Quick Start guide

ProDG for PlayStation®2

SN Systems

Version 3.0

December 2002

Quick Start guide to ProDG for PlayStation 2 v3.0

Take Care Handling Your ProDG CD

The CD-ROM disc enclosed with this package must be treated with care:

- Hold the disc by its edges and always keep it in the protective case when not in use.
- Do not allow dirt to be deposited on the disc. Clean the disc with a dry lint-free cloth, carefully wiping the cloth in straight lines from the centre outwards. Do not use any liquid cleaning agents or solvents.
- Do not allow the CD to become bent, scratched or to get wet.
- Do not expose the CD to extremes of temperature, especially do not leave it exposed to direct sunlight or a source of heat.

Customer support

First line support for all SN Systems products is provided by the Support areas of our website. To view these pages you must be a registered user with an SN Systems User ID and Password.

- If you have forgotten your User ID and Password, send an e-mail to **webmaster@snsys.com** and we will send you a reminder.

Once you have a valid User ID and Password you can visit our website Support areas at these URLs:

www.snsys.com/support (English)
www.snsys.jp/support (Japanese)

If the answer to your problem cannot be found on the Support areas of our website, you can also e-mail our support team at:

support@snsys.com (English)
j-support@snsys.com (Japanese)

Please make sure that you explain your problem clearly and include details of your software version and hardware setup. If you have been given an SN Systems support log number (LN number) then this should be quoted in all correspondence about the problem.

SN Systems Limited
4th Floor - Redcliff Quay
120 Redcliff Street
Bristol BS1 6HU
United Kingdom

Tel.: +44 (0)117 929 9733
Fax: +44 (0)117 929 9251

WWW: www.snsys.com (English)
www.snsys.jp (Japanese)

Updating ProDG over the web

You will find regular updates of all the components in ProDG for PlayStation 2 on the SN Systems web site (see "Customer support" on page i). We recommend that you visit the support pages regularly and also look out for SN Systems e-mails alerting you to new versions of the software.

You must be a licensed ProDG customer, with an SN Systems user name and password, to access the support pages.

Updating Sony libraries and toolchain

You do NOT normally need to reinstall ProDG for PlayStation 2 when Sony issue patch and minor upgrades to the libraries. However, we recommend that you DO reinstall ProDG for PlayStation 2 when Sony issue a major new release of the libraries, e.g. from v1.6 to v 2.0. To avoid "contamination" between full releases it is advisable to do a fresh install so you do not have duplicate libraries in the ..\libs directory and the ..\libs\OLD subdirectory.

When Sony releases new toolchain versions, DO NOT install them until you obtain the equivalent toolchain build from SN Systems. Information about the currently supported version of the Sony toolchain is available from the SN Systems web site.

Contents

1 Introduction

Thank you for purchasing **ProDG for PlayStation 2**, the leading development tool suite for Sony PlayStation 2 game developers.

ProDG for PlayStation 2 is a suite of development tools for building and debugging PlayStation 2 games. It consists of a C/C++ compiler, assemblers, linker, debugger and target manager. An optional Visual Studio 6.0 integration provides App-Wizards for building executables and libraries, and launching the ProDG Debugger from within Visual Studio 6.0.

2 Installation

This section describes how to install the various components of ProDG in your development PC.

System requirements

Your development PC must meet the following minimum system requirements.

- Windows 98SE, Windows NT, Windows 2000 or Windows XP Pro
- Pentium II (or equivalent) or higher processor
- 256MB of RAM
- 8MB of video memory
- At least 300MB of hard disk space
- CD-ROM drive
- Network card (used by SN Systems licensing)

In addition you will need:

- Sony Computer Entertainment's PlayStation 2 Development Tool DTL-T10000 (available from Sony Computer Entertainment Inc.). You must be a licensed Sony developer.

You will also need to have available the following software:

- ProDG for PlayStation 2 (CD-ROM with this package)
- Sony PlayStation 2 libraries (v1.6.0 or later)
- Microsoft Visual Studio 6.0 (if using the ProDG Visual Studio 6.0 integration to provide an IDE)

Installation order

ProDG requires that the Sony development libraries are installed first.

Next you must decide if you will be using Microsoft Visual Studio 6.0 as your ProDG IDE, as you must install Visual Studio 6.0 before you install the ProDG tools. Then you can install ProDG for PlayStation 2.

When you put the ProDG CD-ROM in your CD drive, you will be given a menu of options which allow you to install the ProDG components separately.

Install the Sony libraries

- Create an empty \usr\local directory on your hard disk, renaming any existing \usr\local directory tree out of the way.
- From the Sony PlayStation 2 libraries CD-ROM, extract the tlib-nnn*.tgz zip file found in \Run Time\Tools-Programming\Standard_Libraries, to your \usr\local directory.

Install Visual Studio 6.0

- If you will be using the Microsoft Visual Studio 6.0 integration, install Microsoft Visual Studio 6.0 from the Visual Studio CD-ROM, and follow the instructions presented.
- Visual Studio 6.0 must be run at least once to initialize some registry settings.

