

# **HelloDevice Super Series**

# **User Customization Guide**

**Version 1.1.0**

**2004-01-12**

### **Copyright Information**

Copyright 1998-2004, Sena Technologies, Inc. All rights reserved.

Sena Technologies reserves the right to make any changes and improvements to its product without providing prior notice.

### **Trademark Information**

HelloDevice™ is a trademark of Sena Technologies, Inc.

### **Technical Support**

Sena Technologies, Inc.

210 Yangjae-dong, Seocho-gu

Seoul 137-130, Korea

Tel: (+82-2) 573-5422

Fax: (+82-2) 573-7710

E-Mail: [support@sena.com](mailto:support@sena.com)

Website: <http://www.sena.com>

## Revision history

Revision	Date	Name	Description
V0.0.1	2003-09-09	H. Yeom	Initial Draft
V1.0.0	2003-09-22	H. Yeom	Initial Release
V1.1.0	2004-01-12	H. Yeom	GDB support and sample filter programs are added

# Content

1. Overview.....	5
2. SDK (Software Development Kit).....	5
3. How to build user programs?.....	6
3.1. Preparing .....	6
3.2. Coding .....	6
3.3. Uploading Files .....	6
3.4. Building and Executing .....	6
4. User Web Customization.....	6
4.1. HTML Files .....	6
4.2. CGI files .....	7
4.3. Java applets and so on .....	8
5. User Filter Customization.....	8
5.1. Understanding Filter Programs .....	8
5.2. Building Filter Program .....	9
5.3. Filter Samples .....	10
6. Debugging with GDB.....	11
6.1. Debugging sample filter program with GDB .....	11

## 1. Overview

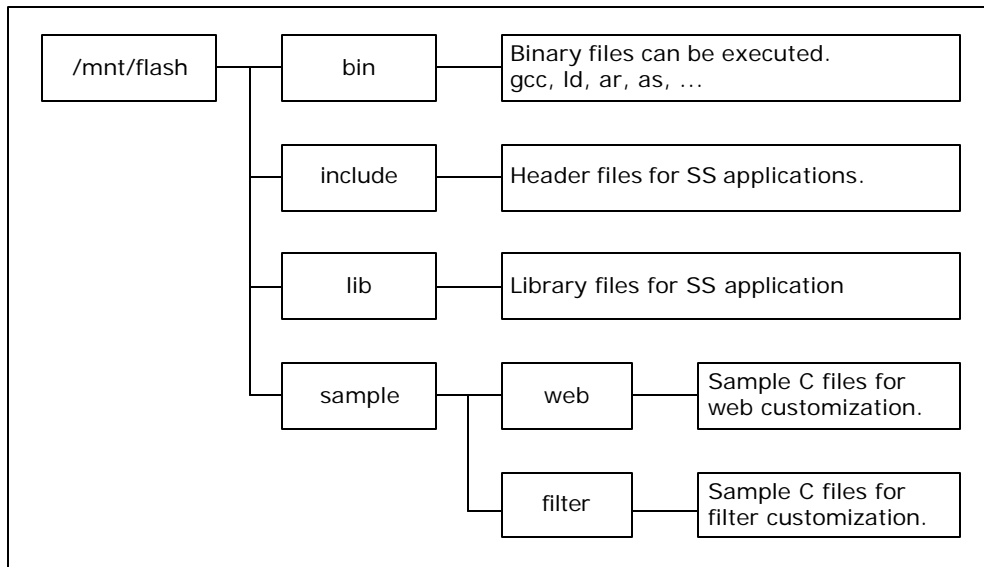
The HelloDevice Super Series is a serial to Ethernet programmable communication gateway for RS-232/422/485 based serial devices with various customization options.

Sena provides easy to use Software Development Kit (SDK) environment to quickly develop custom applications that run on the HelloDevice Super Series. Users can customize the web management interface, and integrate the programmed dynamic web pages to web menu.

In addition, user can manipulate the data stream which routes from serial device to socket and socket to serial device. The user-defined filter program communicates with other programs that are reading/writing serial port and socket by using FIFOs, and so user can easily manipulate the serial data without programming related to serial port and socket. The unit runs the embedded Linux Operating system and facilitates programming using command-line interface or one of several prepared scripts, using UNIX / Linux commands.

## 2. SDK (Software Development Kit)

To make user's own application code, SDK for SS110/400/800 is needed. SS110/400/800 SDK is provided in the form of PC CF card (Please contact Sena Technical Support to get SDK for SS110/400/800.). The SDK contains compiler, linker, library files, header files and some sample C files. The directory structure of the SDK is as follows.



