SCIOPTA

High Performance
Real-Time Operating Systems

ARM
Druid
System Level Debugger

User's Guide

Headquarters

SCIOPTA Systems AG
Fiechthagstrasse 19
4103 Bottmingen
Switzerland
Tel. +41 61 423 10 62
Fax +41 61 423 10 63
email: sales@sciopta.com
www.sciopta.com

Document No. S07018RL1

Table of Contents

1 Introduction

1.1 SCIOPTA ARM Real-Time Operating System

SCIOPTA ARM is a high performance real-time operating system for microcontrollers using the ARM cores ARM7TDMI, ARM7TDMIS, ARM966E-S, ARM940T, ARM946E-S, ARM720T, ARM920T, ARM922T, ARM926E-JS and other derivatives of the ARM 7/9 family. Including:

Atmel AT91SAM
AT91SAM7S, AT91SAM7SE, AT91SAM7X, AT91SAM9 and all other ARM7/9 based microcontrollers.

NXP LPC2000
LPC21xx, LPC22xx, LPC212x, LPC23xx, LPC24xx, LPC28xx, LPC3180, and all other ARM7/9 based microcontrollers.

Sharp ARM7 and ARM9 MCU and SoC
LH754xx, LH795xx, LH7A4xx and all other ARM7/9 based microcontrollers.

STMicroelectronics STR7 and STR9
STR71x, STR72x, STR75x, STR91x and all other ARM7/9 based microcontrollers.

Including all microcontrollers from other suppliers which have ARM7/9 cores.

The operating system environment includes:

- KRN - Pre-emptive Multi-Tasking Real-Time Kernel
- BSP - Board Support Packages
- IPS - Internet Protocols (TCP/IP)
- IPS Applications - Internet Protocols Applications (Web Server, TFTP, DNS, DHCP, Telnet, SMTP etc.)
- SFATFS - FAT File system
- SFFS - FLASH File system, NOR
- SFFSN - FLASH File system, NAND support
- USBD - Universal Serial Bus, Device
- USBH - Universal Serial Bus, Host
- DRUID - System Level Debugger
- SCIOPTA PEG - Embedded GUI
- CONNECTOR - support for distributed multi-CPU systems
- SMMS - Support for MMU
- SCAPI - SCIOPTA API on Windows or LINUX host
- SCSIM - SCIOPTA Simulator

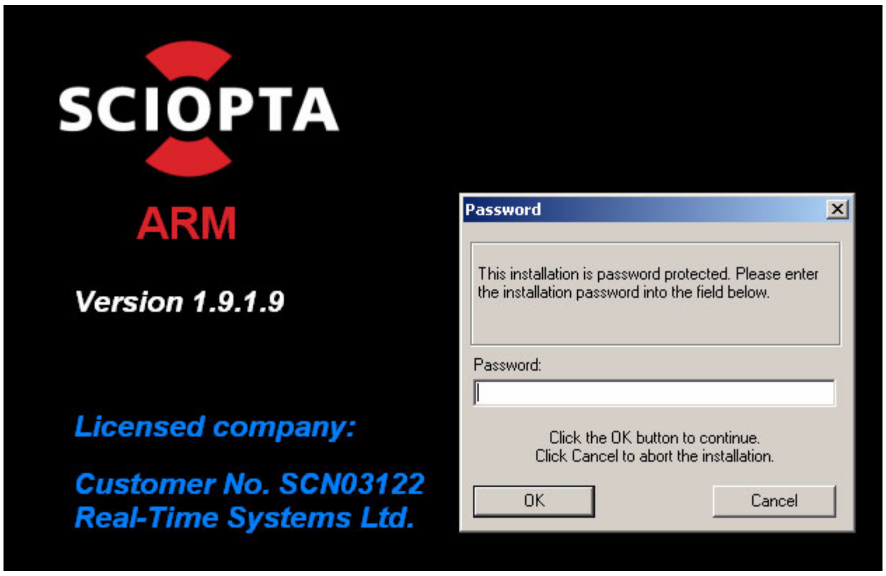
1.2 About This Manual

DRUID is a system level debugger for the **SCIOPTA** real-time operating systems. This manual includes a description of the DRUID architecture and gives an introduction on how to configure and use it. Detailed descriptions of the DRUID functionality including all windows, menus and commands are given.

Please consult also the SCIOPTA ARM - Kernel, User's Guide.

2 Installation

Please consult chapter 2 Installation of the SCIOPTA ARM - Kernel, User's Guide for a detailed description and guidelines of the SCIOPTA installation.



This installation is password protected. Please enter the installation password into the field below.

Click the OK button to continue.
Click Cancel to abort the installation.

Figure 2-1: Main Installation Window

3 Getting Started

3.1 Introduction

These are small tutorial examples which gives you a good introduction into typical SCIOPTA systems and products.

They can be used as a starting point for more complex applications and your real projects.

Please note:

- The Getting-Started examples are using the Eclipse IDE, the MSys Make Utility, the GNU GCC Cross Compiler, the SCIOPTA BSPs (Board Support Packages) and the SCIOPTA examples. If you are using another board, CPU, compiler or IDE you might have to adapt the examples and you might need to change some or all the following files:

  Project SCIOPTA configuration file: hello.xml
  Project file: system.c
  Linker script: <board_name>.ld for GNU GCC
  Board assembler files such as: led.S, resethook.S
  BSP C files such as: druid_uart.c, eth.c, serial.c, simple_uart.c systick.c
  C Startup assembler file: cstartup.S

- Install **GCC version 3.4.4**, GNU Binutils version 2.16.1 and **MSys Build Shell version 1.0.10**. These products can be found on the SCIOPTA CD delivery.
- In the getting started examples we are using the **ECLIPSE** IDE. Install the Eclipse Platform including the **CDT** C/C++ Development Toolkit. These products can be downloaded from the **http://www.eclipse.org/** and the **http://www.eclipse.org/cdt/** web sites.
- We will use the MSys make utility for the Eclipse Platform. Therefore you need to add the MSys **bin** directory in your **PATH environment variable** (e.g. c:\msys\1.0\bin)
- Include the GNU GCC compiler **bin** directory in your **PATH environment variable** as described in chapter "GNU Tool Chain Installation" of the SCIOPTA ARM - Kernel, User's Guide.
- Check that the **environment variable $SCIOPTA_HOME** is defined as described in chapter "SCIOPTA_HOME Environment Variable" of the SCIOPTA ARM - Kernel, User's Guide.
- For every example we are copying all needed files into a local folder. This is not very useful in normal project development, but here it is done so, to show you what files are needed for the examples.
- You might need to setup your source-level emulator/debugger to initialize the memory. Please check the example linker script to fit to your board memory map.

3.2 Getting Started - DRUID System Level Debugger Example

3.2.1 Description

- The Getting Started System for the SCIOPTA DRUID System Level Debugger shows how to set-up DRUID in a typical SCIOPTA system. The SCIOPTA application consists of the simple kernel getting started example (please consult the getting started chapter of the SCIOPTA ARM - Kernel, User's Guide for more information).
- The DRUID host GUI communicates over serial connection to the target system.