Install ProDG

- Install ProDG for PlayStation 2 from the ProDG CD-ROM, and follow the instructions presented.

  Note: The installer will attempt to write to \autoexec.bat. If this is not present on the installation disk, e.g. under

Windows ME, create an empty \autoexec.bat and ensure that it is not read-only.

Note: On Windows NT and Windows 2000, you must be logged in with Administrator rights to be able to make the necessary changes to the registry and environment variables.

Reboot

- Restart your PC (unless installing to Windows NT or Windows 2000). You can do this at the end of the install process or later.

Obtain your SN Systems run-time license

All SN Systems products must be licensed. This section explains how you obtain and install your license key file.

1. When you ordered ProDG you would have been provided with a User ID. Go to **www.snsys.com** and click the link labelled **Product Activation**.
2. Click on the **Online Activation Centre** link and then enter your User ID in the form provided and click **Submit**. Confirm your company, name and product details, then click the **Issue License** link.
3. You will be sent an e-mail containing the key file and its contents will be displayed on screen. Place it in the same directory as your ProDG software and the software will be licensed.

3 Configuration

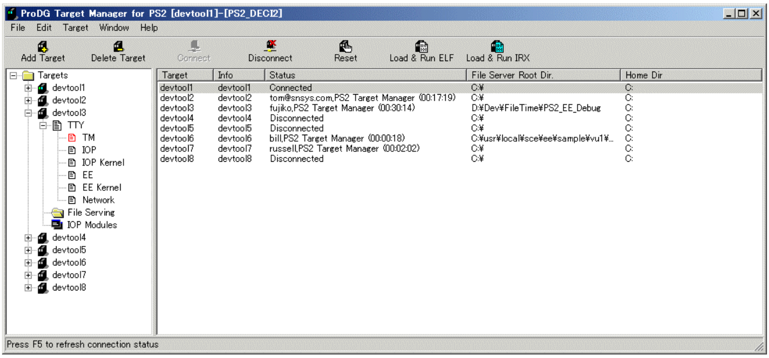
Before you can use any of the ProDG components, you must first configure the Target Manager utility so that the ProDG programs can communicate with your PlayStation 2 Development Tool DTL-T10000.

Overview of the Target Manager

ProDG Target Manager for PlayStation 2 is used to control access to any Sony PlayStation 2 Development Tool DTL-T10000 targets that you have on your network. It provides control, host fileserving and TTY output for all PlayStation 2 development kits on a LAN.

ProDG Target Manager for PlayStation 2 can be launched via the **ProDG for PlayStation2 > ProDG Target Manager for PS2** shortcut in the **Start** menu, and is automatically started when you start ProDG Debugger.

Target Manager main window



Target	Info	Status	File Server Root Dir.	Home Dir
devtool1	devtool1	Connected	C:¥	C:
devtool2	devtool2	tom@snsys.com,PS2 Target Manager (00:17:19)	C:¥	C:
devtool3	devtool3	fujiko,PS2 Target Manager (00:30:14)	D:¥Dev¥FileTime¥PS2_EE_Debug	C:
devtool4	devtool4	Disconnected	C:¥	C:
devtool5	devtool5	Disconnected	C:¥	C:
devtool6	devtool6	bill,PS2 Target Manager (00:00:18)	C:¥usr¥local¥sce¥ee¥sample¥vu1¥...	C:
devtool7	devtool7	russell,PS2 Target Manager (00:02:02)	C:¥	C:
devtool8	devtool8	Disconnected	C:¥	C:

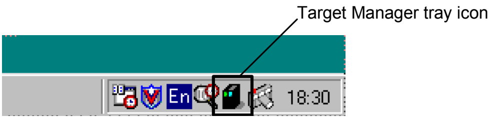
Press F5 to refresh connection status

Target Manager resembles Windows Explorer with target names on the left and properties on the right.

- The left-hand part of the window shows targets with their connection status (targets to which you are connected have the LEDs highlighted on the target icon).
- The right-hand part of the window shows target properties.

Target manager tray icon

When you minimize Target Manager it becomes a system tray icon:



To open the main window again, either double-click the tray icon or click **Open** in the tray icon's shortcut menu.

Adding targets

This section tells you how to set up a Development Tool DTL-T10000 target in Target Manager.

To add a new target

1. Click **Add Target** in the **File** menu. The Add Target dialog is displayed.
2. Enter a **Name** to identify the target e.g. "devtool1". Then click **Next**. The Target Properties dialog is displayed.
3. Enter the IP Address or host name of the Development Tool in the **IP Address** field. Enter the connection port in the **Port** field (normally 8510). Click **Next**.

   The Add Target Completed confirmation window is displayed, with details about the target to be added.
4. Click **Finish** to add the target.

Connecting to targets

Once a target has been added to Target Manager, you need to establish a connection session.

To connect to a target

1. Select the target that you want to connect to in the right-hand pane of the Target Manager main window.
2. Double-click on the target name, or click **Connect** in the **Target** menu, or click the Connect button in the toolbar.

