



**HUNT ENGINEERING**  
Chestnut Court, Burton Row,  
Brent Knoll, Somerset, TA9 4BP, UK  
Tel: (+44) (0)1278 760188,  
Fax: (+44) (0)1278 760199,  
Email: sales@hunteng.co.uk  
<http://www.hunteng.co.uk>  
<http://www.hunt-dsp.com>



## Getting started with the HUNT ENGINEERING Server/Loader and HERON products.

Rev 3.0 P.Warnes 04-3-02 (changed to include HEART based carriers)

The HUNT ENGINEERING Server/Loader tool provides a way to load a multi-processor network with separate coff files that have been generated using the TI Code Generation Tools. Beginning with version 3.4, the Server/Loader also supports downloading bit streams to FPGA/HERONIO modules. It uses a simple text file that describes the processors and FPGA/HERONIO modules, how they are connected, which programs to load onto each processor, and which bit streams to load onto what FPGA/HERONIO module.

This document is provided as a “quick start guide” and not a replacement for the full user documentation accessed through the “user manuals” section of the HUNT ENGINEERING CD.

## **Installation**

During the HUNT ENGINEERING software installation, (accessed from the “Install Drivers and Tools” option of the HUNT ENGINEERING CD) you will be asked to enter your Server/Loader password. With your system you should have received an envelope that contains this password.

The HUNT ENGINEERING installation program will set an environmental variable HESL\_DIR to indicate which directory the Server/Loader was installed into. This is a sub-directory of your main API installation, for example “c:\heapi\hesl”, if you installed the API into “c:\heapi”.

This installation will install CCS “plug-ins” and register them in the windows registry. This means that when Code Composer Studio is installed it will automatically detect the plug-ins.

When you open Code Composer Studio the tools menu will contain an entry for the HUNT ENGINEERING Server Loader. If it does not there are instructions on how to make the installation “by hand” in the User manual for the Server/Loader.

## **DSP/BIOS**

DSP/BIOS is the multi-threading environment provided as part of the Code Composer development Environment. It also provided services for configuring processor features such as hardware interrupts and timers.

As it is included in Code Composer Studio (CCS), along with the Compile tools for the C6000, all users of HERON hardware will be able to use it.

This is used to configure the multi-tasking etc at the processor level. Simply program and use CCS to compile the application for each DSP separately. This results in a single .out file that contains all of the interrupt service routines etc for that processor.

The HUNT ENGINEERING Server/Loader can load any .out file, so DSP/BIOS features can be embedded in the programs loaded using the Server/Loader.

## **HERON-API**

HERON-API is the communications library that HUNT ENGINEERING provides to perform the inter-processor and processor to I/O communications. Its purpose is to prevent the user from needing to intimately understand the communications mechanism, by providing an optimised way to use the limited DMA resources of the C6000 in a choice of ways.

It also serves the purpose of providing a common software interface to the various C6000 HERON modules that HUNT ENGINEERING produce or plan to produce. The hardware of those modules will be different but the HERON-API interface will not.

For details of how to use the features of HERON-API refer to the user documentation for HERON-API. HERON-API is compatible with and relies upon DSP/BIOS.

The HERON-API library is compiled into the .out file, so can be used in any application that is loaded using the HUNT ENGINEERING Server/Loader.

## **Command line Vs library**

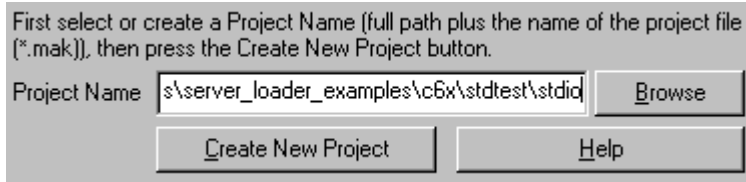
A user who simply needs to load their system with their application program, and perhaps allow the application program to have access to the Host machine’s Standard I/O system should simply use the pre-compiled command line versions of the tool.

Advanced users however might wish to load the DSP system with an application that then communicates with an application program on the HOST machine. To do this the Host application program can use the library version of the loader, and then communicate with the DSP system using the Host API.

## Starting

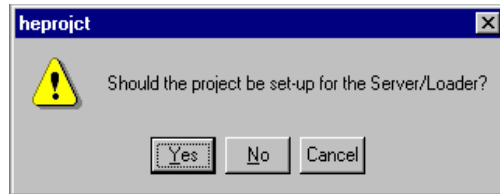
The best way to become familiar with the Server/Loader is to look at the examples provided in the “server\_loader\_examples” directory of the CD. This contains simple examples that show the use of the STDIO library and loading 2 and more processors, and even host program examples for customised server/loader programs and programs that use the loader library, and then perform their own communications with the DSP via the Host API.