### 3. How to build user programs?

#### 3.1. Preparing

Follow the below steps to prepare the development environment for customization.

- 1) Get a SDK for SS110/400/800 in the form of CF card.
- 2) Insert the SDK into the PCMCIA (PC card) slot of the SS device.
- 3) On the configuration menu, probe the PC card and then save and apply it.

#### 3.2. Coding

Write source files in the C language and Makefiles. You can edit the source files on the Super Series CLI but if it is uncomfortable, you can do it on PC or Linux box. Almost all the Linux libraries are located at /mnt/flash/lib directory.

#### 3.3. Uploading Files

If you edit the files on the Super Series CLI, skip over this section. To compile the source files, you must have uploaded the files to the SS device. You can upload the files in three ways as follows.

- SCP
- FTP
- Configuration menu

For more detail information, refer to the User Guide.

#### 3.4. Building and Executing

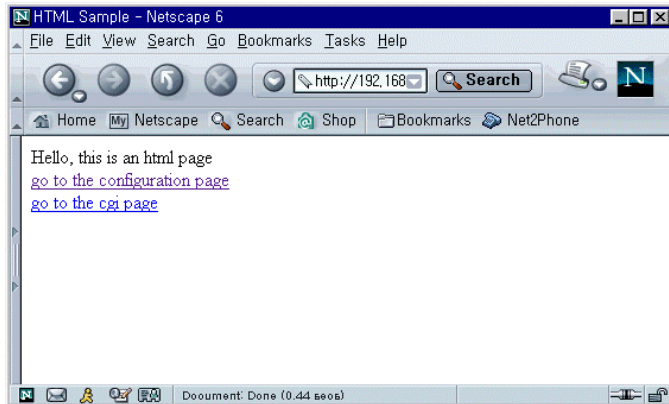
Compile, link and execute the programs as follows. (Imagine, a program named *prog* consists of two C source files *proga.c* and *progb.c*).

```
# gcc -o prog proga.c progb.c
# ./prog
```

### 4. User Web Customization

#### 4.1. HTML Files

You may customize the web management interface by adding your own HTML files in Super Series. All the HTML files must be in the /usr2/usrweb directory. There is a sample HTML file that is named index.html. Copy the file to /usr2/usrweb, and then you can see the HTML page like below.



## 4.2. CGI files

### 4.2.1 Building CGI files

Below is the procedure to build CGI files. Sample CGI source file is located at `/mnt/flash/sample/web/cgi/shell.c`,

**Step 1.** Build the CGI file.

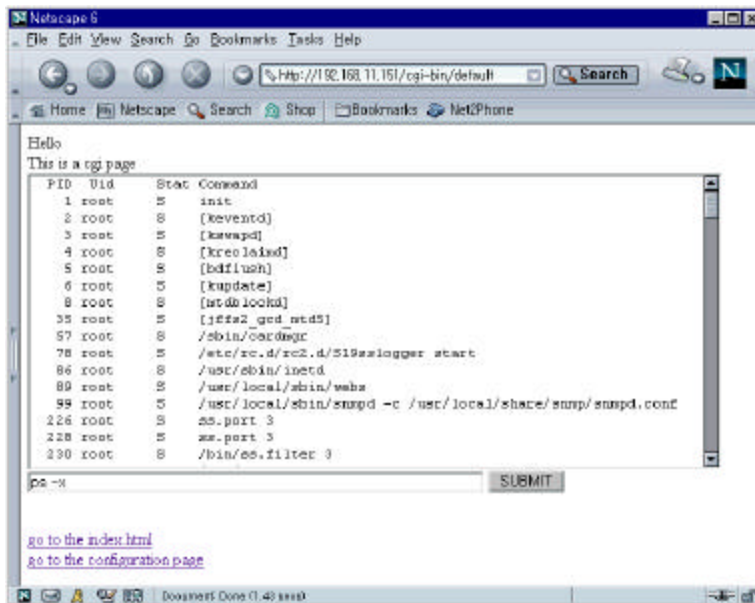
```
# cd /mnt/flash/sample/web/cgi
# make
```

**Step 2.** Copy the CGI file.

```
# cp shell.cgi /usr2/cgi-bin/
```

**Step 3.** Execute a Web browser, connect to the SS110/400/800, and log in.

**Step 4.** At the Web browser, go to `http://192.168.1.2/cgi-bin/shell.cgi`. (Suppose that the IP address of the SS110/400/800 is 192.168.1.2)



### 4.2.2 Using CGI files

The CGI files must be in the `/usr2/cgi-bin` directory. If there is a CGI file named default, the file can

be an entry page of the custom pages.

