Lesson 0 - Introduction to Playstation 3 programming

Summary

A brief overview of the Playstation 3 development environment, and how to set up a PS3 project solution to run on the PS3 Devkits.

New Concepts

Using the ProDG Target Manager, ELF and SELF files, APP_HOME directory, PPU and SPU programming

Introduction

Writing basic programs to run on the Playstation 3 is simple - it has fully ANSI C/C++ compliant compilers, so the previous tutorials should just compile and run straight away. However, unlocking the full potential of the Playstation 3 hardware requires efficiently using the 6 Synergistic Processing Units, and that is a much more difficult task. The development environment is also slightly different than you will be used to, despite being based on Visual Studio. These tutorials will show you how to compile and run programs on the PS3 and how to use the SPUs for various tasks, before moving on to programming the graphics hardware, reading input from the joypad, and finally how to output sound.

Development Environment

Playstation 3 SDK

The Playstation 3 SDK consists of the headers and libraries you'll use when programming, example code, extensive documentation of the provided APIs, and Visual Studio integration. This integration is provided by the SN Systems *ProDG* product suite, which has everything you need to compile, run, and debug code created in Visual Studio on the Playstation 3. It also includes the software required to interface with a PS3 Devkit, the *ProDG Target Manager*. For the Visual Studio integration to work, you'll need Visual Studio *2003*, *2005*, or *2008*; *2010* is currently unsupported.

Playstation 3 Devkits

The Playstation 3 Development Kits we have are *DECR-1400A* models - the most up-to-date devkits that Sony provide. Externally, they look pretty much the same as the original 60GB retail Playstation 3s, but with an additional LAN port on the back, used to send and receive development code and debug data. Internally, they have 512MB of system RAM compared to the 256MB in retail units, and have custom firmware, which among other things, provides access to terminal output and disables BLURAY and DVD movie playback. They can load and play PS3 games just as a normal retail PS3 can, and otherwise act just like the PS3 consoles you are used to.

ProDG Target Manager

Connecting to, and controlling the Devkits is done using the ProDG Target Manager. The Target Manager allows you to see terminal output, take screen grabs of the video output, and remote control various aspects of the devkits. For example, it allows you to reset the devkit remotely - useful if it's in a different room (on a different floor!), and you've just ran some broken code on it.

API Documentation

Sony include extensive documentation for almost everything in their SDK. It can be searched through by using the *quick_find_e* catalog, found in the SDK install folder (By default this folder will be *C:/usr/local/cell*). Try it! use quick_find_e and search for 'Target Manager' or 'SELF', you'll find lots of references to the new things being introduced to you here. Throughout the rest of the tutorials, you'll find lots of references to Sony SDK API functions - although their basic usage will be explained in each tutorial, you are encouraged to use the API documentation to learn more about what these functions do and what parameters they need.

File formats

Although you will be using Visual Studio to develop PS3 applications, the underlying programs used when compiling your code are different, and they don't output the .exe and .lib files you might be used to. Instead, whether you are writing PPU or SPU programs, you will be dealing with '*elf*' files. So what's an elf file? Elf stands for *Executable and Linkable Format*, which as the name suggests, is a format to represent executables and libraries - both static and dynamicly linked. It's quite a popular file format in UNIX, and is used by both Sony and Nintendo for their games consoles. More specifically, we will be using *Signed* ELF (SELF) files. As part of the Digital Rights Management scheme used on the Playstation 3, all executables must be digitally signed by Sony before they'll run on a PS3. The PS3 Devkits we will be using will also run 'fake signed' code, which is generated behind the scenes when compiling PS3 software with the PS3 SDK.

Running programs

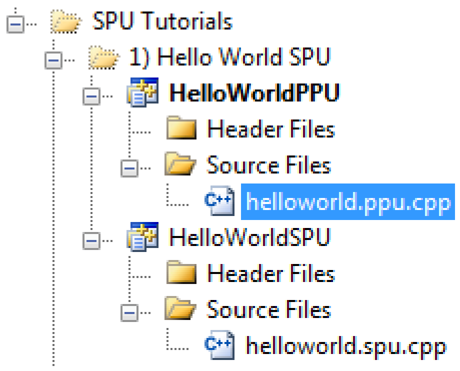
Running PPU programs is easy - if you've set up your Target Manager correctly, you can run your PPU program just as you would a Windows program. The ProDG Debugger will connect to the devkit, reset it, and make it load in your compiled ELF file. From the target manager, you can see the console output of your program, and take snapshots of what your program is sending to the 'video out' - handyif your devkit isn't connected to a monitor and you want to check your graphics code is working. Running SPU programs is less easy - you have to load your SPU program in from inside a PPU program, and execute it from there. If you've ever loaded in a .dll file at runtime in one of your programs, you can think of SPU programs as just being very special dll files that only run on SPUs. We'll cover how to load in these SPU files in the first proper tutorial, but for now we'll look at where to put SPU SELF files to get them loaded in.