When making a new project for use with the Server/Loader, you should use the “Create new HERON\_API project” plug in found in Code Composer Studio under Tools→HUNT ENGINEERING→ Create new HERON\_API project.

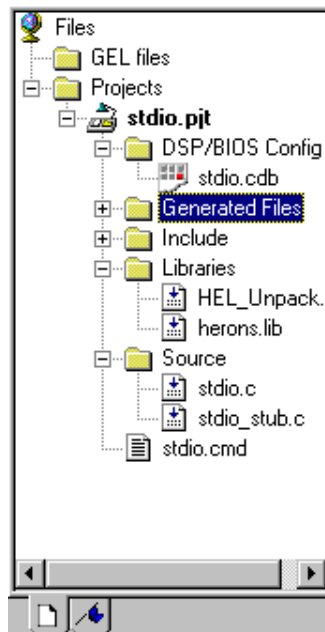


How to use this plug-in is explained in detail in the “Starting your Development” document found on the HUNT ENGINEERING CD (Getting Started → to start using C6000 modules and tools look here → Starting your Development), named “starting\_development.pdf”.

The plug-in will ask you several questions. One of the questions is whether you want to create this new project for use with the Server/Loader. You need to choose that you are making a project for use with the Server/Loader, i.e. answer “Yes” when you see a window as below appear.



Note that when creating a new project for the Server/Loader, an extra “stub” file is added to your project. This “stub” file makes a connection between the HERON-API library and the Server/Loader library. (This way the Server/Loader library is independent from the HERON module type you use).



## Requirements for the C program

Once the system is loaded, the Server/Loader has no effect on the DSP program, unless you have linked in the Standard I/O library. This is literally a library, and performs no control functions.

In order to allow the whole system to be loaded it is necessary to call the `bootloader()` function at the beginning of each program. This allows other DSPs to be loaded “through” this one, so it must be called as the very first thing to execute on this DSP. A good place to call `bootloader()` is in the `main()` routine of your program. With DSP/BIOS, the `main()` routine is used for initialisations. No DSP/BIOS task or thread is running yet, and all interrupts are disabled. Calling `bootloader()` in `main()` thus avoids possible concurrency problems (like one of your tasks trying to communicate with another DSP that hasn’t been booted yet).

```
main()
{
    bootloader();
}
```

The processor that is connected to the host can call “stdio” functions. That is, the DSP will send messages to the host (i.e. the Server/Loader) to ask the Server/Loader to execute calls like “printf”, “fwrite” and “fread”. To use this functionality, you must `#include` the “`stdioc60.h`” file, as follows.

```
#include <stdioc60.h>          /* use Server/Loader stdio calls */
#include <string.h>
#include <std.h>
#include <swi.h>
```

In order to use the stdio library from the Server/Loader it is necessary to include the Server/Loader `stio62x.lib` ahead of the `rts6201.lib` in the project. The way to do this is to include it in the linker command file (`*.cmd`). In this file there is a line to include the standard I/O library. When using the “Create new HERON-API project” plug-in, this will already have been done for you. Had you created a new project by hand, you must have added a `cmd` file to your project. (Use the template “`heronx_slbios.cmd`” (where ‘x’ is the HERON module types, e.g. ‘1’ for HERON1 and ‘4’ for HERON4)). For more information on “linker command file”, please refer to TI documentation.

For example, the “linker command file” as created by the “Create new HERON-API project” plug-in would be as follows.

```
-l c:\heapi\hesl\lib\stio62s.lib
-l stdiocfg.cmd

SECTIONS
{
    heronapi_code          >          SBSRAM
    heronapi_data          >          IDRAM
}
```

The first line tells the TI linker to use the Server/Loader “`stio62s.lib`” library. The “`stdiocfg.cmd`” is created from the DSP/BIOS CDB file and project information when saving or creating the CDB file. The SECTIONS part tells CCS where to store HERON-API sections. You can change this mapping, if you wish or need to.

You can build your Server/Loader DSP application in the same way as one that does not use the Server/Loader. Because the Server/Loader adds extra software sections to IDRAM, you may find that at the linking stage CCS complains that there’s not enough memory to store all sections in IDRAM (e.g. “error: can’t allocate ‘.IDRAM\$heap’”). Such problems can be fixed by reducing the

size of software segments that are placed in IDRAM. Or by removing segments altogether and mapping them onto other physical memory. A quick way that works for Server/Loader examples is by reducing the DSP/BIOS heap in IDRAM. (Open the CDB file, open the memory manager (“MEM – Memory Section Manager”, in CCS 2.x first open “System”), right-click on “IDRAM”, select “Properties”. Change the “heap size” to, for example, 0x2000. Now rebuild.)