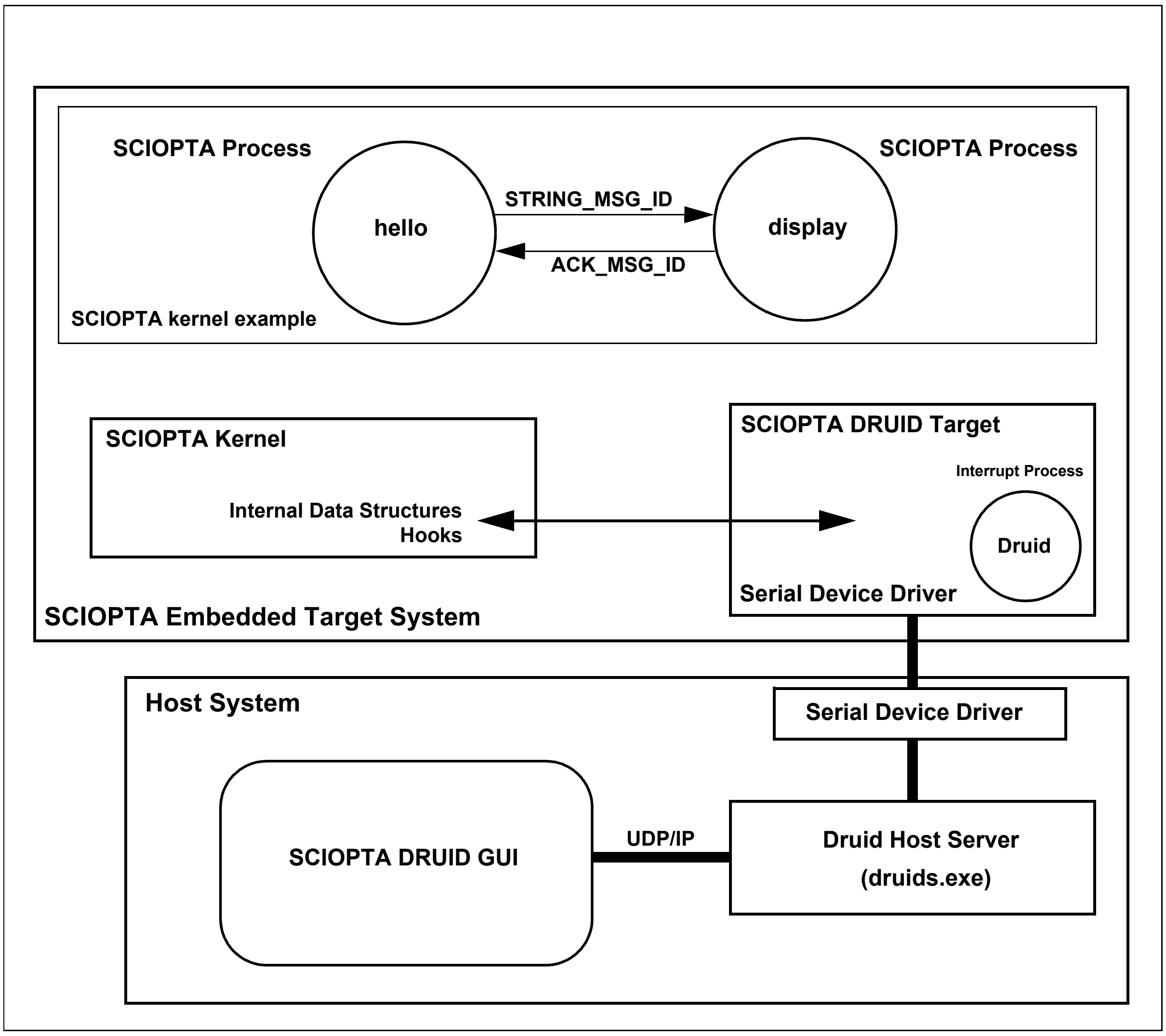


Figure 2-1: DRUID Architecture

3.2.2 DRUID Example - Windows Host

3.2.2.1 Equipment

The following equipment is used to run this getting started example:

- Personal Computer or Workstation with: Intel® Pentium® processor, Microsoft® Windows XP, 256 MB of RAM and 200 MB of available hard disk space.
- Target board which is supported by a SCIOPTA board support package (BSP). For each supported board there is a directory in the example folder: <install_folder>\sciopta\<version>\exp\krn\arm\druidhello\
- Source-level emulator/debugger for ARM connected to the target board.
- A serial cable connected between the UART port of the target board which was selected for DRUID and one of the UART COM ports of your host computer.
- SCIOPTA ARM - Base Package.
- SCIOPTA ARM - Kernel.
- SCIOPTA ARM - DRUID System Level Debugger.
- GCC version 3.4.4, GNU Binutils version 2.16.1 and MSys Build Shell version 1.0.10. These products can be found on the SCIOPTA CD delivery.
- Eclipse Platform Version 3.2.1 including CDT C/C++ Development Toolkit version 3.1.1. These products can be downloaded from the **http://www.eclipse.org/** and the **http://www.eclipse.org/cdt/** web sites.
  In order to run the Eclipse Platform you also need the Sun Java 2 SDK, Standard Edition for Microsoft Windows.

3.2.2.2 Step-By-Step Tutorial

1. Create a project folder to hold all project files (e.g. c:\myproject\sciopta) if you have not already done it for other getting-started projects.
2. Launch Eclipse. The Workspace Launcher window opens.
3. Select your created project folder (e.g. c:\myproject\sciopta) as your workspace (by using the Browse button).
4. Click the OK button. The workbench windows opens. Close the Welcome Tab.
5. Maximize the workbench and deselect **“Build Automatically”** in the **Project** menu.
6. Open the **New Project** window (menu: **File -> New -> Project ...**).
7. Expand the **C** folder and select **Managed Make C Project**. Click the **Next** button.
8. Enter the project name (e.g. **krn_druidhello**) and click on the **Finish** button.
9. **Confirm Perspective Switch** by clicking on the **Yes** button. A new workbench opens and you can see the crated project in the Navigator window. Eclipse has crated a project folder (e.g. c:\myproject\sciopta\krn_druidhello).
10. Select the new created project in the browser window and close the project (menu: **Project -> Close Project**).
11. The next steps we will execute outside Eclipse.
12. Open a **Windows Explorer** or a **Command Prompt** window.

13. Copy the batch file **copy_files.bat** from the example directory of your board: <install_folder>\sciopta\<version>\exp\krn\arm\druidhello\<board> to the working directory of your Eclipse project (e.g. c:\myproject\sciopta\krn_druidhello).
14. Browse to the working directory of your Eclipse project (e.g. c:\myproject\sciopta\krn_druidhello).
15. Double click on the **copy_files.bat** file to execute the batch file and the file copy process.
16. Launch the SCIOPTA configuration utility **sconf** from the desktop.
17. Load the SCIOPTA example project file **hello.xml** from your project folder into sconf.
    Menu: **File->Open**.
18. Click on the **Build All** button or press CTRL-B to build the kernel configuration files. The following files will be created in your project folder: **sciopta.cnf**, **sconf.c** and **sconf.h**.
19. Swap back to the Eclipse workbench. Make sure that the kernel hello project is highlighted (krn_hello) and open the project (menu: **Project -> Open Project**).
20. Expand the project by selecting the [+] button and make sure that the kernel hello project is highlighted (krn_hello).
21. Type the F5 key (or menu **File -> Refresh**). Now you can see all files in the Eclipse Navigator window.
22. Open the Console window at the bottom of the Eclipse workbench to see the project building output.
23. Be sure that the project (**krn_druidhello**) is high-lighted and build the project (menu **Project -> Build Project**). The executable (**sciopta.elf**) will be created in the debug folder of the project (e.g. c:\myproject\sciopta\krn_druidhello\debug).
24. Launch your source-level emulator/debugger.
25. For some specific emulators/debuggers you might found project and board initialization files in the original example directories.
26. Download the resulting **sciopta.elf** on your target board.
27. Make sure that the serial cable is connected between the UART port of the target board and one of the UART COM ports of your host computer board which you have selected for DRUID. The target UART unit to be used for DRUID is defined in the board configuration file **config.h** located in the project directory.
28. Start the target system.
29. Launch the druid server (druids.exe) on the host computer as explained in chapter **8 "Druid Server" on page 8-1**. Make sure that you select the right correct host COM-port and UDP settings.
30. Launch the druid debugger GUI (druid.exe) and connect to the druid server and start a debugging session as explained in chapter **9 "Using Druid" on page 9-1**.