   If your connection was successful then the target properties in the right-hand part of the window will be updated to show 'Connected' in the **Status** column.

   If another user is currently connected to the target a dialog is displayed saying that the target is in use by another user, and the identity of the user that is currently connected is shown in the **Status** column.

To update target connection status

You can refresh display of the current connection status and directory settings for all of the available targets, by pressing the shortcut key `<F5>`.

Troubleshooting connecting to targets

Your connection may time out which means that the target is unavailable on the network. There may be several reasons for this, including the following:

- The selected target may not be connected to the network correctly or it may be switched off.
- The TCP/IP software may not be correctly configured. For example, the PlayStation 2 may have a duplicate IP address. To test for this you can power off the PlayStation 2 development platform and ping the IP address. If you receive a response then the IP address is duplicated on your network.

- Check that the IP address or DNS name of the target is correctly specified.
- The PlayStation 2 Development Tool may simply need to be rebooted.

Disconnecting from targets

If you wish to disconnect from the target because you have finished working with it, then you will need to do this in the Target Manager.

To disconnect from a target

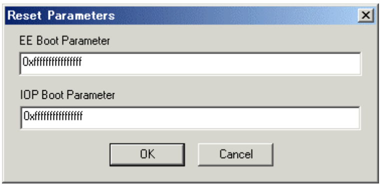
1. Select the connected target that you want to disconnect from in the right hand list of the Target Manager main window.
2. Double-click on the target name, or click **Disconnect** in the **Target** menu.

   The target properties will update to show 'Disconnected' in the **Status** column.

Resetting the target

The target can be reset using the **Reset** button. A dialog appears in which you can set the boot parameters for the EE and IOP units separately.

For more information on the boot parameters, refer to the Sony PlayStation 2 Developer Tool documentation.


Reset Parameters

EE Boot Parameter

0xffffffffffffffff

IOP Boot Parameter

0xffffffffffffffff

Click **OK** to reset the target.

4 Building your game

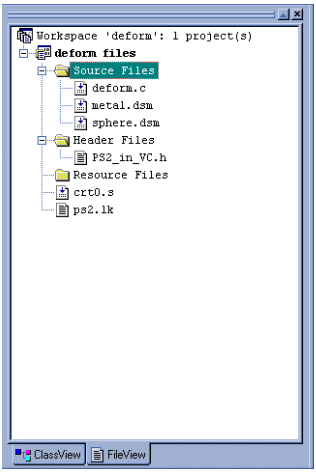
The ProDG Build Tools for PlayStation 2 enable you to build your game titles for the PlayStation 2. You can either build using the Visual Studio 6.0 IDE, or you can access the tools directly from the command prompt.

Visual Studio 6.0 integration

This section describes how to create and configure a PlayStation 2 EE project in Visual Studio 6.0 to build the Sony sample file deform.elf using the ProDG tools.

1. Start Microsoft Visual Studio 6.0. Click **Customize** in the Tools menu, select the **Add-ins and Macro Files** tab, and in the dialog that appears select the **ProDG for PS2 Developer Studio Add-in** to enable it.
2. Click **New** in the **File** menu and click the **Projects** tab. The New Projects dialog is displayed.
3. Select the PlayStation 2 AppWizard **PlayStation 2 EE Project (ELF)**.
4. Enter 'deform' in the **Project name** box.
5. Specify the location as: \usr\local\sce\ee\sample\vu1.

   Note that the directory deform should be automatically appended to the end of the directory path.
6. Once the project has been created, in the file view add the files deform.c and metal.dsm and sphere.dsm to the Source Files folder. The files pane should now resemble the following:


Workspace 'deform': 1 project(s)
deform files
Source Files
deform.c
metal.dsm
sphere.dsm
Header Files
PS2_in_VC.h
Resource Files
crt0.s
ps2.lk

7. Click **Build deform.elf** in the **Build** menu, and deform.elf should be successfully built.

Building demo programs using make

If you choose not to use Microsoft Visual Studio 6.0, you can also build your programs directly from the command prompt using make.

The directories in \usr\local\sce\ee\sample contain the Sony EE demo programs.

1. To build a particular sample, navigate to the required subdirectory below \sample.

   Note: The makefile in this directory is written for Linux and must be modified to work under Windows. You will need to download a corresponding Windows makefile for this sample program from the SN Systems website (look in the downloads table for ProDG for PlayStation 2 in the Support area).

2. Copy the Windows makefile over the Linux makefile.

3. From the command prompt type 'make' and the sample ELF file will be built.

Building for debugging

To use your C/C++ program with the ProDG Debugger you must:

1. Remove compiler optimizations, i.e. delete the -Ox entry from the CFLAGS= and/or CXXFLAGS= variables in your makefile.
2. Set the debug information flag. Add the -g switch to the CFLAGS= and/or CXXFLAGS= variables in your makefile.
3. If you wish to do profiling or VU debugging, you will need to include libsn.a, preferably as the first included library in your LIBS= variable.