## **Switching between Server/Loader and Code Composer only projects.**

Code Composer Studio itself also supports “stdio” functions. These are executed via the JTAG and its output can be seen in a separate window in CCS. The Server/Loader uses the FIFOs instead, and execution of “stdio” calls is very much faster. Even then, you should always consider that execution of “stdio” functions is relatively slow. In a real-time program where every microsecond counts you don’t want to bother with “stdio” calls. But “stdio” calls may be very useful for debug phases, non-real-time or part real time programs (for example: take a real-time snapshot of e.g. 1 second, then write the data to harddisk), or perhaps it can be used in a low-priority separate task.

You can switch between CCS “stdio” functions and Server/Loader “stdio” functions by means of the header file. In the above, “stdio60.h” were used; hence Server/Loader “stdio” functions were called. To switch the above to use CCS “stdio” calls, simply change “stdioc60.h” to “stdio.h”.

```
#include <stdio.h>                /* use CCS stdio calls */
#include <string.h>
#include <std.h>
#include <swi.h>
```

Next, remove or comment out the call to bootloader(). Finally, change the order of libraries in the linker command file (\*.cmd) in the project. For example, in the “stdtest” example the linker command file is called “stdio.cmd”. Upon project creation the linker command file will have:

```
-l c:\heapi\hesl\lib\stio62s.lib
-l stdiocfg.cmd
```

Swap the ordering of these two so that the linker command file reads:

```
-l stdiocfg.cmd
-l c:\heapi\hesl\lib\stio62s.lib
```

You don’t need to add, remove or change libraries in your project. A rebuild will now be sufficient.

But note that this swapping between Server/Loader and CCS-only projects will only work if you had created a Server/Loader project in the first place. If you want to go the other way (create a Server/Loader project out of a project originally created for CCS-only) then (1) add a call to bootloader(), (2) change #include from “stdio.h” to “stdioc60.h”, (3) add a line to include “stio62s.lib” in the linker command file, and (4) add a stub file (there are template stub files in the API installation’s hesl\lib directory, called “stubx.c”).

## **Network file.**

The Network file is the text file that describes the system. For a full definition see the Server/Loader manual. It is here that the name of the \*.out file for each processor is defined, and the “bootpath” connections between the processors.

REMEMBER it is necessary to set the HERON-ID field to reflect the board slot number that the processor is plugged into. Failure to get this correct will prevent the system from booting.

## **HEART.**

For HEART based boards, such as the HEPC9, it is necessary to first create FIFO links between processors, FPGA/HERONIO modules, GDIO modules, and the host interface. The Server/Loader requires a full duplex FIFO link between a DSP processor (on which you wish to execute “stdio” calls) and the host PC interface. Such links can be created / defined using HEART statements in the network file.

When creating a new Server/Loader project, the “Create new HERON-API project” plug-in will automatically create a template network file. This template will assume 1 DSP module, and connect this module to the host interface with a duplex FIFO connection.

```
#ND Board Name Type CC-id HERON-id filename
c6 0 HERON ROOT (0) 00000001 stdio.out
pcif 0 host1 normal 0x05

# Create a connection between host (fifo 0) and the 'C6x (fifo 0). It uses
# 1 timeslot. The precise timeslot is chosen by Server/Loader or HeartConf
heart host1 0 heron 0 1

# And create a connection back, from the 'C6x (fifo 0) to host (fifo 0)
heart heron 0 host1 0 1
```

Therefore, when using the Server/Loader, there’s no need for a separate invocation of HeartConf.

The created network file can also be used by HeartConf. So, if for example you switch from a Server/Loader project to a non-Server/Loader project, you can use the Server/Loader network file to create FIFO links using HeartConf, whilst using CCS to boot the processors.

## **Running Server/Loader examples**

To run a Server/Loader program, open a DOS box, then browse to the example directory – the one that you have built a Server/Loader project for. There’s a batch file usually called “w32.bat” that you can use to run the example. The “mysl” and “sl\_api” examples use “my32.bat” (using the Microsoft Server/Loader library) or “bl32.bat” (using the Borland Server/Loader library) instead.

Make sure that all processors in the system are running in “RunFree” mode. (You can keep CCS running, but do a “Debug → RunFree” in all processor windows.) If you fail to do so, CCS will keep controlling the DSPs and the Server/Loader will be unable to boot any processor.

## **Debugging Server/Loader applications**

DSP applications for use with the Server/Loader will be configured and built using Code Composer Studio. Code Composer Studio has some excellent debug facilities, but special considerations are necessary when using these in conjunction with the Server/Loader. Please follow the example presented in “using Server/Loader with CCS” to learn about those considerations.