4 DRUID Architecture

4.1 Description

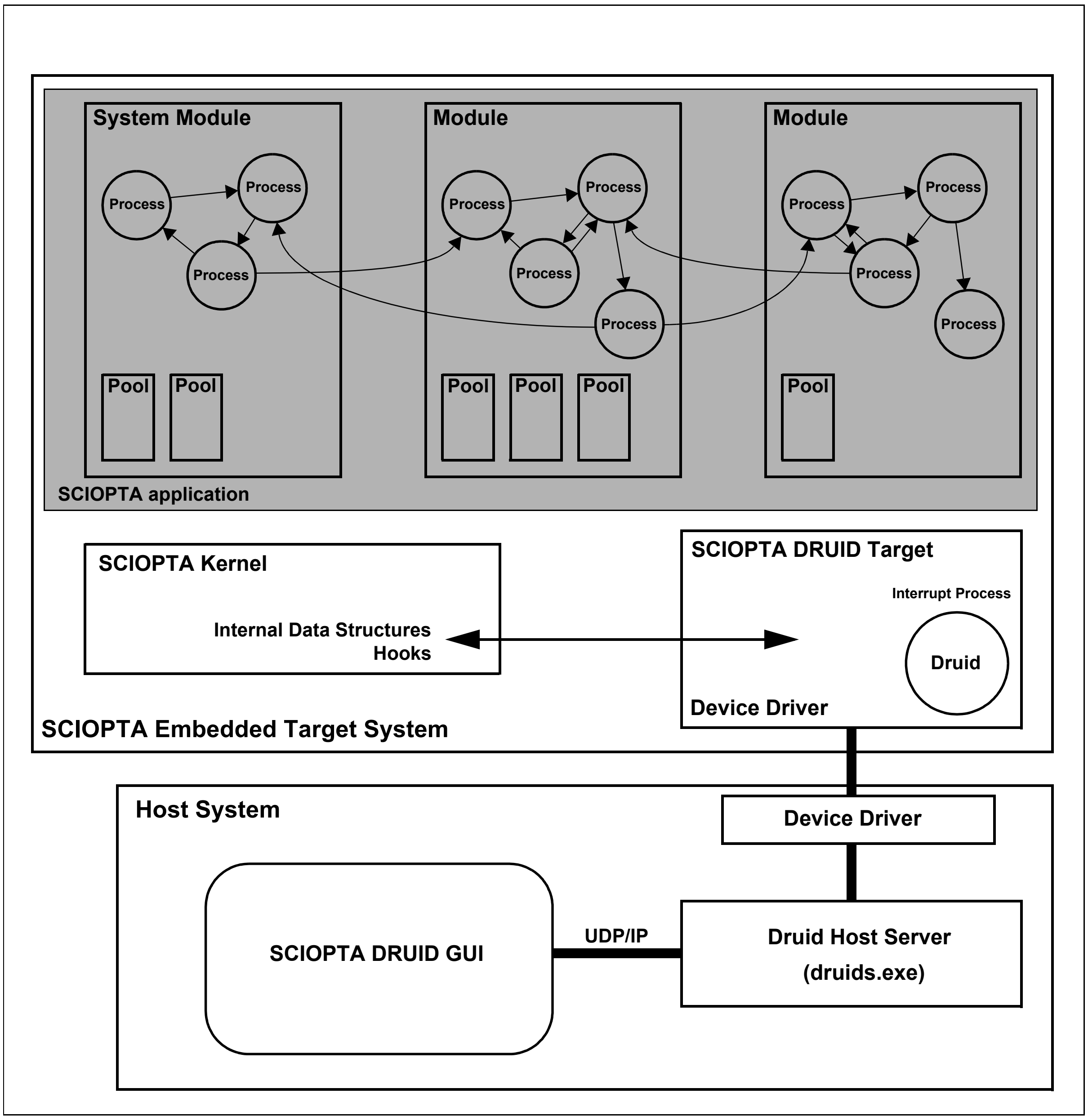


Figure 2-1: DRUID Architecture

In a typical real-time system the crucial parts are the interfaces between the different system components and the real-time behaviour and status of system functions and resources. In a SCIOPTA system, the components are processes and the interface is usually made up of SCIOPTA messages.

With a traditional source code debugger it is quite difficult to supervise or debug these interfaces because a source level debugger concentrates on code lines. The DRUID system level debugger is designed to make it easy to work with the message passing interfaces.

DRUID is a stand-alone debugger which is controlled over a separate communication interface such as a serial line for instance. It can be used in conjunction with any other debugging tool such as in-circuit debuggers or emulators from any supplier.

The DRUID system consists of

- a graphical user interface (GUI) and some communication programs running on the PC workstation host
- a specific device driver and some additional modules on the target system and
- a communication system (cable) connecting the host to the SCIOPTA target.

There is a specific binary communication protocol used between DRUID host and DRUID target which is not disclosed.

4.2 DRUID Target

The DRUID target system is a program which works alongside the SCIOPTA real-time operating system and has as little as possible undesirable influence on the target application. It can access all SCIOPTA internal data such as:

- SCIOPTA specific global data
- Module control blocks
- Process control blocks
- Message pools
- Process stacks

DRUID can read these data while the system is running or from a halted system. The user can run or stop the target system from within the DRUID debugger.

A strong feature of the DRUID system level debugger is to trace SCIOPTA messages and to do some actions on specific message passing. To implement this function, the DRUID target program maintains a trace memory. The size of the trace memory is fully configurable and can be adapted to the target system memory map. For tracing message the debugger uses some specific kernel features such as message and process hooks. Please consult the SCIOPTA kernel and target manuals for more information about hooks.

The DRUID target includes a device driver for serial communication. Although any communication interface may be used for DRUID we have included a serial driver by default. The reason is that serial interfaces exist in almost all target systems and they are less and less used for the application.

To avoid interferences with the SCIOPTA application the DRUID target systems is using no SCIOPTA messages and no SCIOPTA system calls. There might be only some SCIOPTA DRUID interrupt and/or device driver processes visible in the system. This allows to debug all kind of SCIOIPTA applications including device driver systems (i.e also for communication stacks).

The DRUID target systems needs to be configured to work properly. Please consult chapter **5 “Configuration” on page 5-1** for more information.

4.3 DRUID Host

The DRUID target is controlled by the DRUID host using a host-target communication link. The DRUID host consists of the DRUID graphical user interface (GUI) and the DRUID server. The GUI host communicates to the DRUID server by using the host UDP/IP stack and a specific DRUID high-level protocol.

The DRUID GUI is the main program for the DRUID user and is written in a specific development environment allowing to run it on a Windows or on a LINUX PC workstation.

The DRUID server translates the high-level protocol into the DRUID binary protocol which is used for the host-target communication. It also controls the host device driver (actually mainly the UART interface).

The DRUID server (druids.exe in Windows) needs to be started separately and can be configured for the used devices. For the UART interface for instance the COM port and the COM parameters can be configured.

5 Configuration

5.1 Introduction

DRUID Configuration is divided in two parts:

1. Configuring and defining all static objects (modules, pools and processes) with the help of the SCIOPTA configuration utility SCONF (sconf.exe). Please consult the SCIOPTA ARM - Kernel User's Guide for more information about using sconf.exe and the static configuration process.

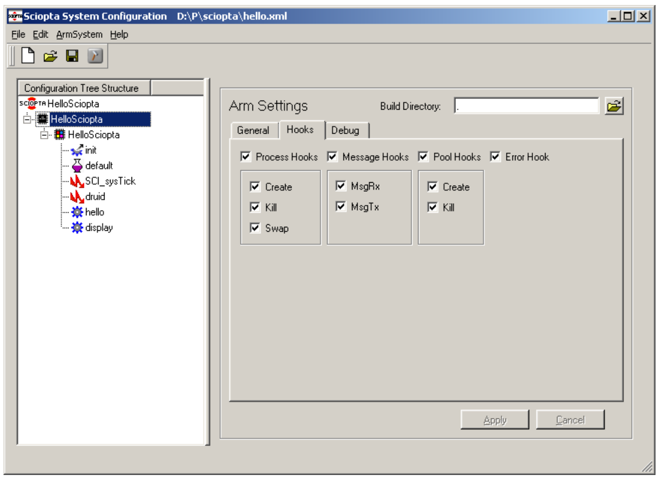
   The static objects will be generated and started automatically at system start.
2. SCIOPTA DRUID configuration by setting some values in specific configuration files.

5.2 System Configuration

After you have selected the system in the SCIOPTA configuration utility SCONF (sconf.exe), the corresponding parameter window on the right side will show the parameters for the selected target CPU system. The system configuration for ARM target systems is divided into 3 tabs: General, Hooks and Debug.