Build tools reference

The following table lists some of the more important tools:

Program	**Description**
cc1.exe	GNU C compiler, normally called by ps2cc. Files sent to cc1 must be preprocessed with cpp first.
cc1plus.exe	GNU C++ compiler, normally called by ps2cc. Files sent to cc1plus must be preprocessed with cpp first.
cpp.exe	GNU compiler pre-processor
ps2cc.exe	ProDG compiler driver
ps2dvpas.exe ps2eeas.exe ps2iopas.exe	ProDG DVP, EE and IOP assemblers
ps2ld.exe	ProDG linker

For more information on using ProDG build tools, including the compiler driver, assemblers and linker, refer to the *ProDG for PlayStation 2 Power User's Guide*, available on the ProDG for PlayStation 2 CD-ROM.

Tip for optimizing your ProDG builds

This section suggests some ways to optimize your ProDG for PlayStation 2 builds. In addition you should check the technical support section of the SN website for further details of some of these optimizations.

Optimization trade-offs

The compiler offers many switches for the optimization processes during builds. For simplicity these are also grouped under the '-O' switch which sets a global optimization level. An optimization level of '-O2' is generally referred to as 'optimize for size', while '-O3' is termed 'optimize for speed'. However these general terms can be misleading when applied to programs destined for games consoles.

The major difference between the settings is the amount of inlining of code performed. On an ideal machine with infinite RAM, inlining would always produce faster but larger code. However, games consoles usually have very limited RAM and rely on memory caches for performance. Due to this, generating inlined, larger, code can actually run slower than non-inlined code due to cache misses. Therefore care should be taken when choosing between -O2 and -O3. In general, compilation modules usually perform best at -O2 with only specific modules benefiting from -O3.

If you are inlining a lot of functions, the compiler switch -fno-implement-inlines can be used to prevent the generation of non-inlined copies of the function. However, if your code does call non-inlined versions of these functions, at least one compilation module must includes instances of them or you will get link time errors.

The GNU compiler also uses an extension to the language to indicate that no non-inlined instances should be generated. If functions are declared 'external inline' in a header they will not generate a non-inline instance.

Optimizing for small ELF size

In C++ if you are not using run time type information (RTTI) or exceptions use the -fno-rtti and -fno-exceptions compiler switches to prevent the generation of code for supporting these language features.

The following linker options can help reduce generated ELF size:

- -s or -strip-all will remove all symbol information when generating the ELF.
- -S or -strip-debug will omit debug information (but not symbols) from the generated ELF.
- -s-lib removes debug info from the libraries but leaves it in the object files.
- -strip-unused will strip unused code from the ELF. This can be very beneficial on debug builds of code with a lot of template usage, as this will remove the duplicate instances of template methods (this is achieved automatically on non-debug builds).

Optimizing for build speed

Taking some care when planning your source project can reduce build times. The most critical area is in the usage of include files. If your project is structured so that compilation modules only include the minimum number of headers this can drastically reduce compilation time, link time and debugger load time for debug builds. For every type contained in header files, the compiler generates debug information. If your modules include a lot of unnecessary headers this will enlarge the debug data. This in turn leads to longer link times and slower loading of program into the debugger.

If you suspect your compilation modules are including unnecessary types you can use the compiler's debug optimization switch ('-opt-stabs' under ProDG for PlayStation 2), however this can increase compilation time slightly but usually the faster link time offsets this penalty.

Another header area that can cause problems in both debug and release builds is the use of templates in C++. If your compilation modules include a lot of inter-dependent template code, each compilation can incur an overhead generating the instances of template methods. Keeping the use of headers containing templates to a minimum can improve build speed.

Use of map or debug log linker options can increase build time.

Maximize system resources

For fast builds the memory usage should not be greater than the amount of physical RAM you have installed. If it does go above this value then it would be worth installing more RAM.

- If you are using a recent Windows version (e.g. Windows 2000) you can check how much memory is being used by checking the Windows Task Manager, Performance tab.

Disabling other applications and services can greatly speed up build times. For example:

- Close your e-mail client and personal web server.
- Close all file browser and web browser windows. File browsing windows that view files that are being updated or created can slow down a build considerably.
- Close any other applications not required during the build.
- Disable sharing of your drives on the network.
- Disable virus checkers.

Note: Always ensure any changes you make are acceptable to your system administrator, especially if disabling a virus checker. Always make a backup of your system before making hardware or software changes.

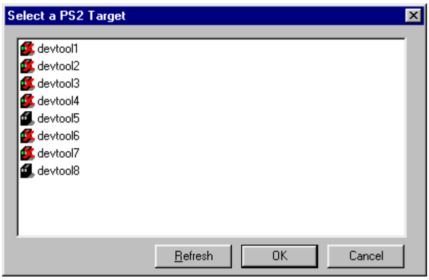
5 Debugging your game

Launching the ProDG Debugger

ProDG Debugger can be launched from the command prompt (by calling ps2dbg.exe), or via the Start menu, or from within Visual Studio 6.0. This section describes the default behavior of the debugger when it is called from the command prompt. The section "Debugging from Visual Studio" on page 27 describes how to use the debugger when using the Visual Studio 6.0 integration.