APP_HOME

When you run a program on the PS3 devkits, they need to know where to look for files - whether they be game assets like textures and sound files, or the SPU programs that will run during your programs execution. In a 'real' game, this would be the Blu-ray drive and the hard drive, but when compiling code to run on the devkits, it's *APP_HOME* - a special function of the Target Manager that points to a folder on your computer. The devkit can then use this folder just as it would its Blu-ray or hard drive. You can set the folder used as APP_HOME in the Target Manager, or directly from in Visual Studio. Doing it from Visual Studio is recommended, as then you can easily have a seperate APP_HOME per project.

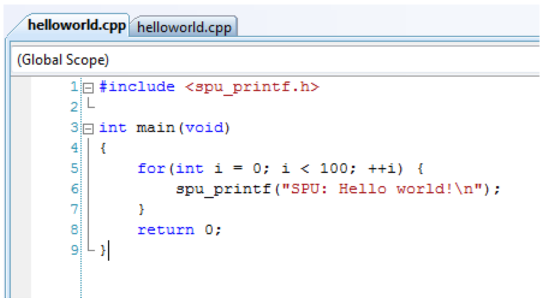
Visual Studio Projects

Some tutorials will require you to create both a PPU and an SPU project. In these cases, I suggest using the same name for both projects, suffixed with PPU or SPU as required. For example, in our next tutorial, we will be creating a 'Hello World' program that requires both PPU and SPU projects - I name them HelloWorldPPU and HelloWorldSPU. Such nomenclature helps keep track of which projects are related, and what type of processor they are being compiled for.



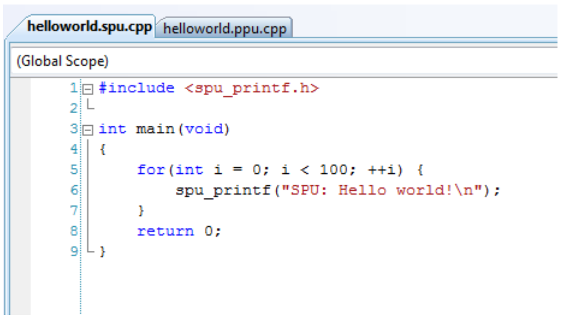
Easy to see that these projects are related...

Along the same lines, when naming .cpp files in such PPU/SPU combined projects, I append ppu or spu as appropriate (main.ppu.cpp, main.spu.cpp etc). This helps keep track of what file is being edited when dealing with multi-project solutions.



```
#include <spu_printf.h>

int main(void)
{
    for(int i = 0; i < 100; ++i) {
        spu_printf("SPU: Hello world!\n");
    }
    return 0;
}
```

What's going on here?



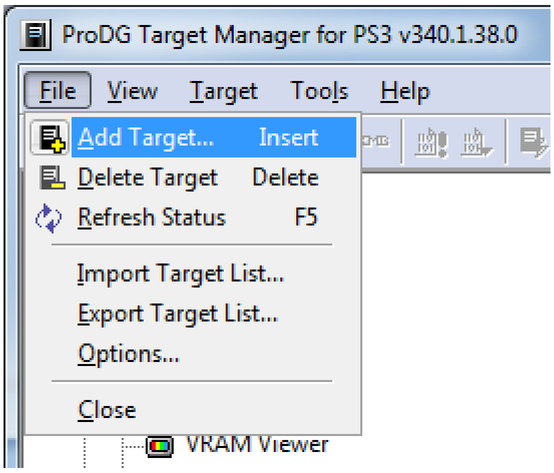
```
#include <spu_printf.h>

int main(void)
{
    for(int i = 0; i < 100; ++i) {
        spu_printf("SPU: Hello world!\n");
    }
    return 0;
}
```