In addition to your usual system configuration for your target system you need to configure the SCIOPTA hooks settings when you are using DRUID.

Please consult the SCIOPTA ARM Kernel, User's Guide for more information about the SCIOPTA configuration tool.



Sciopta System Configuration D:\P\sciopta\hello.xml

Configuration Tree Structure
- HelloSciopta
  - HelloSciopta
    - HelloSciopta
      - init
      - default
      - SCI_sysTick
      - druid
      - hello
      - display

Arm Settings — Build Directory: .

Tabs: General | Hooks | Debug

- [✓] Process Hooks: [✓] Create, [✓] Kill, [✓] Swap
- [✓] Message Hooks: [✓] MsgRx, [✓] MsgTx
- [✓] Pool Hooks: [✓] Create, [✓] Kill
- [✓] Error Hook

Figure 5-1: DRUID System hooks configuration

5.2.1 Hooks

Hooks are user written functions which are called by the kernel at different locations. They are only called if the user defined them at configuration. User hooks are used by the DRUID debug system for the trace function. You can find the hook configuration setting on the **system level** of the SCONF tool (hook tab).

5.2.1.1 Message Hooks

For tracing the target message transmission and reception the SCIOPTA kernel message hooks are used.

Select the "Message Hooks" checkbox (including MsgRx and MsgTx) when running the SCIOPTA configuration utility SCONF.

The function code of the message hooks are included in the main DRUID target program (**druidprotocol.c**). The hook registration is done dynamically (by using the **sc_msgHookRegister**) and is also located in the main DRUID program.

5.2.1.2 Process Hooks

Please Note:

In the actual DRUID version process swap, create and kill trace is not yet supported. You can select the "Process Hooks" checkbox but it has no effect on the DRUID functionality. Process swap, create and kill trace will be added in one of the next versions.

5.2.1.3 Pool Hooks

Please Note:

In the actual DRUID version pool create and kill trace is not yet supported. You can select the "Pool Hooks" checkbox but it has no effect on the DRUID functionality. Pool create and kill trace will be added in one of the next versions.

5.2.1.4 Error Hooks

In a typical SCIOPTA system the user must write the error hook as only the user can decide what to do if a specific error occurred.

To enable error hook functionality select the "Error Hook" checkbox (including MsgRx and MsgTx) when running the SCIOPTA configuration utility SCONF.

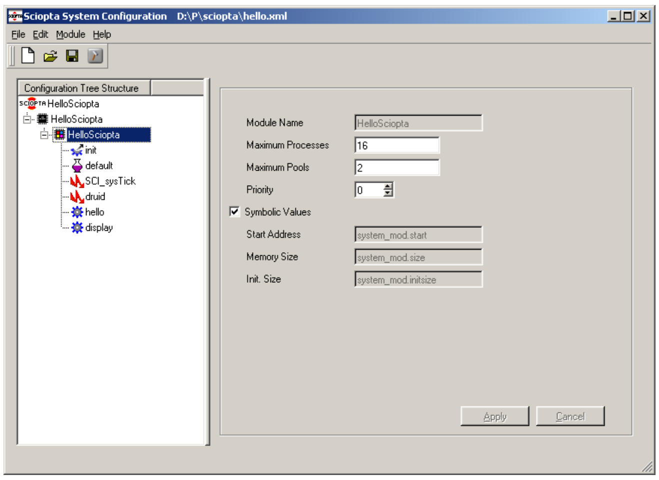
5.3 DRUID Module

For SCIOPTA systems using DRUID you will normally put some basic processes (Device Drivers) in the System Module (each SCIOPTA application has a System Module sometimes also called module 0).

All other remaining processes might reside in other specific application Module. But you could place all processes in the System Module.

In our delivered getting started example we are just using the system module which contains the basic device drivers and a small application.

Please consult the SCIOPTA Kernel, User’s Guide for more information about the SCIOPTA configuration tool.


Sciopta System Configuration D:\P\sciopta\hello.xml
File Edit Module Help
Configuration Tree Structure
Apply
Cancel

- HelloSciopta
  - HelloSciopta
    - HelloSciopta
      - init
      - default
      - SCI_sysTick
      - druid
      - hello
      - display

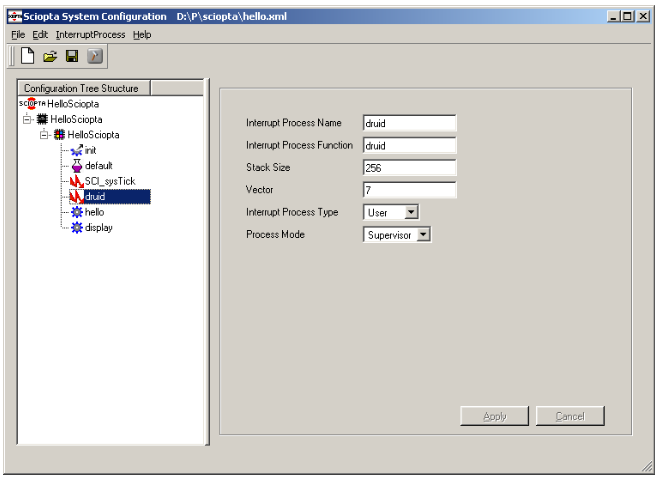
Module Name	HelloSciopta
Maximum Processes	16
Maximum Pools	2
Priority	0
✓ Symbolic Values	
Start Address	system_mod.start
Memory Size	system_mod.size
Init. Size	system_mod.initsize

Figure 5-2: DRUID System with only one module

5.3.1 DRUID Uart Interrupt Processes

DRUID target includes a device driver. If an UART is used for communication between host and target you need to define a SCIOPTA interrupt process (druid). This interrupt process is included in the file **druid_uart.c.**

File location: <install_folder>\sciopta\<version>\bsp\arm\<cpu>\src\


Sciopta System Configuration D:\P\sciopta\hello.xml
File Edit InterruptProcess Help
Configuration Tree Structure
HelloSciopta
HelloSciopta
HelloSciopta
init
default
SCI_sysTick
druid
hello
display
Apply
Cancel

Interrupt Process Name	druid
Interrupt Process Function	druid
Stack Size	256
Vector	7
Interrupt Process Type	User
Process Mode	Supervisor

Figure 5-3: DRUID UART Interrupt Process

5.4 DRUID Configuration

In the previous chapters we have described how to configure the static modules, pools and processes for SCIOPTA DRUID. This consisted mainly in declaring the static modules, pools and process and configuring the names, sizes and priorities in the SCIOPTA configuration utility SCONF.

To run DRUID you need also to configure some parameters depending on the DRUID specified properties.

In the SCIOPTA DRUID example we have include these settings in the file **system.c** which can be found in the examples delivery. This is a board specific file which contains some important system functions such as a start hooks, error hooks and device driver setup code.

File location: <install_folder>\sciopta\<version>\exp\krn\arm\druidhello\<board>\

Other board independent DRUID settings are defined in the file **druidcnf.h** which can also be found in the example delivery.

File location: <install_folder>\sciopta\<version>\exp\krn\arm\druidhello\

5.4.1 Board Setup File system.c

5.4.1.1 Error Hook

When DRUID is used, the kernel errors are captured by the main DRUID target program. In the user written error hook there must be a call to the DRUID error handling (**druid_errorHook**). This function is included in the main DRUID target program (**druidprotocol.c**). The call to **druid_errorHook** must be the first function call in the user error hook.

Example

```
int myErrorHook ( ... ) {
  /* Call to DRUID error handling first */
  druid_errorHook( ... );

  /* User error hook code may be included now */

}
```

5.4.1.2 Start Hook

The start hook is mainly used to do chip, board and system initialization.

The start hook must always be present and must have the name start_hook. The start hook must be written by the user. If a start hook is declared the kernel will jump into it after the C environment is initialized. As the C environment is initialized it can be written in C. After the start hook has executed the kernel will call the dispatcher and the system will start.

In the Start Hook we can register the Error Hook by using the sc_miscErrorHookRegister system call.