Connecting to targets

When you start the Debugger the Target Manager is also started, and a dialog appears showing the targets that you have set up:



Targets which are in use are shown with a red cross, whereas targets which are available are shown in black. Targets to which you are already connected have the LEDs highlighted on the target icon.

1. Select an available target from this list and click **OK**.

   The Debugger attempts to connect to the new target.

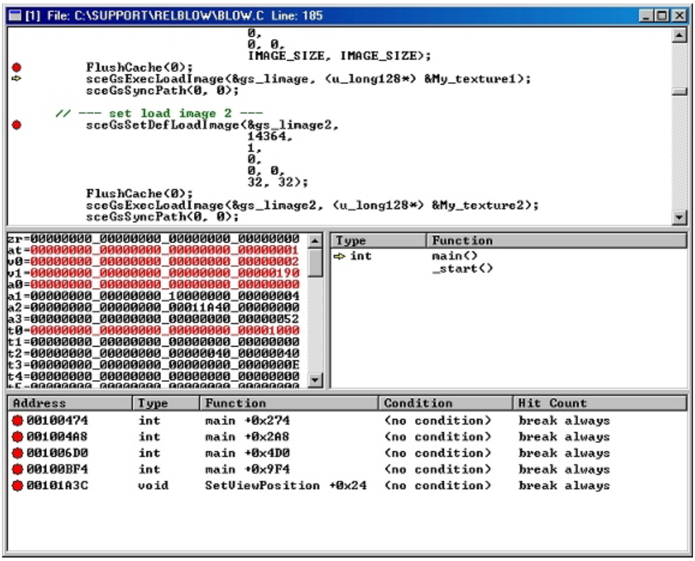
2. If the Debugger cannot connect to the new target then a dialog appears telling you that connection failed.

3. Once you have selected a target session, the next time the Debugger is started it will automatically connect to the target that you were working on when you quit the Debugger (provided the dbugps2.ps2 configuration file has not changed).

4. You can change the target that you are working on while using the Debugger using the **Select Target PS2** option from the **Debug** menu.

Windows and panes

ProDG Debugger is made up of one or more *windows.* Each window may contain one or more different *panes* for viewing different types of information. If there is more than one pane in a window it is a *split pane view*, as in this example:



```
                                        0,
                                        0, 0,
                                        IMAGE_SIZE, IMAGE_SIZE);
        FlushCache(0);
        sceGsExecLoadImage(&gs_limage, (u_long128*) &My_texture1);
        sceGsSyncPath(0, 0);

    // --- set load image 2 ---
        sceGsSetDefLoadImage(&gs_limage2,
                                        14364,
                                        1,
                                        0,
                                        0, 0,
                                        32, 32);
        FlushCache(0);
        sceGsExecLoadImage(&gs_limage2, (u_long128*) &My_texture2);
        sceGsSyncPath(0, 0);
```

```
zr=00000000_00000000_00000000_00000000
at=00000000_00000000_00000000_00000001
v0=00000000_00000000_00000000_00000002
v1=00000000_00000000_00000000_00000190
a0=00000000_00000000_00000000_00000000
a1=00000000_00000000_10000000_00000004
a2=00000000_00000000_00011A40_00000000
a3=00000000_00000000_00000000_00000052
t0=00000000_00000000_00000000_00001000
t1=00000000_00000000_00000000_00000000
t2=00000000_00000000_00000040_00000040
t3=00000000_00000000_00000000_0000000E
t4=00000000_00000000_00000000_00000000
```

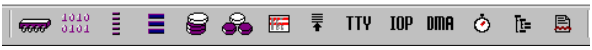
Type	Function
int	main()
	_start()

Address	Type	Function	Condition	Hit Count
● 00100474	int	main +0x274	(no condition)	break always
● 001004A8	int	main +0x2A8	(no condition)	break always
● 001006D0	int	main +0x4D0	(no condition)	break always
● 00100BF4	int	main +0x9F4	(no condition)	break always
● 00101A3C	void	SetViewPosition +0x24	(no condition)	break always

The layout of windows and panes is saved in the Debugger configuration file at the end of each debug session.

Debugger pane types

A new window containing one of the Debugger panes can be opened by using the main toolbar buttons:



Reading from left to right, these buttons open the following Debugger panes:

Registers view	View registers. It also shows the program counter and the status of the target.
Memory view	View memory.
Disassembly view	View disassembly. It also shows the instruction that the program counter is set to, and you can set breakpoints and single-step execution.
Source file view	View the source of the program that is currently running. The current program counter is shown and you can set breakpoints and single-step through your source running on the target unit.
Local variables view	View the values of all the local variables in the current function. You can expand or close the display of any structures or arrays using the pane shortcut menu.
Watch view	View a selected set of variables that you wish to track. You can add or remove watches, and expand or close the display of members of any watched structures or arrays.
Breakpoint view	View all the breakpoints that have been set. They are indicated by the address of the line of source or disassembly in

	memory.
CallStack view	View the function calls on the call stack that have been made to arrive at the current position in the program. The most recently called function is shown at the top of the call stack. You can change the Debugger context by selecting a different function call.
TTY console view	View any standard output generated.
IOP modules view	View a list of IOP modules loaded.
DMA view	View data sent to a DMA channel.
Profile view	Produce a basic profile of main CPU usage, so that you can see which processes are taking most time.
Workspace view	Create a workspace for viewing in a tree view the ELF's source files, functions, variables and classes.
Kernel view	View information about all the threads running on the target.