Ah! One is the PPU project, the other the SPU project

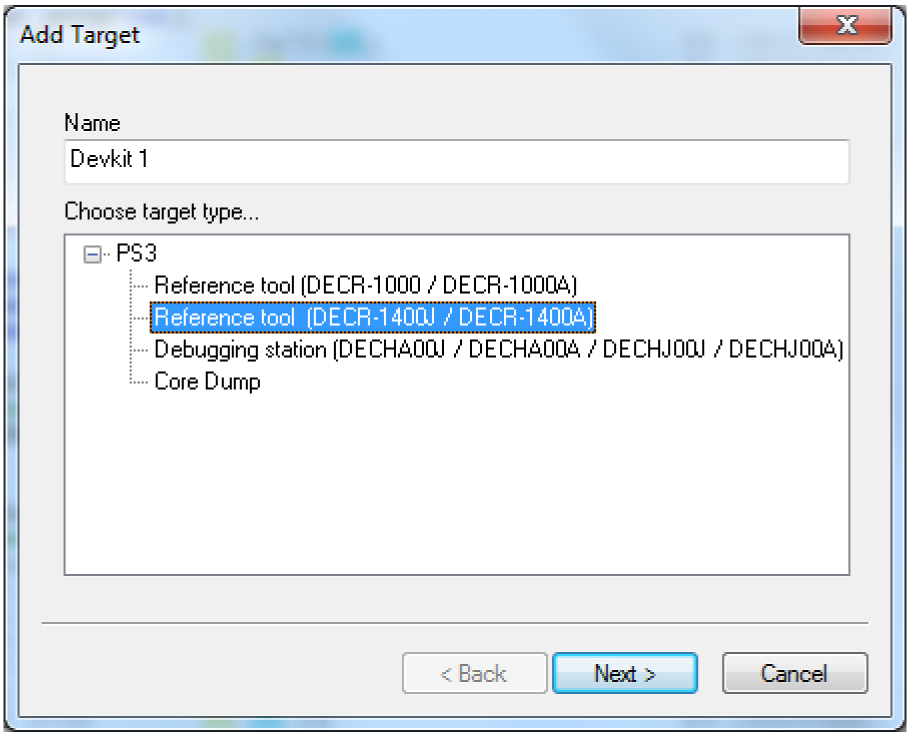
Appendices

Appendix A: Connecting to a PS3 Devkit

Step 1) Start the ProDG Target Manager, click 'File', then click 'Add Target...'



Step 2) Choose a name for the devkit, and select 'Reference tool (DECR-1400J / DECR-1400A) then click 'Next'



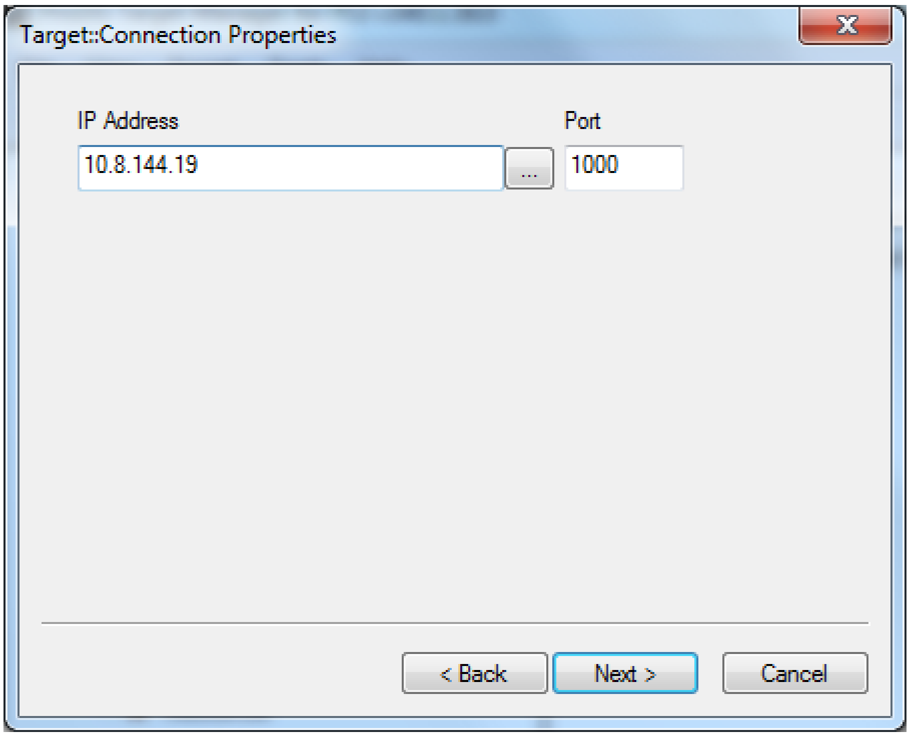
Name

Devkit 1

Choose target type...

PS3
- Reference tool (DECR-1000 / DECR-1000A)
- Reference tool (DECR-1400J / DECR-1400A)
- Debugging station (DECHA00J / DECHA00A / DECHJ00J / DECHJ00A)
- Core Dump

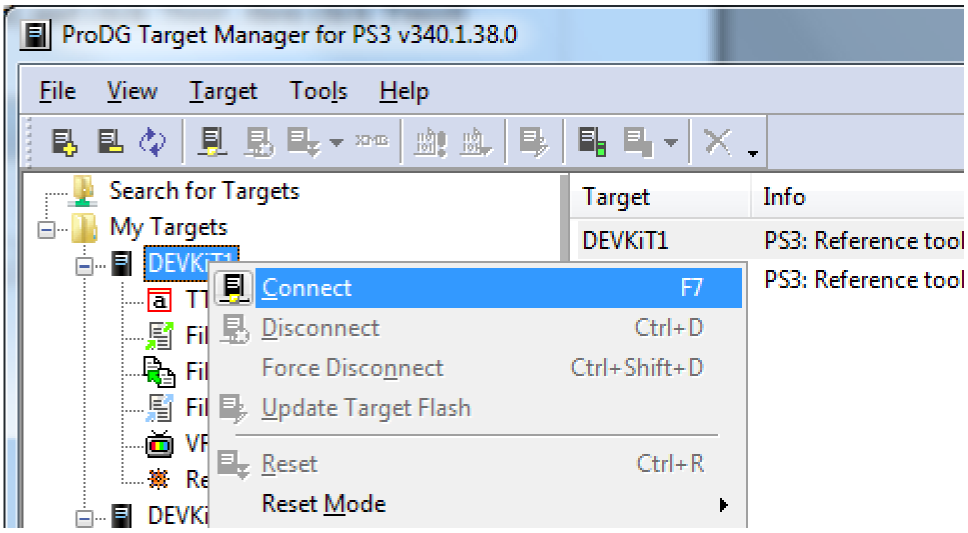
Step 3) Enter the IP address of the devkit you wish to add, and click 'Next', then click 'Finish'