Example

```
.
sc_miscErrorHookRegister (myErrorHook);
.
```

In the Start Hook we can also define the way a system starts with included DRUID system level debug functionality. This is done by calling the **druid_startup()** function.

```
void druid_startup(mode);
```

Parameter

mode

DRUID start method.

This parameter can be one of the following values:

Value	Meaning
0	The target system starts without waiting any commands from the DRUID debugger on the host.
1	The target system waits on a start command from the DRUID debugger on the host (see chapter 9.15 “Start and Stop” on page 9-31).

5.4.1.3 Example of a Board Setup File

```
#include <sciopta.h>
#include <druid/device.h>
#include <simple_uart.h>

int myErrorHook(sc_errcode_t err,sc_extra_t extra,int user,const sc_pcb_t *pcb)
{
  if (user) {
    druid_errorHook (err, extra, user, pcb);
    return 1;
  }
  else {
    return druid_errorHook (err, extra, user, pcb);
  }
}

void start_hook(void)
{
  uart_init(0,115200);
  sc_miscErrorHookRegister(myErrorHook);
  druid_startup (1);
}

void HelloSciopta(void)
{
  /* Place further code executed at init priority */
}

void __putchar(int c)
{
  if ( c == '\n' ){
    uart_putchar(0,'\r');
  }
  uart_putchar(0,c);
}
```

5.4.2 DRUID Configuration File druidcnf.h

Actually DRUID is supporting SCIOPTA message trace. You can setup **trace-start** and **trace-stop** conditions and DRUID will trace the defined messages between start and stop in reserved target RAM.

You need to define three values for the message trace in the druid header file (**druidcnf.h**). This file will be included by the main DRUID target program (**druidprotocol.c**).

1. CNF_DRUID_OPTION_MAX, maximum entries in the Trace Option List (see chapter **9.13.3 “Trace Option List” on page 9-26**).
2. CNF_DRUID_TRACE_MEM_MAX, maximum number of stored trace frames.
3. CNF_DRUID_MSG_DATA_MAX, Maximum number of stored data bytes per message.

5.4.2.1 CNF_DRUID_OPTION_MAX

A DRUID Trace Option is a definition for the DRUID debugger what message is to be traced. You can define what specific messages sent from specific modules/processes to specific modules/processes should be traced. DRUID maintains a Trace Option List of such definitions in the target.

The CNF_DRUID_OPTION_MAX definition allows you to define a maximum number of Trace Options.

The default value is 5.

The number of bytes used for one Trace Option is target processor dependent. Please consult the SCIOPTA target manual of your processor.

5.4.2.2 CNF_DRUID_TRACE_MEM_MAX

A message passing traced by DRUID is called a frame. A DRUID frame contains the sender process ID, the receiver process ID, the message ID and a specific number of message data bytes.

The CNF_DRUID_TRACE_MEM_MAX definition allows you the define a maximum number of DRUID frames stored in the trace memory. The trace memory is organized as a circular buffer.

The default value is 4096.

The number of bytes used for one DRUID frame is target processor dependent. Please consult the SCIOPTA target manual of your processor.

5.4.2.3 CNF_DRUID_MSG_DATA_MAX

For every trace DRUID frame a specific number of message data bytes are stored in the trace memory.

This number is user selectable and is defined by the CNF_DRUID_MSG_DATA_MAX value. Please select this number with care as a high value will consume a lot of target memory and will affect the real-time behaviour. Every message byte to trace will be copied in the trace memory.

The default value is 20.

6 System Building

6.1 Introduction

In this chapter we will give you some information about the building process for a Freescale ARM target system including the DRUID system level debugger.

Please consult chapter “System Building” of the SCIOPTA ARM - Kernel, User’s Guide for general information about system building. You will find there information about:

- System Files
- Data Types
- System Building Diagram
- Assembling, Compiling and Linking
- Kernel and Utilities Libraries
- Include Files
- Linker Scripts
- Memory Map

6.2 System Design

First you have to determine the specification of the system. As you are designing a real-time system, speed requirements needs to be considered carefully including worst case scenarios. Defining function blocks, environment and interface modules will be another important part for system specification.

Systems design includes defining the modules, processes and messages. SCIOPTA is a message based real-time operating system therefore specific care needs to be taken to follow the design rules for such systems. Data should always be maintained in SCIOPTA messages and shared resources should be encapsulated within SCIOPTA processes.

6.3 System Files

For initializing and setting up your target you need some specific system files such as board setup files, files to initialize the C/C++ environment, device driver and interrupt handler files. You will find SCIOPTA system files for many popular boards in the Board Support Package (BSP) delivery. If you are using a different board or you are designing your own hardware these file represent a good model and starting point for writing your own system files.

Please consult chapter **8 “Board Support Packages” on page 8-1** and the SCIOPTA - Device Driver, User’s Guide.

6.4 Include Files

No specific include files search directories for DRUID needs to be declared. Please consult chapter “Include Files” of the SCIOPTA ARM - Kernel, User’s Guide for all information about include files.

6.5 SCIOPTA ARM DRUID Libraries

The device driver independent functions of the DRUID target part are included in the file **druidprotocol.c**.

This file is delivered in source code and therefore no libraries need to be included in the build process.

File location: <install_folder>\sciopta\<version>\druid\target\

6.6 Linking the SCIOPTA ARM System

Please consult chapter “Linking the SCIOPTA ARM System” of the SCIOPTA ARM - Kernel, User’s Guide for general information about linking.

To include DRUID in your target application you need to add the following files in your link process:

- Device driver independent functions (**druidprotocol.o**)
- DRUID device driver (i.e. **druid_uart.o** for a serial device)

7 Board Support Packages

Only the device driver for SCIOPTA ARM - DRUID are listed here. Please consult chapter “Board Support Packages” of the SCIOPTA ARM - Kernel, User’s Guide for all other information about SCIOPTA BSPs.

7.1 Atmel AT91SAM7 Boards and Drivers

7.1.1 Atmel AT91SAM7 DRUID Serial Device Driver

This is a DRUID serial driver for the Atmel AT91SAM7 controllers and is not board dependent and can be used for all boards using the Atmel AT91SAM7 chip.

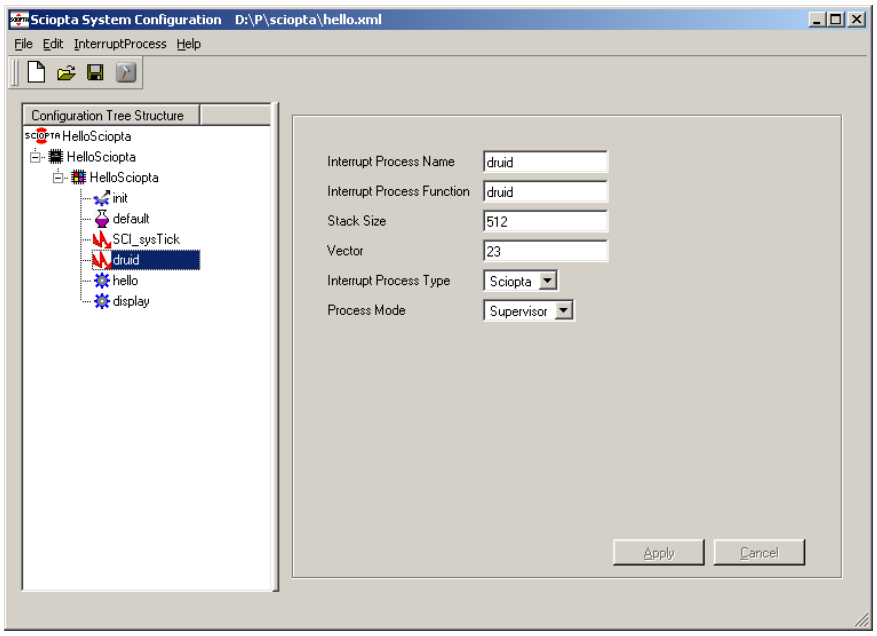
7.1.1.1 Atmel AT91SAM7 DRUID Serial Device Driver Source Files

druid_uart.c	Serial driver for DRUID.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\src\

7.1.1.2 Atmel AT91SAM7 DRUID Interrupt Process

You need to declare the process **druid** in the SCIOPTA SCONF Utility:


Sciopta System Configuration D:\P\sciopta\hello.xml
File Edit InterruptProcess Help
Configuration Tree Structure
HelloSciopta
HelloSciopta
HelloSciopta
init
default
SCI_sysTick
druid
hello
display
Apply
Cancel

Interrupt Process Name: druid
Interrupt Process Function: druid
Stack Size: 512
Vector: 23
Interrupt Process Type: Sciopta
Process Mode: Supervisor

The process is included in the file druid_uart.c.

