통 같이 빌드 되는 타겟들의 집합들에 대해서 정의합니다.

모듈부는 `nRoot` 클래스를 상속한 클래스의 헤더 파일(`.h`)과 소스 파일(`.cc`)로 정의되며 모듈부는 하나의 `nRoot` 클래스를 상속한 클래스만을 포함할 수 있습니다. 즉, 두 개의 새로운 `nRoot` 클래스를 상속한 클래스가 있다면 두 개의 모듈부를 정의해 주어야 합니다.

그러면 첫번째로 모듈부를 정의하는 방법부터 살펴 보도록 하겠습니다.

```
beginmodule nmyclass1
  setdir wheretofindthefollowing
  setfiles { nmyclass1_main }
  setheaders { nmyclass1 }
endmodule
```

```
beginmodule nmyclass2
...
endmodule
```

모듈부는 `beginmodule`와 `endmodule` 키워드로 모듈의 시작과 끝을 정의하게 됩니다. `beginmodule` 다음에는 모듈의 이름이 위치하게 됩니다. 위의 예에서 첫번째 모듈의 이름은 `nmyclass1`입니다.(클래스 이름 명명법과 마찬가지로 모듈의 이름 역시 **Nebula** 엔진에서는 접두사 `n-`으로 시작하는 것이 관례입니다)

`setdir` 키워드는 모듈을 구성하는 헤더 파일과 소스 파일이 위치하는 디렉토리를 정의하는 키워드입니다. `setfiles`는 소스 파일을 정의하는 키워드이고 `setheaders`는 헤더 파일을 정의하는 키워드입니다.

nRoot 클래스를 상속하는 클래스들은 스크립팅 인터페이스를 가지는 것이 보통인데 이 때 C++ 멤버 함수들을 정의하는 소스 파일은 `nmyclass1_main.cc`과 같이 접미사로 `'_main'`을 스크립트 인터페이스를 정의하는 파일은 `'_cmds'` 접미사를 가지는 것이 관례입니다. `setfiles`와 `setheaders`에 소스 파일과 헤더 파일을 정의할 때 `.cc` 와 `.h` 파일 확장자는 생략합니다.

모듈부에는 하나의 nRoot 클래스를 상속한 클래스의 헤더 파일과 소스 파일만 정의해야 한다고 했습니다. 만약에 모듈부를 정의한 다음 빌드 시스템을 통해서 솔루션 파일을 만들어서 빌드했을 경우 다음과 같은 에러가 발생했다면 모듈부에 하나 이상의 nRoot 클래스를 상속한 클래스를 정의하지 않았는지 살펴 보기 바랍니다.

```
LNK2005: "bool __cdecl n_init_Module_App(class nClass *,class nKernelServer *)"
(?n_init_Module_App@@YA_NPAVnClass@@PAVnKernelServer@@@Z) already defined in
myapp.obj
LNK2005: "void * __cdecl n_new_Module_App(void)" (?n_new_Module_App@@YAPAXXZ)
already defined in myapp.obj
```

위의 링크 에러에서 볼 수 있듯이 모듈을 초기화하는 `n_init` 함수나 메모리 할당을 위한 `n_new` 함수가 이미 정의되어 있다는 것은 같은 모듈부에 하나 이상의 모듈이 정의되어 있기 때문에 발생하는 에러입니다.

(※ `n_init` 함수는 빌드 시스템을 통해서 모듈마다 자동으로 생성됩니다.)

다음으로는 타겟부를 정의하는 방법에 대해서 살펴 보겠습니다.

```
begintarget mylibrary
  settype lib
  setmodules { nmyclass1 nmyclass2 }
endtarget
```

모듈과 마찬가지로 `begintarget` 키워드와 `endtarget` 키워드로 시작과 끝을 정의하며 `begintarget` 키워드 다음에는 타겟의 이름을 정의합니다.

`settype` 키워드는 타겟의 형태를 정의하는 것으로 타겟은 실행 파일(exe), 정적 라이브러리(lib), 동적 라이브러리(dll)의 형태중 하나를 가지게 됩니다.

`setmodules` 키워드는 타겟을 구성하는 모듈들을 정의하는 키워드입니다. 모듈들을 같은 파일내에 위치하지 않더라도 빌드 시스템에서 빌드 파일들을 해석할 때 자동으로 찾게 됩니다.