Target::Connection Properties

IP Address	Port
10.8.144.19	1000

Step 4) The devkit should then appear under 'My Targets' in the ProDG Target Manager, and can be connected to by right clicking on the devkit, and selecting 'Connect'

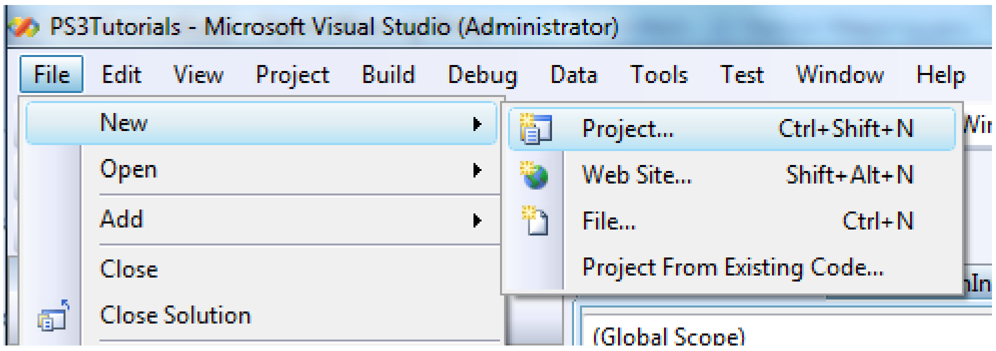


ProDG Target Manager for PS3 v340.1.38.0

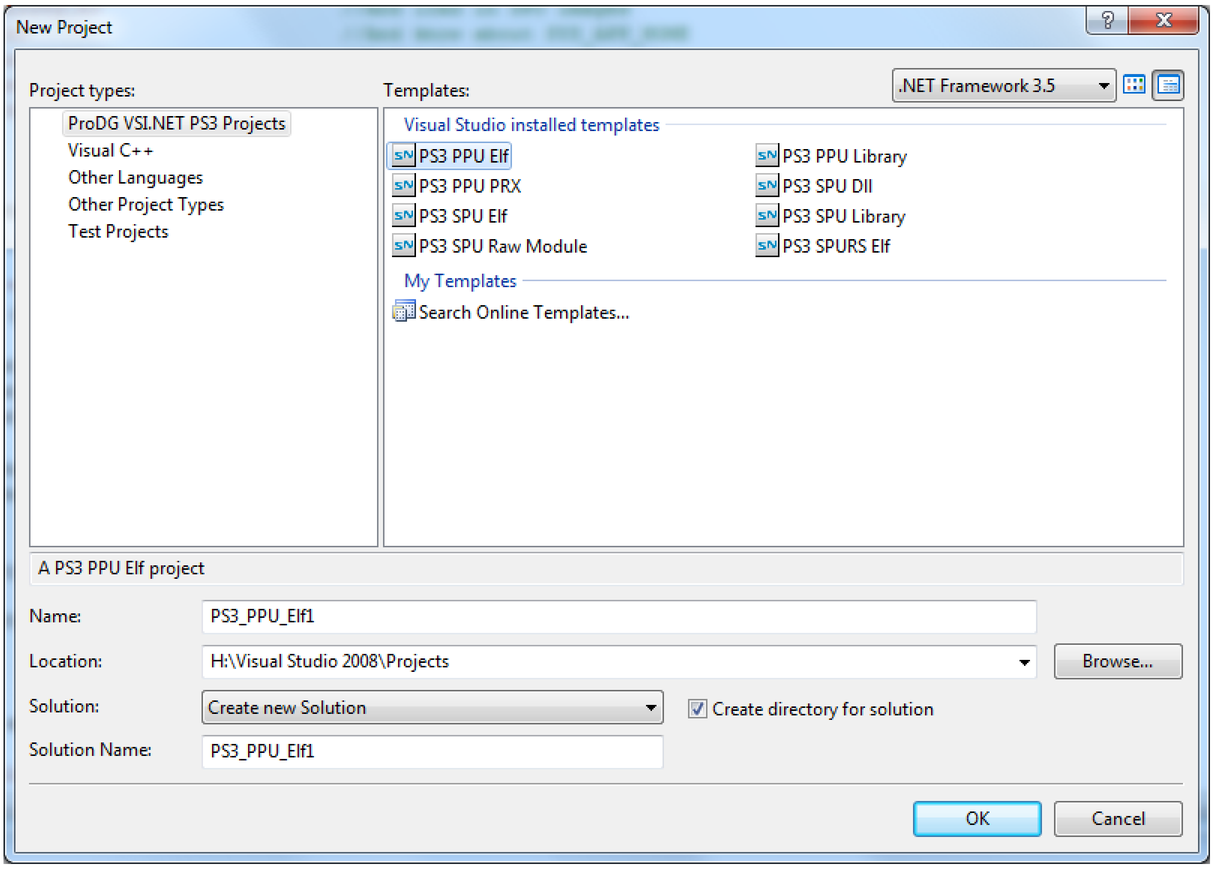
Target	Info
DEVKiT1	PS3: Reference tool
	PS3: Reference tool