Here is a Makefile to make the shell.c.

```
CC = gcc
BIN = shell.cgi
OBJS = shell.o util CGI.o
LDFLAG = -L/mnt/flash/lib

BIN : $(OBJS)
    $(CC) -o $(BIN) $(OBJS) $(LDFLAG)
c.o :
    $(CC) -c $<
all : $(BIN)
clean :
    rm -f $(BIN) $(OBJS)
```

The util CGI.h and util CGI.c are attached at the Appendix A. These CGI programs must be in the /usr2/cgi-bin directory.

### **4.3. Java applets and so on**

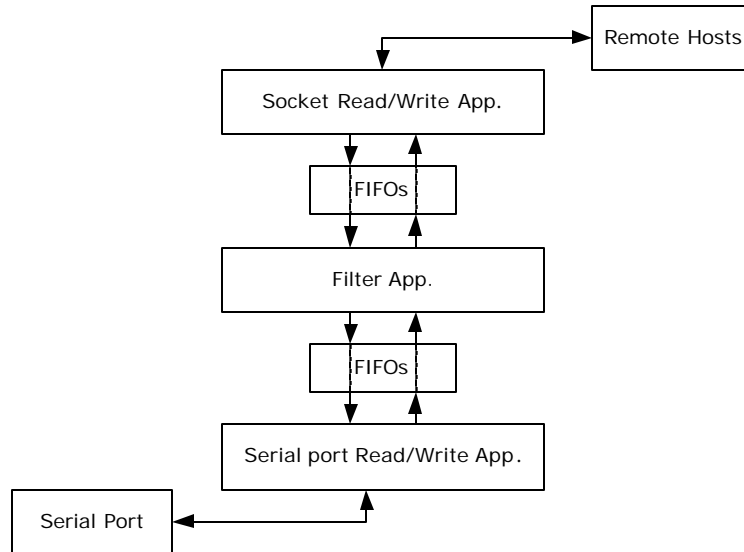
JAVA applets are also available. The applet files and all web files except CGI program must be in the /usr2/usrweb directory.

## **5. User Filter Customization**

### **5.1. Understanding Filter Programs**

SS110/400/800 uses FIFO method for inter-process communication. User can insert a program to manipulate the data stream which routes from serial device to socket and socket to serial device. The user-defined filter program communicates with other programs that are reading/writing serial port and socket by using FIFOs.





The user-defined filter reads a FIFO that is streaming the serial port data, manipulate the data and write the manipulated data to a FIFO that is sent to socket. The data stream that goes from socket to serial port can be manipulated in the same way. It must be satisfactory for following qualifications.

- 1) Write the data stream going to serial port to FIFO that is named  
`/tmp/port_fifos/portX_f2s` ( $X$  is the port index based on 1 ).
- 2) Write the data stream going to serial port to FIFO that is named  
`/tmp/port_fifos/portX_f2e`.
- 3) Read the datastream coming from serial port to FIFO that is named  
`/tmp/port_fifos/portX_s2f`.
- 4) Read the data stream coming from socket to FIFO that is named  
`/tmp/port_fifos/portX_e2f`.
- 5) Open four FIFOs on being executing and do not close the FIFOs except to be terminated.
- 6) Have more than one arguments and the first argument must indicate the port number.
- 7) Record the PID (Process ID) to the file that is named `/var/run/portX_filter.pid`.  
 (This PID file is used to terminate the filter application, but it is terminated only in case the port is disabled.)
- 8) Be terminated when it receives the SIGTERM signal.

## 5.2. Building Filter Program

Below is the procedure to build Filter program. The sample programs are located at

`/mnt/flash/sample/filter/` directory.

**Step 1.** Build a filter program.

```
# cd /mnt/flash/sample/filter
# make
```

**Step 2.** Copy one of sample filter programs to user space.

```
# cp data_conversion /usr2/sample_filter
```

**Step 3.** Configure the filter application settings as follows:

The screenshot shows a configuration window titled "Serial port configuration - 3 : Port #3". The window has a dark blue header with a dropdown menu on the right that says "--- Move to ---". Below the header, there are several menu items: "Enable/Disable this port", "Port title", "Apply all ports settings", "Host mode configuration", "Remote host configuration", "Port IP filtering", "Cryptography configuration", "Filter application" (which is highlighted), "Serial port parameters", "Modem configuration", "Port logging", and "Port event handling". The "Filter application" section contains two text input fields: "Filter application path :" with the value "/usr2/sample\_filter" and "Filter application arguments :" with the value "ps 100". Below these fields are three buttons: "Save to flash", "Save & apply", and "Cancel".