마지막으로 워크스페이스부를 정의하는 방법입니다. 워크스페이스부는 Visual Studio의 솔루션 파일에 대응하는 부분입니다.

```
beginworkspace myProjectWorkspace
  setbinarydir ./bin/
  settargets {
```

```
    mylibrary
    nkernel
    nnebula
}
endworkspace
```

앞서 살펴 본 모듈 및 타겟과 마찬가지로 워크스페이스부 역시 *beginworkspace*와 *endworkspace* 키워드로 시작과 끝을 정의하며 *beginworkspace* 다음에는 워크스페이스부의 이름을 정의하게 됩니다. *setbinarydir*는 생성되는 바이너리 파일을 저장할 디렉토리를 정의하는 키워드로 Nebula 엔진을 설치한 \$home 디렉토리의 하위 디렉토리로 지정이 됩니다. 즉, 위의 예에서 Nebula 엔진이 'd:/dev/nebula2'에 인스톨 되어 있는 경우라면 바이너리 파일은 'd:/dev/nebula2/bin' 디렉토리 아래에 생성되는데 이 때 디버그 모드의 파일은 'win32d' 디렉토리에, 릴리즈 모드의 파일은 'win32' 디렉토리에 생성이 됩니다.

*settarget* 키워드는 워크스페이스를 구성하는 타겟들을 정의하는 키워드입니다. 그리고 *settarget* 키워드에 정의된 첫번째 타겟이 Visual Studio 솔루션의 활성화 프로젝트가 됩니다.

매번 빌드 파일을 작성하는 것이 번거롭다고 생각되면 Python이나 Lua, Perl, Ruby 등과 같은 다른 스크립트 언어를 사용해서 주어진 모듈 등의 이름에 따라 적절히 빌드 파일을 자동으로 생성하는 간단한 프로그램을 만들어 보는 방법을 생각해 볼 수도 있습니다.

Nebula엔진에서 이렇게 별도의 빌드 시스템을 사용하는 가장 큰 이유는 바로 모듈러 프로그래밍을 위해서입니다. 개발하는 게임에 따라 필요한 모듈들이 다양한데 이렇게 모듈에 기반한 빌드 시스템을 사용하면 필요한 모듈만 모아서 관리하는 것이 매우 쉽기 때문입니다. 만약 별도의 빌드 시스템 없이 Visual Studio의 솔루션 파일로만 관리를 해야 한다고 하면 수십개의 프로젝트 파일을 일일이 추가하고 삭제해야 하므로 불편할 뿐만 아니라 시간도 오래 걸리게 됩니다. Nebula 엔진의 스크립트를 사용한 빌드 시스템은 이러한 문제에 대한 좋은 예입니다. (최근의 게임 엔진과 같은 규모가 큰 소프트웨어는 Nebula와 유사한 빌드 시스템들을 가지고 있는 경우가 많습니다. Nebula에서는 Tcl이나 Python을 사용하고 있지만 다른 엔진에서는 Perl(<http://www.perl.org/>)도 많이 사용되고 있는 스크립트 언어 중의 하나입니다.)

#### 도움말 빌드하기

소스 코드의 빌드 외에 Nebula의 문서도 빌드를 통해서 컴파일된 도움말의 형태로 만들 수 있습니다. Nebula의 문서를 빌드하기 위해서는 doxygen이 필요합니다. doxygen은 아래의 사이트에서 다운 받을 수 있습니다.

- <http://www.stack.nl/~dimitri/doxygen/>

doxygen을 통한 도움말 html 생성은 커맨드 라인 창(혹은 Linux나 Mac OS X의 경우 터미널)에서 다음의 방법으로 생성할 수 있습니다

```
$nebulas2/code/doxycfg> doxygen nebula2.cfg
```

'*nebula2.cfg*' 파일은 *nebula2 doxygen*의 설정 파일(configuration file)로 도움말 html 생성을 위한 설정을 정의하고 있습니다. 생성된 html 파일은 윈도우즈의 경우 *Html Help* 를 사용해서 컴파일된 형태의 도움말로 변경할 수도 있습니다.

소프트웨어를 개발할 때는 주석뿐만 아니라 작성하는 소프트웨어에 대한 다양한 정보를 담고 있는 도움말을 작성하는 것은 매우 중요한 일입니다. 이것은 소프트웨어 작성후 개발 기간이나 소프트웨어의 크기에 따라 생략할 수 있는 선택 사항이 아니라 개발 초기부터 코드 작성과 같이-그리고 같은 중요도로- 병행되어야 하는 중요한 작업입니다. 특히나 작은 규모의 소프트웨어가 아닌 게임 엔진 처럼 개발 주기가 길고 다른 많은 사람들이 접근하게 되는 소프트웨어는 더욱 중요하다고 할 수 있겠습니다. 그러므로 문서화를 위해서 **doxygen**과 같은 툴을 사용하는 것부터 전체 소프트웨어의 개발 주기에 맞추어서 문서화와 관련한 계획도 개발 초기부터 세워서 같이 병행하는 것이 중요합니다.