Appendix B: Creating a PPU Project

Step 1) Open Visual Studio 2008, click 'File', highlight 'New', and click 'Project...'



Step 2) Click 'ProDG VSI.NET PS3 Projects', click 'PS3 PPU Elf', choose a name and location for your new project, and click 'OK'. Simple!



A PS3 PPU Elf project

Name: PS3_PPU_Elf1

Location: H:\Visual Studio 2008\Projects

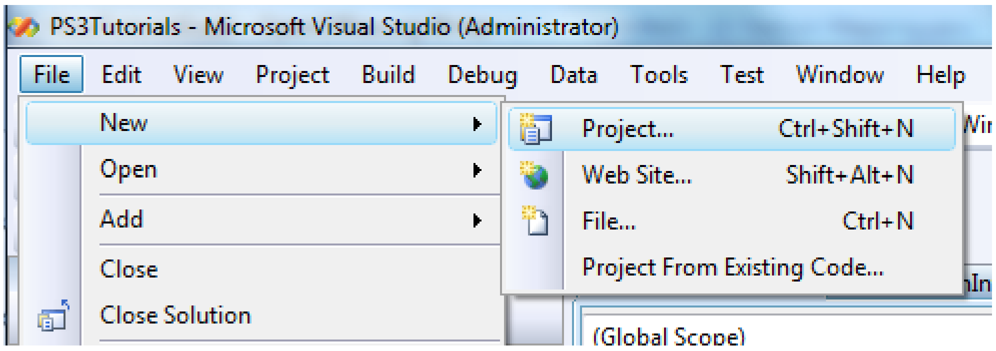
Solution: Create new Solution

☑ Create directory for solution

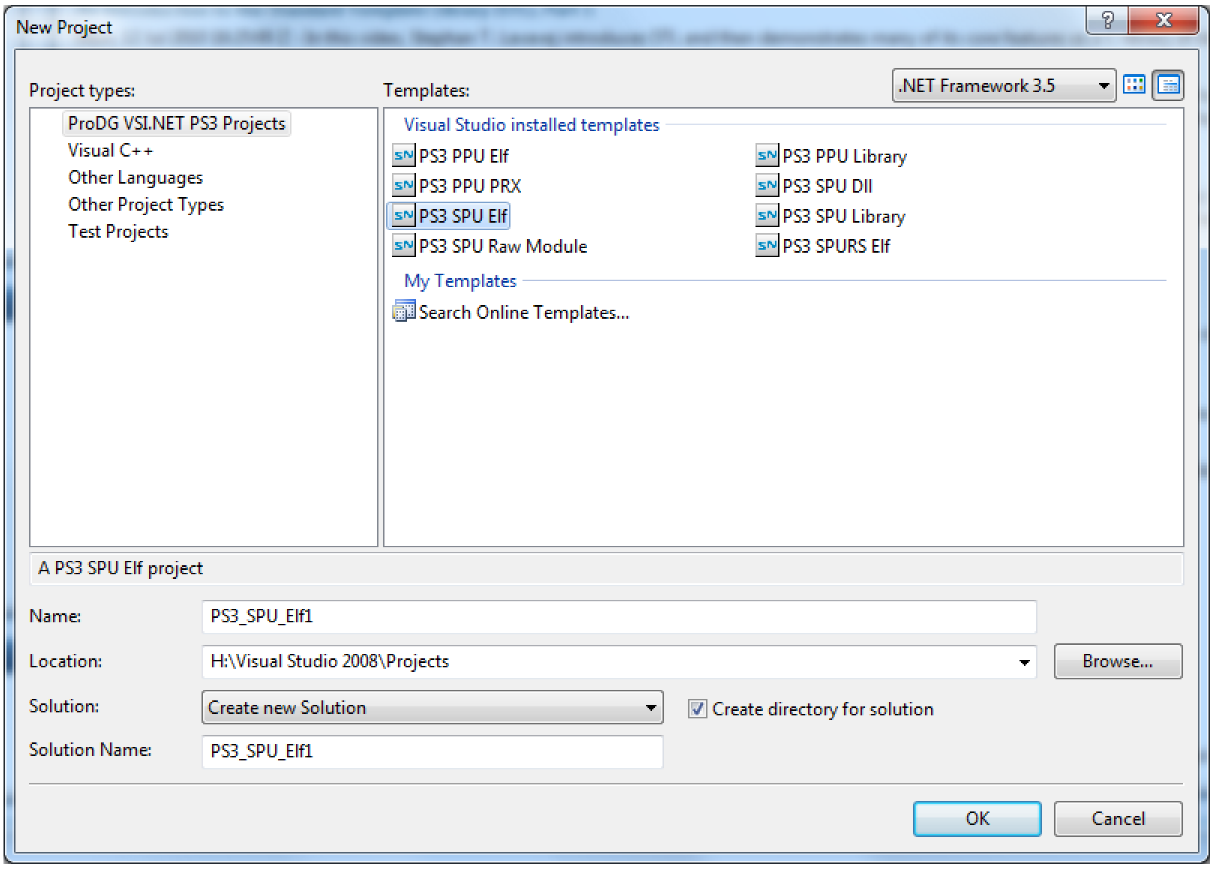
Solution Name: PS3_PPU_Elf1

Appendix C: Creating an SPU Project

Step 1) Open Visual Studio 2008, click 'File', highlight 'New', and click 'Project...'