Debugger pane update

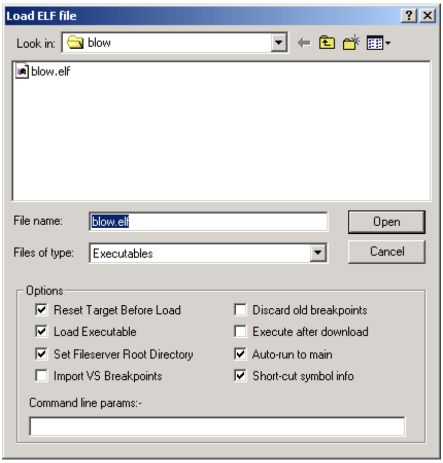
	The **Update all views** button performs a one-time refresh of all pane contents.
	You can enable auto-updating by pressing the **Toggle auto-update** button. This causes the panes to be continually refreshed by polling the target for up-to-date information. The **Toggle auto-update** button will appear to be depressed when the auto-update feature is ON. Press the button again to turn this feature OFF.

Loading and running ELF files

You can load and run ELF files on the target directly from the Debugger.

To load and run an ELF file

1. Click **Load ELF** file in the **File** menu.
2. In the Load ELF File dialog select the ELF file that you wish to load and debug:



Load ELF file

Look in: blow

blow.elf

File name: blow.elf

Files of type: Executables

Open

Cancel

Options

- [✓] Reset Target Before Load
- [✓] Load Executable
- [✓] Set Fileserver Root Directory
- [ ] Import VS Breakpoints
- [ ] Discard old breakpoints
- [ ] Execute after download
- [✓] Auto-run to main
- [✓] Short-cut symbol info

Command line params:-

3. Set the ELF file load and run options:

Reset Target Before Load	Resets the target before the ELF file is loaded.
Load Executable	Loads application executable in the ELF file as well as the symbol information.
Set Fileserver Root Directory	Sets the application file serving directory to the ELF file directory.
Import VS breakpoints	Any breakpoints set in Visual Studio 6.0 are imported into the Debugger when the ELF file is reloaded.

Discard old breakpoints	Any breakpoints set by ProDG Debugger will be discarded.
Execute after download	Load and run the ELF file without stopping at main().
Auto-run to main	Load and run the ELF file to main() then stop.
Short-cut symbol info	Assume all types of the same name in different modules are identical. If you have different types with the same name in different modules then you should not check this option and enable safe symbol loading instead.
Command line params	Specify command-line parameters for your PlayStation 2 game.

4. Click **Open** to activate the load.

   The file is loaded and run to main() or executed on the target, depending on the option that you selected. You can now start debugging the newly loaded application ELF file.

Running your game on the PlayStation 2

Once you have started the Debugger and downloaded your ELF file, you can start it running on the target PlayStation 2. You can start and stop your application running at any time using the commands provided. This section describes how to just start and stop your application running using the target control commands in the **Debug** menu.

To start your application running

To start the application running from its current program counter position:

- Click **Go** in the **Debug** menu.

To stop your application running

If your application has not already stopped on a breakpoint or exception, then you can stop it manually at any time:

- Click **Stop** in the **Debug** menu.

You will notice that once your application has stopped running on the target, any other open panes will be updated to show the current information at the new program counter position.

Using the **Go to PC** commands in the source or disassembly panes you can view the current position of the program counter. Alternatively if you leave either the source or disassembly pane as the active pane, then it will automatically update to show the current program counter position when the target stops.

Note: The source pane will only show the program counter if it can successfully be mapped to the original source code.

To restart the target

You can restart the application at any time. This means that the target PlayStation 2 is reset, the ELF file is loaded again, and your application runs to main again.

- Click **Reset and Reload** in the **Debug** menu.

This is useful if you change the PlayStation 2 target, or just need to reset the target to its initial load state.

Breakpoints and stepping

The key to tracing bugs in your code is to maintain fine control over its execution on the target. This section describes how to set breakpoints in your code, and the different ways to step through your code.

- By setting a breakpoint in the Debugger, either in a source or disassembly view of the program, you can interactively stop the program running at any point in its execution path.

- If you have manually halted your application on the target or if it has stopped for another reason (breakpoint, etc.) you can then step through the execution path one line of code at a time.

Setting and viewing breakpoints

A breakpoint indicates that you want the execution of your program to stop when the program counter reaches the specified line of C, C++ or assembly code.

By default the position of breakpoints in your application code is saved between sessions in the dbugps2.ps2 configuration file.