7.2 Atmel AT91SAM9 Boards and Drivers

7.2.1 Atmel AT91SAM9 DRUID Serial Device Driver

This is a DRUID serial driver for the Atmel AT91SAM9 controllers and is not board dependent and can be used for all boards using the Atmel AT91SAM9 chip.

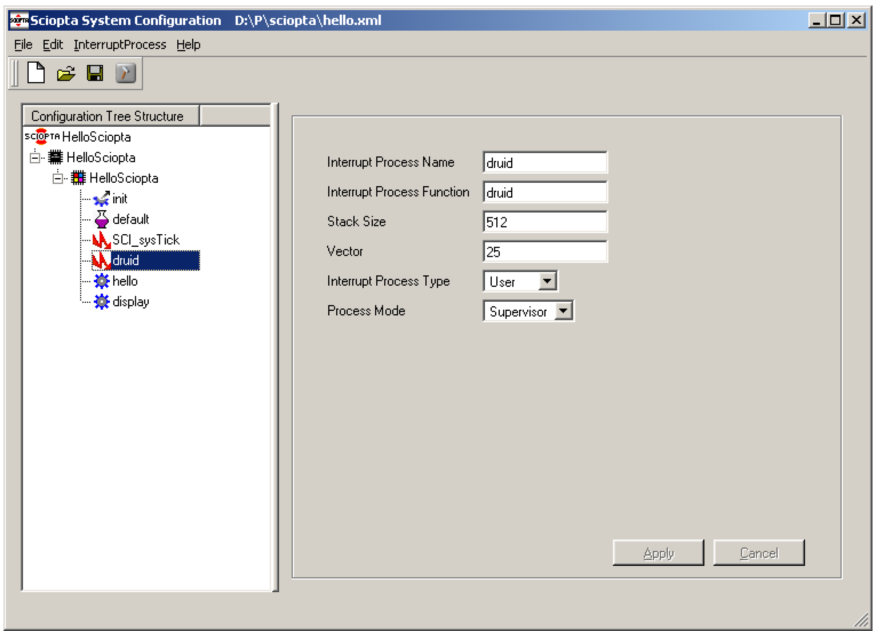
7.2.1.1 Atmel AT91SAM9 DRUID Serial Device Driver Source Files

druid_uart.c	Serial driver for DRUID.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\src\

7.2.1.2 Atmel AT91SAM9 DRUID Interrupt Process

You need to declare the process **druid** in the SCIOPTA SCONF Utility:


Sciopta System Configuration D:\P\sciopta\hello.xml
File Edit InterruptProcess Help
Configuration Tree Structure
HelloSciopta
HelloSciopta
HelloSciopta
init
default
SCI_sysTick
druid
hello
display
Apply
Cancel

Interrupt Process Name: druid
Interrupt Process Function: druid
Stack Size: 512
Vector: 25
Interrupt Process Type: User
Process Mode: Supervisor

The process is included in the file druid_uart.c.

7.3 NXP LPC2000 Boards and Drivers

7.3.1 NXP LPC2000 DRUID Serial Device Driver

This is a DRUID serial driver for the NXP LPC2000 controllers and is not board dependent and can be used for all boards using the NXP LPC2000 chip.

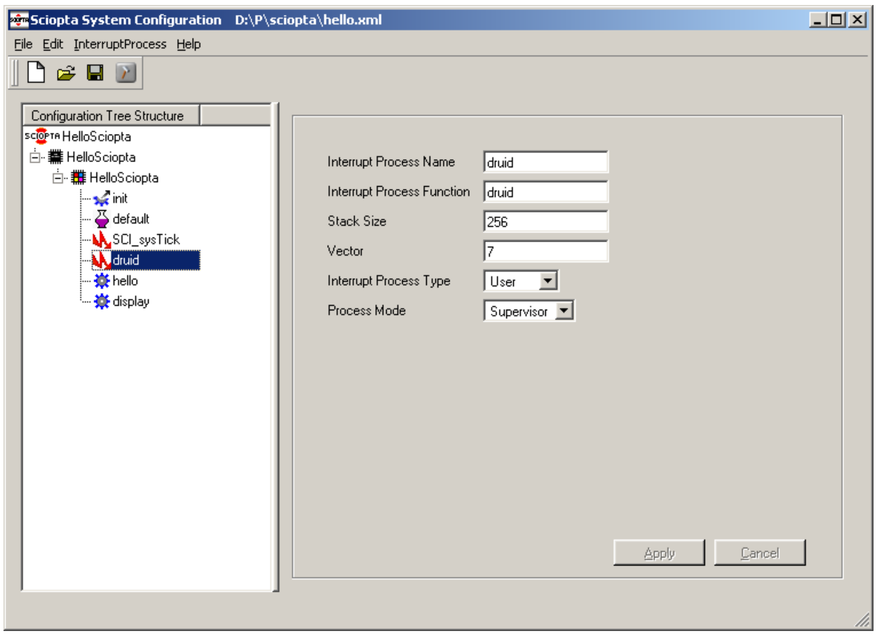
7.3.1.1 NXP LPC2000 DRUID Serial Device Driver Source Files

druid_uart.c	Serial driver for DRUID.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc21xx\src\

7.3.1.2 NXP LPC2000 DRUID Interrupt Process

You need to declare the process **druid** in the SCIOPTA SCONF Utility:


Sciopta System Configuration D:\P\sciopta\hello.xml
File Edit InterruptProcess Help
Configuration Tree Structure
HelloSciopta
HelloSciopta
HelloSciopta
init
default
SCI_sysTick
druid
hello
display
Apply
Cancel

Interrupt Process Name	druid
Interrupt Process Function	druid
Stack Size	256
Vector	7
Interrupt Process Type	User
Process Mode	Supervisor

The process is included in the file druid_uart.c.

7.4 STMicroelectronics STR7 Boards and Drivers

7.4.1 STMicroelectronics STR7 DRUID Serial Device Driver

This is a DRUID serial driver for the STMicroelectronics STR7 controllers and is not board dependent and can be used for all boards using the STMicroelectronics STR7 chip.

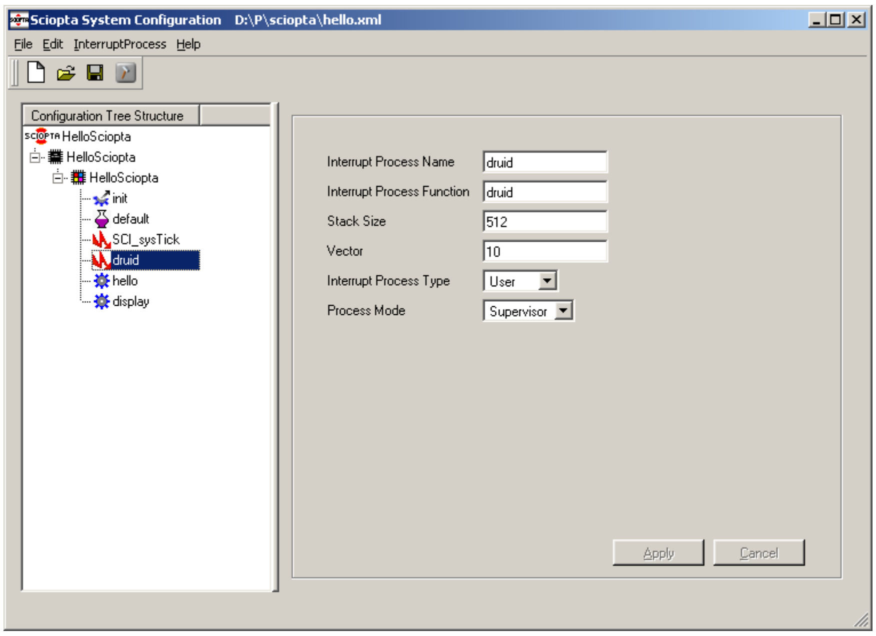
7.4.1.1 STMicroelectronics STR7 DRUID Serial Device Driver Source Files

druid_uart.c	Serial driver for DRUID.