Step 2) Click 'ProDG VSI.NET PS3 Projects', click 'PS3 SPU Elf', choose a name and location for your new project, and click 'OK'



A PS3 SPU Elf project

Name: PS3_SPU_Elf1

Location: H:\Visual Studio 2008\Projects

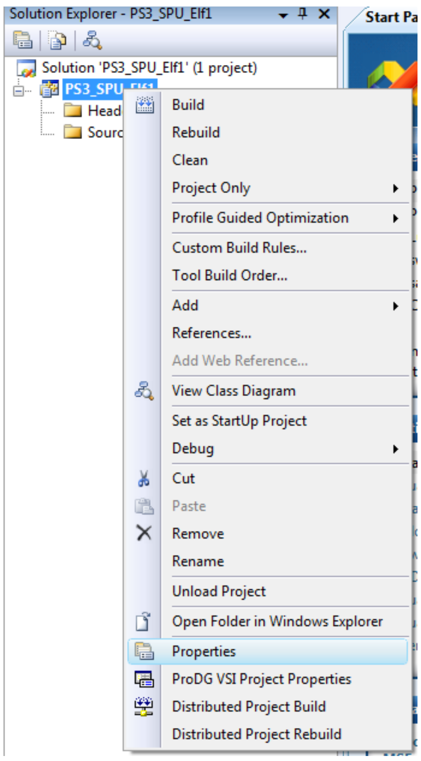
Solution: Create new Solution

[✓] Create directory for solution

Solution Name: PS3_SPU_Elf1

Note: These steps are optional, but are recommended if you don't want to manually move around the SPU .elf file every time you compile it.

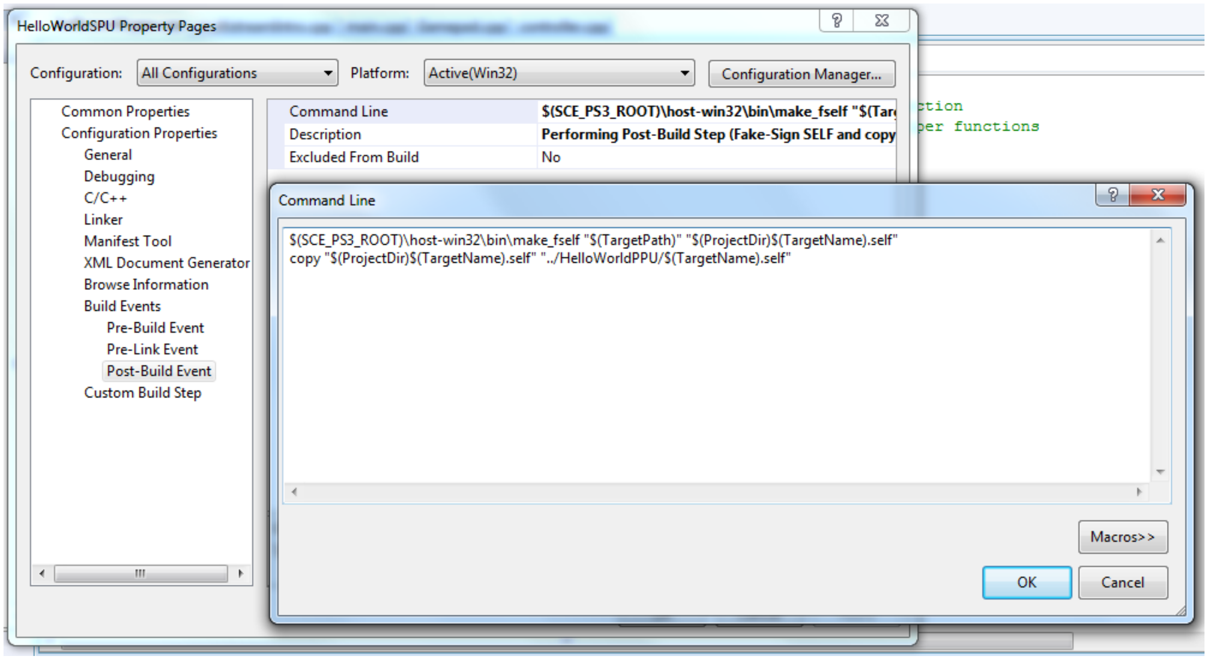
Step 3) Right click on your Solution in the Solution Explorer, and select 'Properties'