To view breakpoint information

At any time all the breakpoints that have been set in your application source or disassembly can be viewed in the breakpoints pane. This lists the address at which the breakpoint has been set and the function that it is set in. You can add other breakpoints or delete the selected breakpoint, or all breakpoints in this pane.

To view the breakpoint settings:

- Click on the **Breakpoint view** toolbar button.

  This will open a new window containing a breakpoints pane.

To set a breakpoint in source or disassembly

1. Ensure that the target is stopped and open a source or disassembly pane.
2. Ensure that you are viewing the source or disassembly on the required PlayStation 2 target unit (via the shortcut menu).
3. Scroll to the line of source or disassembly that you wish to set the breakpoint on.
4. In the source pane, double-click in the vertical border on the left of the pane, or click **Breakpoint** in the shortcut

menu. If there is already a breakpoint there it will be toggled off. Otherwise the new breakpoint will be set on this line and indicated by the line being set in a different color.

Note: If you try to set a breakpoint on a line that the program counter cannot halt on (e.g. a comment), then the breakpoint will be set on the next valid line of code after the selected line. In addition if you cannot correctly set a breakpoint, or the application stops at the breakpoint in the wrong part of the program, it could be that optimization may have been used when compiling the source code. To get around this problem you will need to rebuild the application and remove the -Ox flag from the compiler arguments (in the makefile).

Now when you start the target, it will stop when the program counter reaches a breakpointed line, and any debug information panes will be updated. In addition if a disassembly or source pane is open and active it will be updated with the current program counter position indicated.

Stepping through your program

You can single-step through your program executing on the PlayStation 2 target. This can either be done in disassembly or source (if you can view the source at the current program counter).

To single-step through your program

1. Ensure that the target is stopped and open a source or disassembly pane.
2. Use the **Step** or **Step Over** commands in the pane shortcut menu to single-step your source or disassembly. The **Step** command will step into any function calls and step through the lines of code in the called function.

Note: You are also able to use the **Step** and **Step Over** commands in the **Debug** menu. If the active pane is a disassembly pane then you will step through disassembly. If the active pane is a source pane, you will step through source code. Otherwise these commands will attempt to step through whichever of source or disassembly was last activated.

There are other **Run** possibilities in the shortcut menus of the disassembly and source panes. For example you can run to the current cursor position using the **Run to Cursor** command.

To step out of the current function

If you are currently single-stepping through a function you can return to the line of source which called the function without having to step through every remaining line of code in the function.

- Click **Step Out** in the **Debug** menu or **Step out of current function** on the toolbar.

  The program counter will normally advance to the line of code following the function call. However, if there are any breakpoints in the code up to the end of the function (or in any functions called before then), the pane may switch to halt on the next breakpointed line.

Viewing program source

Whenever the target stops running, if a source pane is open it updates to try and show the current position of the program counter in your application source.

To view your program source

If the current program counter position can be viewed in the source code, you can see it simply by opening a source pane.

- Click on the **Source file view** button in the toolbar.

  A new window containing a source pane will be opened and the current program counter will be indicated with a ➡ character. If the program counter cannot currently be viewed in the source, or if the source file cannot be located, the pane will appear with the message "No Source File Loaded" showing.

Note: You can use the **Go to PC** option in the source pane shortcut menu to quickly move the display to show the new program counter position.

To navigate through your program source

You can view different parts of the source much as you would navigate through the source file using a text editor.

- Use the standard Windows shortcuts <PgUp> and <PgDn> to scroll through the source a page at a time.
- Use the <arrow keys> to scroll through the source a line at a time.

To view another source file

As well as viewing the line of source code that is currently being executed on the target you can also view any other source file.

1. Click **Load Source File** in the **File** menu.
2. Locate the required source file in the dialog that is displayed.

   A new source pane is opened to display the contents of the selected source file. This can be used in the same way as the source pane that contains the program counter, and you can set breakpoints in this file and browse through it as required.

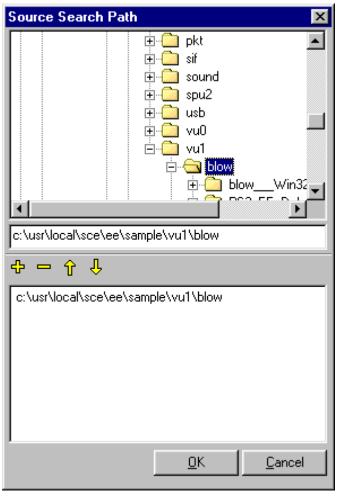
Locating source files

The Debugger is able to locate the application source file that needs to be opened, if it is found in the directory where your ELF file was built. If the source file has been moved then you must add a search path to the Debugger.

To locate source files

1. Click **Source Search Path** in the **Debug** menu.

   The Source Search Path dialog appears, like this:



c:\usr\local\sce\ee\sample\vu1\blow

c:\usr\local\sce\ee\sample\vu1\blow

2. In the dialog that appears use the tree view to select the full path(s) of any directories where the Debugger should look for your application source files. You can add a selected directory to the search path by clicking on the [+] symbol, or remove a selected directory by clicking on the [–] symbol.