File location: <install_folder>\sciopta\<version>\bsp\arm\str7\src\

7.4.1.2 STMicroelectronics STR7 DRUID Interrupt Process

You need to declare the process **druid** in the SCIOPTA SCONF Utility:


Sciopta System Configuration D:\P\sciopta\hello.xml
File Edit InterruptProcess Help
Configuration Tree Structure
HelloSciopta
HelloSciopta
HelloSciopta
init
default
SCI_sysTick
druid
hello
display
Apply
Cancel

Interrupt Process Name	druid
Interrupt Process Function	druid
Stack Size	512
Vector	10
Interrupt Process Type	User
Process Mode	Supervisor

The process is included in the file druid_uart.c.

7.5 STMicroelectronics STR9 Boards and Drivers

7.5.1 STMicroelectronics STR9 DRUID Serial Device Driver

This is a DRUID serial driver for the STMicroelectronics STR9 controllers and is not board dependent and can be used for all boards using the STMicroelectronics STR9 chip.

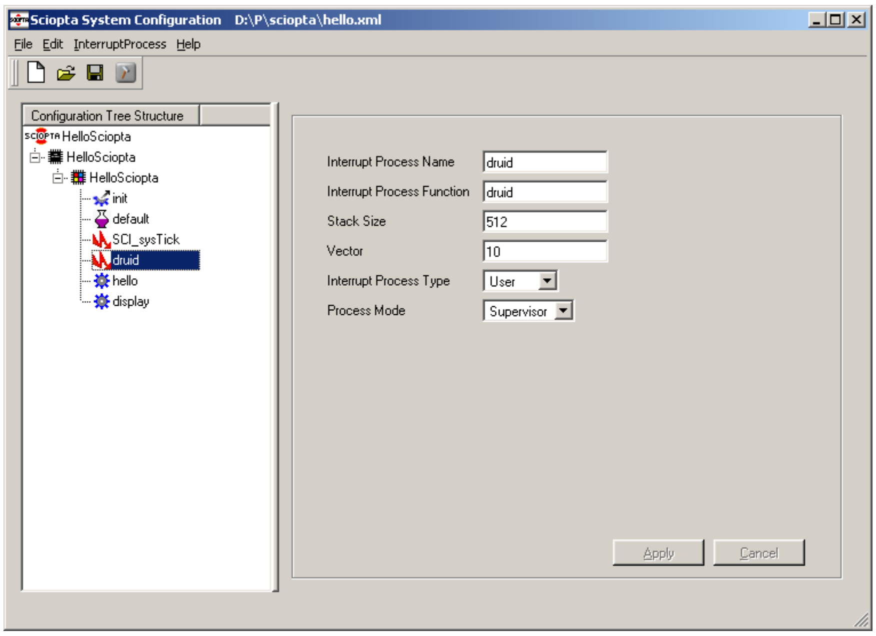
7.5.1.1 STMicroelectronics STR9 DRUID Serial Device Driver Source Files

druid_uart.c	Serial driver for DRUID.

File location: <install_folder>\sciopta\<version>\bsp\arm\str9\src\

7.5.1.2 STMicroelectronics STR9 DRUID Interrupt Process

You need to declare the process **druid** in the SCIOPTA SCONF Utility:


Sciopta System Configuration D:\P\sciopta\hello.xml
File Edit InterruptProcess Help
Configuration Tree Structure
HelloSciopta
HelloSciopta
HelloSciopta
init
default
SCI_sysTick
druid
hello
display
Apply
Cancel

Interrupt Process Name: druid
Interrupt Process Function: druid
Stack Size: 512
Vector: 10
Interrupt Process Type: User
Process Mode: Supervisor

The process is included in the file druid_uart.c.

8 Druid Server

After you have generated the application including the DRUID target system as described in the foregoing chapters, downloaded the application with your source level debugger into the target system and connected the DRUID communication cable (i.e. serial cable) between host and target you are now ready to run the host part of the DRUID system.

The DRUID host consist of the DRUID server (**druids.exe** for Microsoft® Windows) and the DRUID user interface (**druid.exe** for Microsoft® Windows).

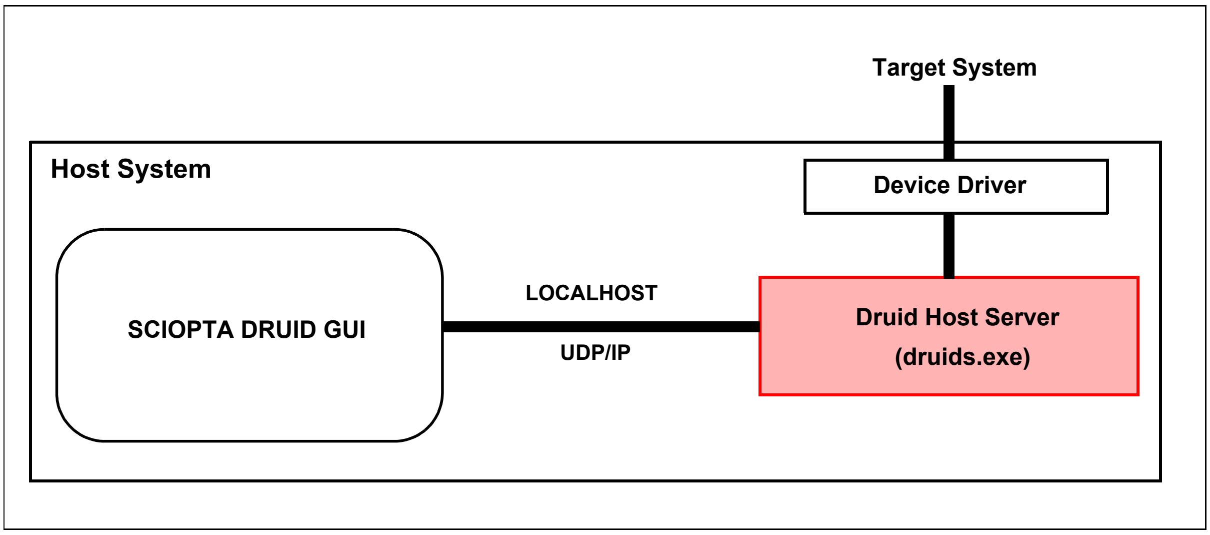


Figure 2-1: DRUID Server

The DRUID server translates the DRUID GUI protocol into a binary protocol for target communication. Actually the DRUID server supports serial (UART) interfaces.

DRUID server and DRUID GUI are connected over the host TCP/IP stack using the UDP protocol. The LOCALHOST IP address can be used.

8.1 Running and Configuring DRUID Server for Windows

8.1.1 Cygwin DLL

The DRUID server support multiple hosts (Microsoft® Windows and Linux). Therefore for Microsoft® Windows you need to have access to the cygwin DLL. The program **cygwin1.dll** is included in the SCIOPTA delivery and can be found in the same directory as the DRUID program. If you run DRUID out of the installed directory the cygwin DLL is automatically found.

8.1.2 Starting DRUID Server

DRUID server must be started before you can run DRUID. You can run DRUID server (druids.exe) in three different ways:

Open a command shell (can usually be found at: %SystemRoot%\system32\cmd.exe). Make the directory where DRUID is installed the current directory.
The file druids.exe is located at: <install_folder>\sciopta\<version>\bin\win32\

Type: druids <options>

Select the DRUID server options for your environment.