Solution Explorer - PS3_SPU_Elf1
Solution 'PS3_SPU_Elf1' (1 project)
Build
Rebuild
Clean
Project Only
Profile Guided Optimization
Custom Build Rules...
Tool Build Order...
Add
References...
Add Web Reference...
View Class Diagram
Set as StartUp Project
Debug
Cut
Paste
Remove
Rename
Unload Project
Open Folder in Windows Explorer
Properties
ProDG VSI Project Properties
Distributed Project Build
Distributed Project Rebuild

Step 4) Click the 'Configuration' drop down box and select 'All Configurations'. Under 'Configuration Properties' select 'Build Events' and then 'Post-Build Event'. Click on 'Command Line', and add the following line underneath the 'make_fself' command.

copy "$(ProjectDir)$(TargetName).self" "../<PPU Project>/$(TargetName).self"

Replace the term *<PPU Project>* with the folder name of the PPU project you want to run the SPU program from, then click OK.



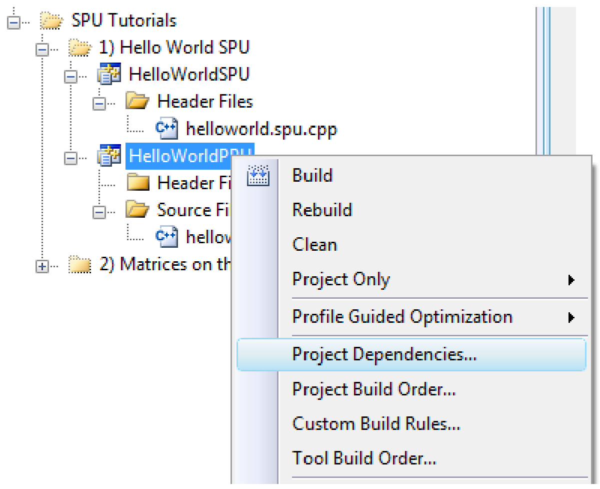
Configuration: All Configurations
Platform: Active(Win32)

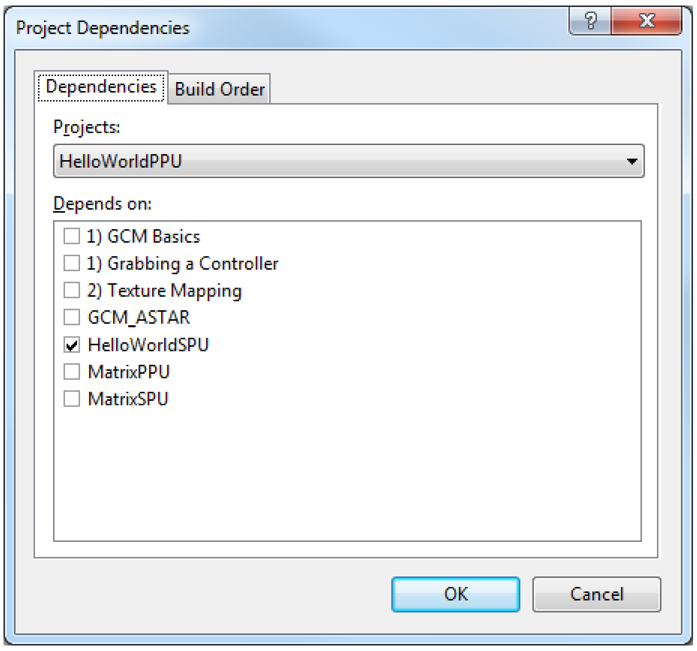
Command Line	$(SCE_PS3_ROOT)\host-win32\bin\make_fself "$(Tar
Description	Performing Post-Build Step (Fake-Sign SELF and copy
Excluded From Build	No

Command Line

```
$(SCE_PS3_ROOT)\host-win32\bin\make_fself "$(TargetPath)" "$(ProjectDir)$(TargetName).self"
copy "$(ProjectDir)$(TargetName).self" "../HelloWorldPPU/$(TargetName).self"
```

Step 5) Set the project dependencies of the PPU project that uses your SPU code so that it depends on the SPU project. This will make the SPU project automatically compile and copy the updated SELF file when necessary.