3. The order of directories in the search path can be altered by clicking on the up and down arrow symbols. To promote the selected directory click on the up arrow, to demote a directory click on the down arrow.

4. You may need to close any existing source file view and open a new one for the change to take effect, and the source file to be located and opened.

Viewing program disassembly

Whenever the target stops running, if a disassembly pane is open it updates to show the current position of the program counter in your application disassembly (indicated by the ➪ character).

To view your program disassembly

If you wish to open a new disassembly pane that will show the current program counter position:

1. Click on the **Disassembly view** toolbar button.
2. A new window containing a disassembly pane is opened. This shows a disassembly of the code currently being run on the selected PlayStation 2 target unit. You can change the unit disassembly being viewed using the shortcut menu, and use the **Go to PC** command to view the program counter.

Viewing and modifying registers and memory

You can view all of the DTL-T10000 registers and memory in the registers and memory panes respectively. You can also modify the values of any register or memory address.

To view and modify registers

To view the values of registers:

- Click on the **Registers view** toolbar button.

  This will open a new window containing a registers pane.

To view and modify memory

To view the value of memory addresses:

- Click on the **Memory view** toolbar button.

  This will open a new window containing a memory pane.

Debugging from Visual Studio 6.0

In Visual Studio 6.0, ProDG Debugger can be invoked, once you have successfully built your project, by clicking the **Run ProDG Debugger for PS2** toolbar button:

The Debugger will be launched if the project was built with one of the PS2 EE/IOP build configurations as the active project configuration AND the ELF file has been successfully built.

Loading and running ELF files

You can load and run an ELF file by clicking the **Load ELF** toolbar button:

Setting breakpoints in your project

When called from the Visual Studio 6.0 integration, ProDG Debugger automatically imports any breakpoints that have been set in your source in Visual Studio.

While ProDG Debugger is running, setting or modifying breakpoints in Visual Studio has no effect on ProDG Debugger breakpoints, even if the ELF file is rebuilt in Visual Studio and re-downloaded. However, any breakpoints set in ProDG Debugger will be immediately reflected in Visual Studio. We therefore recommend that you set or modify breakpoints only in the ProDG Debugger source or disassembly panes. These breakpoints will be automatically updated in your source files as viewed by Visual Studio.

When you exit ProDG Debugger, all of the breakpoints still set will be exported back to Visual Studio.

Editing your source in Visual Studio 6.0

Any time you are debugging your project in ProDG Debugger you can switch from the source pane to the Visual Studio 6.0 editor. This can be done by moving to the part of the file that you wish to edit, and clicking **Edit in Visual Studio** from the source pane shortcut menu.

Any changes you make to the source will need to be rebuilt into a new ELF file. You can then either restart ProDG Debugger from Visual Studio 6.0 by clicking the **Run ProDG**

Debugger for PS2 toolbar button or download the newly built ELF file using the ProDG Debugger **Load ELF file** option (**File** menu).

Note that setting or unsetting breakpoints in Visual Studio 6.0, *while ProDG Debugger is running*, will have no effect.

Basic EE profiling

For information on basic EE profiling and using the profile pane, refer to the *ProDG for PlayStation 2 Power User's Guide*, available on the ProDG for PlayStation 2 CD-ROM.

IOP debugging

For information on IOP debugging and using the IOP modules pane, refer to the *ProDG for PlayStation 2 Power User's Guide*, available on the ProDG for PlayStation 2 CD-ROM.

VU and DMA debugging

For information on VU and DMA debugging, and using the DMA pane, refer to the *ProDG for PlayStation 2 Power User's Guide*, available on the ProDG for PlayStation 2 CD-ROM.

Accessing other debugger features

For information on using other pane types: the locals, watch, call stack, TTY, kernel and workspace panes, refer to the *ProDG for PlayStation 2 Power User's Guide*, available on the ProDG for PlayStation 2 CD-ROM.

6 Enhanced Debugger features

Enhanced ProDG Debugger features are exclusively available to ProDG Plus customers. This section describes which features are available in this release.

Scripting

The ProDG Plus script interpreter is a C program which you can modify as the ProDG Debugger is running. It uses a new *script pane* type to allow you to quickly and interactively enter programs to perform different tasks. These can be one-off 'execute until completion' type scripts or they can be bound to target and debugger events like exceptions, window updates and keystrokes. Scripts can access the target console, for example to fetch memory contents, and they can do console output to a debugger pane.

The scripting component comprises a C interpreter, built into the Debugger, that supports most ANSI C, with access to a number of special built-in functions to allow the scripts to interface with the Debugger and the target console. When scripts are loaded (or immediate mode script functions are executed) the script is compiled to a byte-code form which is then interpreted.

The ProDG Plus script interpreter has many advantages over normal Windows scripting or compiled-code approaches. The script interpreter can typically execute one million C statements per second, so it is *much* faster than trying to use COM interfaces to control an application. Being written in C, the scripting interface is very familiar to C programmers.

For more information on scripting, refer to the *ProDG for PlayStation 2 Power User's Guide*, available on the ProDG Plus for PlayStation 2 CD-ROM.

Index