### 5.3. Filter Samples

All sample programs are composed of three threads, one main thread, one reading on serial port thread and one reading on socket thread. The main thread creates the other threads, and then waits until SIGTERM signal is received. If main thread receives SIGTERM signal, it cancels the other threads. The readings on serial port/socket threads are modified to its purpose.

- **empty.c**

This sample filter connects the serial port read/write application and socket read/write application with FIFOs. It does not manipulate the data.

- **periodic\_command.c**

This sample filter connects the serial port read/write application and socket read/write application with FIFOs and writes a command to serial port periodically.

- **data\_conversion.c**

This sample filter is an example for protocol conversion.

- **data\_calibration.c**

This sample filter is an example that calculates the average of periodically incoming serial port data (e.g. thermometer data, hygrometer data) and sends it.

- **data\_storing.c**

This sample filter is an example that watches serial port data (e.g. thermometer data, hygrometer data) and saves it to RAM disk.(under /tmp directory)

- **data\_event\_handling.c**

This sample filter is an example that watches serial port data (e.g. thermometer data, hygrometer data) and sends SNMP traps.

---

## - cq.c

Utility routines for circular queue.

## 6. Debugging with GDB

The SDK for HelloDevice Super Series supports GNU GDB debugger so that user can see what is going on inside user program while it executes.

( Please note that GDB support is added from the SDK v1.1.0 or later)

GDB can do four main kinds of things to help user catch bugs in the act:

- ☞ Start your program, specifying anything that might affect its behavior
- ☞ Make user's program stop on specified conditions
- ☞ Examine what has happened, when user's program has stopped
- ☞ Change things in user's program so user can experiment with correcting the effects of one bug and go on to learn about another.

GDB is invoked with the shell command `gdb`. Once started, it reads commands from the terminal until user tells it to exit with GDB command ***quit***. User can get online help from `gdb` itself by using the command ***help***.

User can run `gdb` with no arguments or options; but the most usual way to start GDB is with one argument or two, specifying an executable program as the argument.

Before run user program with GDB, user should compile his program with `-g` option.

### 6.1. Debugging sample filter program with GDB

This section describes how to debug sample filter program (`data_conversion.c`) with GDB debugger step by step. To debug sample filter program properly, it is recommended that user should set each parameters for serial ports previously using configuration tools such as Web UI or `editconf`. And also disable all serial ports using configuration tools so that user can run each port functions manually at the appropriate time.

#### Step 1. Copy sample program to user space

```
# cp /mnt/flash/sample/filter/Makefile /usr2/  
# cp /mnt/flash/sample/filter/data_conversion.c /usr2/  
# cp /mnt/flash/sample/filter/cq.c /usr2/  
# cp /mnt/flash/sample/filter/cq.h /usr2/
```

#### Step 2. Modify Makefile to compile program with `-g` option.

Add `-g` option to `CFLAGS` variable.

```
# cd /usr2/  
# vi Makefile
```

```
...
//CFLAGS = -pipe
CFLAGS = -pipe -g
...
```

**Step 3.** Modify source program to fit it to run with GDB debugger.

Remove `do_daemon()` and `save_pid(portnum)` functions calls.

Please note that user should run his program on the foreground to debug it with GDB.

```
# vi data_conversion.c
...
int main(int argc, char **argv)
{
    if (argc < 2) {
        fprintf(stderr, "\nUsage: %s [portnumber] [echo|no_echo]\n\n",
get_program_name(argv[0]));
        return -1;
    }

    portnum = atoi(argv[1]);
    if (argc>2 && !strcmp(argv[2], "no_echo")) echo_flag = 0;

    (void) signal(SIGTERM, handle_sigterm);
    (void) signal(SIGPIPE, handle_sigterm);

    // Remark following two lines to run this program on the foreground
    // do_daemon();
    // save_pid(portnum);

    do_filter();
    close_fifos();

    return 0;
}
...
```

**Step 4.** Compile sample filter program.

```
# make data_conversion
```

**Step 5.** Start serial port daemon(ss.port) and tcp socket daemon(ss.tcp) programs manually.

The `ss.port` is a daemon program for serial port and `ss.tcp` is a daemon program for tcp port.

These two programs are located under `/bin` directory of SS device.