Project Dependencies

Dependencies | Build Order

Projects: HelloWorldPPU

Depends on:

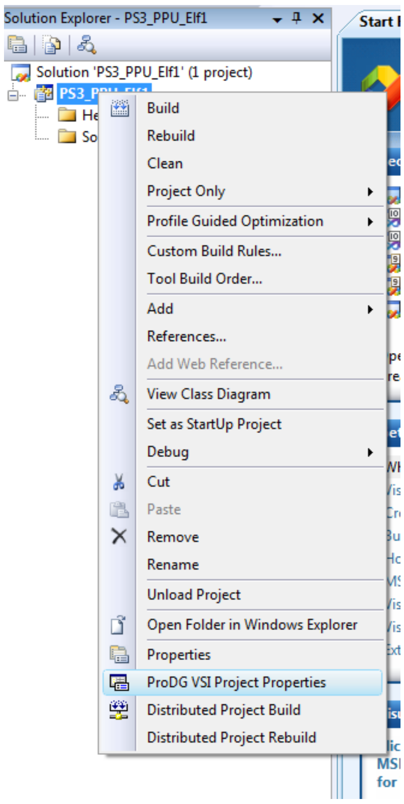
- [ ] 1) GCM Basics
- [ ] 1) Grabbing a Controller
- [ ] 2) Texture Mapping
- [ ] GCM_ASTAR
- [x] HelloWorldSPU
- [ ] MatrixPPU
- [ ] MatrixSPU

OK | Cancel

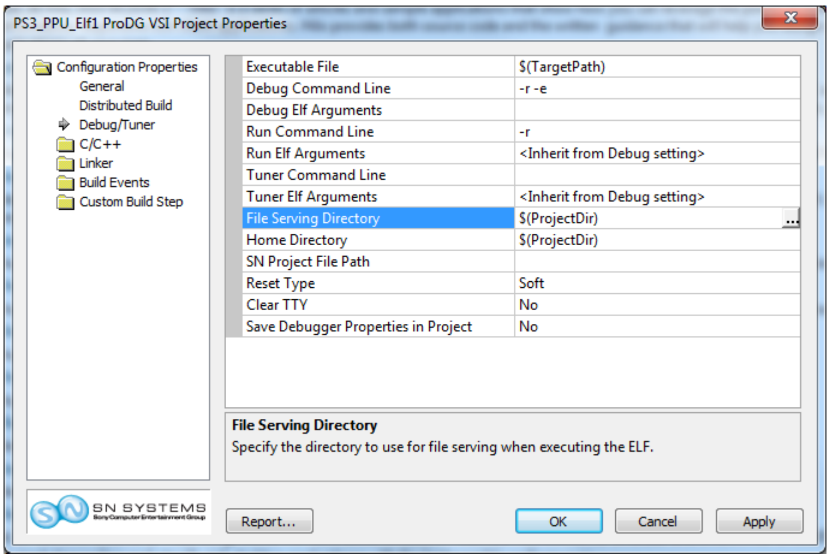
Appendix D: Setting the APP_HOME directory

If your program uses external assets such as sound files or textures, it must know where these files are kept. To do this, we set a 'home directory' to indicate where to find such resources.

Step 1) Right click on your Solution in the Solution Explorer, and select 'ProDG VSI Project Properties'


Solution Explorer - PS3_PPU_Elf1
Solution 'PS3_PPU_Elf1' (1 project)
Build
Rebuild
Clean
Project Only
Profile Guided Optimization
Custom Build Rules...
Tool Build Order...
Add
References...
Add Web Reference...
View Class Diagram
Set as StartUp Project
Debug
Cut
Paste
Remove
Rename
Unload Project
Open Folder in Windows Explorer
Properties
ProDG VSI Project Properties
Distributed Project Build
Distributed Project Rebuild

Step 2) Under 'Configuration Properties', click 'Debug/Tuner' and choose locations for the 'File Serving Directory' and the 'Home Directory', then click OK. Generally I use '$(ProjectDir)', a macro which will set the directories to wherever your project solution is saved to.



Executable File	$(TargetPath)
Debug Command Line	-r -e
Debug Elf Arguments	
Run Command Line	-r
Run Elf Arguments	<Inherit from Debug setting>
Tuner Command Line	
Tuner Elf Arguments	<Inherit from Debug setting>
File Serving Directory	$(ProjectDir)
Home Directory	$(ProjectDir)
SN Project File Path	
Reset Type	Soft
Clear TTY	No
Save Debugger Properties in Project	No

File Serving Directory
Specify the directory to use for file serving when executing the ELF.