Argument '1' means that it is run for port #1.

```
# ss.port 1
# ss.tcp 1
# ps -ef
  PID Uid   Stat Command
   1 root    S    init
   2 root    S    [keventd]
   3 root    S    [kswapd]
   4 root    S    [kreclaimd]
   5 root    S    [bdflush]
   6 root    S    [kupdate]
   8 root    S    [mtdblockd]
  35 root    S    [jffs2_gcd_mtd5]
  57 root    S    /sbin/cardmgr
  80 root    S    dhcpd eth0
  87 root    S    /etc/rc.d/rc2.d/S19sslogger start
  97 root    S    /usr/sbin/inetd
 100 root    S    /usr/local/sbin/webs
 110 root    S    /usr/local/sbin/snmpd -c /usr/local/share/snmp/snmpd.conf
```

```

113 root    S    /bin/linkupchecker -c 1
118 root    R    /etc/rc.d/rc2.d/S53ss800mand start
126 root    S    /etc/rc.d/rc2.d/S53ss800mand start
128 root    S    /usr/sbin/cron
129 root    S    -bash
523 root    S    ss.port 1
524 root    S    ss.port 1
525 root    S    ss.port 1
526 root    S    ss.port 1
527 root    S    ss.port 1
529 root    S    ss.tcp 1
530 root    S    ss.tcp 1
532 root    S    ss.tcp 1
533 root    S    ss.tcp 1
534 root    S    ss.tcp 1
535 root    R    ps -ef

```

**Step 6.** Run sample filter program using GDB.

```
# /mnt/flash/bin/gdb data_conversion
```

**Step 7.** After running GDB, user can use GDB commands to control it.

In this example, first user should set argument for sample filter program as follows,

```

GNU gdb 5.0
Copyright 2000 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "powerpc-hardhat-linux"...
(gdb) set args 1

```

Argument '1' means that this sample filter program is run for port #1 .

**Step 8.** And then set breakpoint at the proper positions to stop program run.

For example, user can set break point on \*e2s\_thread function to check what is entering from tcp port.

```
(gdb) break *e2s_thread
Breakpoint 1 at 0x10003054: file data_conversion.c, line 255.
```

And then run program with 'r' command

```

(gdb) r
Starting program: /usr2/data_conversion 1
[New Thread 539 (manager thread)]
[New Thread 538 (initial thread)]
[New Thread 540]
[New Thread 541]
[Switching to Thread 541]

Breakpoint 1, e2s_thread (arg=0x0) at data_conversion.c:255
255  {
(gdb)

```

**Step 9.** After for a while, program will be stopped at the entry point of \*e2s\_thread function.

User can run next step by entering 'n' (next) command.

```
(gdb) n
e2s_thread (arg=0x0) at data_conversion.c:257
257         int nread=0;
(gdb) n
261         pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
(gdb) n
262         pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
(gdb) n
...
```

**Step 10.** When program run is reached at the line #274, user should make a tcp connection to a serial port #1 of SS device and send any data to serial port #1 using any terminal emulation program such as TeraTerm Pro or Hyper Terminal so that sample filter program reads data from tcp buffer and keeps going to run next step.

```
...
(gdb) n
270         if (f2s_fd < 0) pthread_exit(NULL);
(gdb) n
272         while(!exit_flag) {
(gdb) n
274             nread = read(e2f_fd, buf, sizeof(buf));
(gdb) n
275             if (nread<=0) continue;
(gdb)
```

**Step 11.** By setting break point at line #275 and display variable buf[0], user can monitor data coming from tcp port whenever there is a data from it.

```
(gdb) display buf[0]
1: buf[0] = 97 'a'
(gdb) break 275
Breakpoint 2 at 0x10003160: file data_conversion.c, line 275.
(gdb) n
277         if (echo_flag) {
1: buf[0] = 97 'a'
(gdb) c
Continuing.

Breakpoint 2, e2s_thread (arg=0x0) at data_conversion.c:275
275         if (nread<=0) continue;
1: buf[0] = 50 '2'
(gdb) c
Continuing.

Breakpoint 2, e2s_thread (arg=0x0) at data_conversion.c:275
275         if (nread<=0) continue;
1: buf[0] = 51 '3'
(gdb)
```

**Step 12.** User can stop the GDB debugger using 'quit' command.

```
(gdb) quit
The program is running.  Exit anyway? (y or n) y
```

To run sample filter program again, repeat from Step 5. after killing existing ss.port and ss.tcp daemon.

```
# killall ss.port
# killall ss.tcp
```

For more detail information about GDB debugger, please refer to GNU documentation page.  
(<http://www.gnu.org/software/gdb/documentation/>)