8.1.3 DRUID Server Options

Usage

```
druids  [-h] [-d <device>] [-v <baudrate>] [-n <target_net_address] [-p <port>]
        [-m <message name data base(s)>] [-e <user error name data base(s)>]
        [-b <ID name data base(s)>] [-t]
```

Parameter

-h

Help. Returns the help text for the different options

-d

Serial communication port COM1 ... COM9

Default value: COM1

-v

Serial communication port baudrate.

This option can have one of the following values: 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400.

Default value: 115200

-n

Reserved for later use (DRUID over TCP/IP). SCIOPTA IPS must be included and run in the target. The port number between target and server is (port+1). The target cannot be halted in this mode.

-p

UDP Port for DRUID GUI and DRUID Server communication

Default value: 20000

-m

Name and location of the message name database(s). This database allows to display the message names in the DRUID GUI. For each message define a line starting with the message ID followed by a space and the message name.

Example: druids -m hello.druid

File hello.druid:

```
0x1000 STRING_MSG_ID
0x1002 ACK_MSG_ID
0x1003 ADC_MSG_ID
0x1004 RES_MSG_ID
0x1005 NOW_MSG_ID
0x1006 CLK_MSG_ID
0x1007 SET_MSG_ID
0x1010 STRDEV_MSG_ID
0x1012 ACKDEV_MSG_ID
0x1020 STRIPS_MSG_ID
0x1022 ACKIPS_MSG_ID
0x1030 STRUSER_MSG_ID
0x1022 ACKUSER_MSG_ID
```

-e

Name and location of the user error name database(s). This feature is not yet implemented and will be included in one of the next versions.

-b

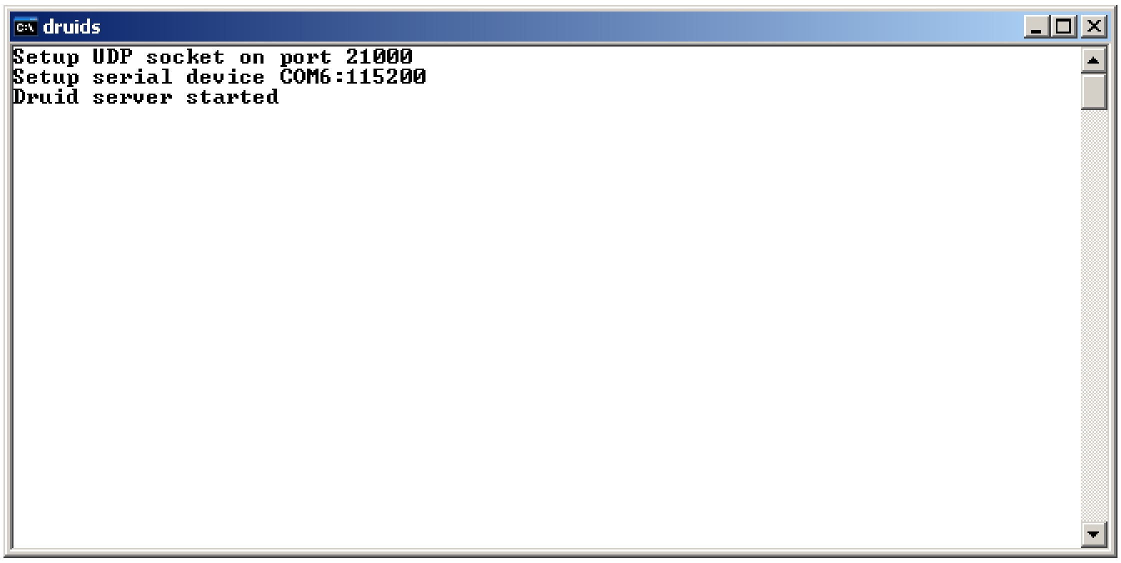
Name and location of the generic ID name database(s). This feature is not yet implemented and will be included in one of the next versions.

-t

Enables test mode to include specific test messages

8.1.4 DRUID Server Window

After druids.exe has been successfully launched the following window will open:



```
Setup UDP socket on port 21000
Setup serial device COM6:115200
Druid server started
```

9 Using Druid

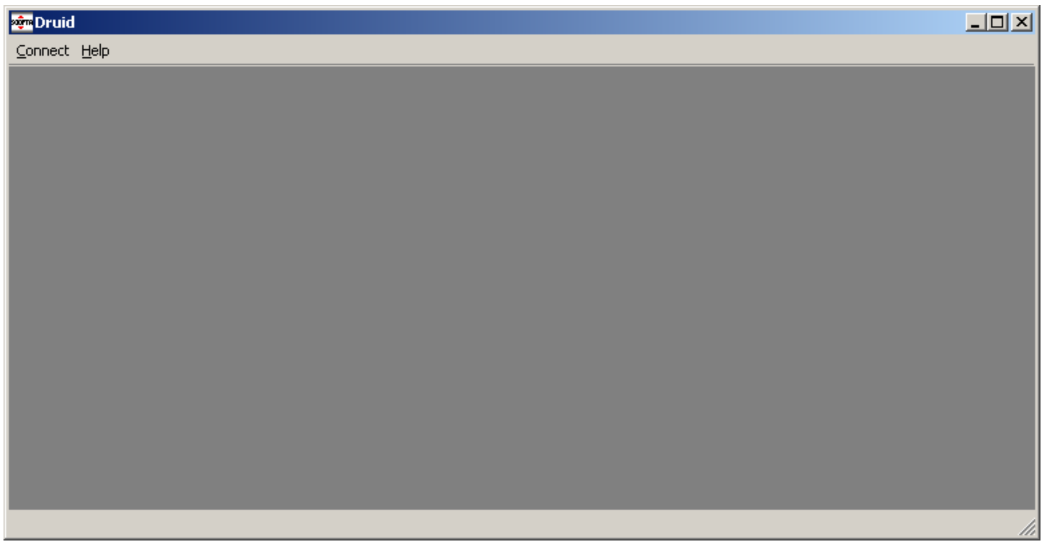
After you have generate the application including the DRUID target system as described in the foregoing chapters, downloaded the application with your source level debugger into the target system, connected the DRUID communication cable (i.e. serial cable) between host and target and started the DRUID server you are now ready to run DRUID (the graphical user interface GUI).

9.1 Starting DRUID

DRUID server must be started before you can run DRUID. You can run DRUID server (druids.exe) in two different ways:

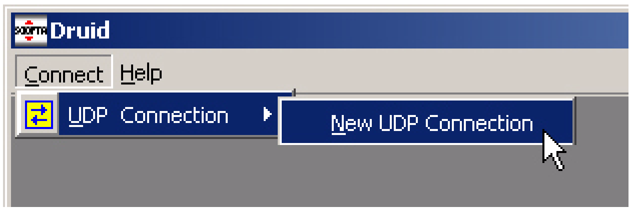
1. Run DRUID from the Microsoft® Windows standard "Start-Programs-Sciopta-**Druid**" menu.
2. Run DRUID from the desktop shortcut.

The following DRUID start window appears:

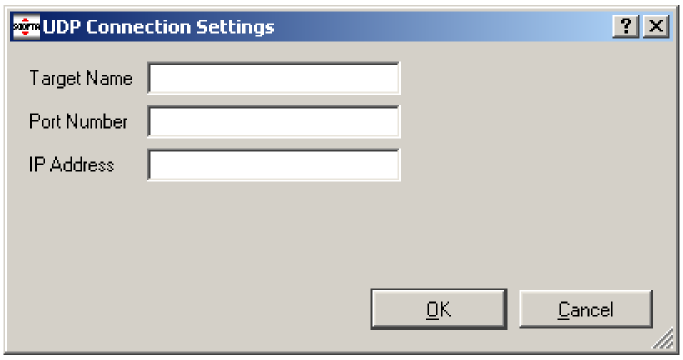


9.2 New Connection

First you need to connect DRUID to the (running) DRUID server. Select the **Connect** menu, the **UDP Connection** and the **New UDP Connection**.



A new window will appear where a new connection can be defined:



9.2.1 Target Name

This is the name for your connection or for your target which can bee freely chosen. The name will appear in the future in the connect menu. So you have to configure a specific connection only once and you can select it later by the name.

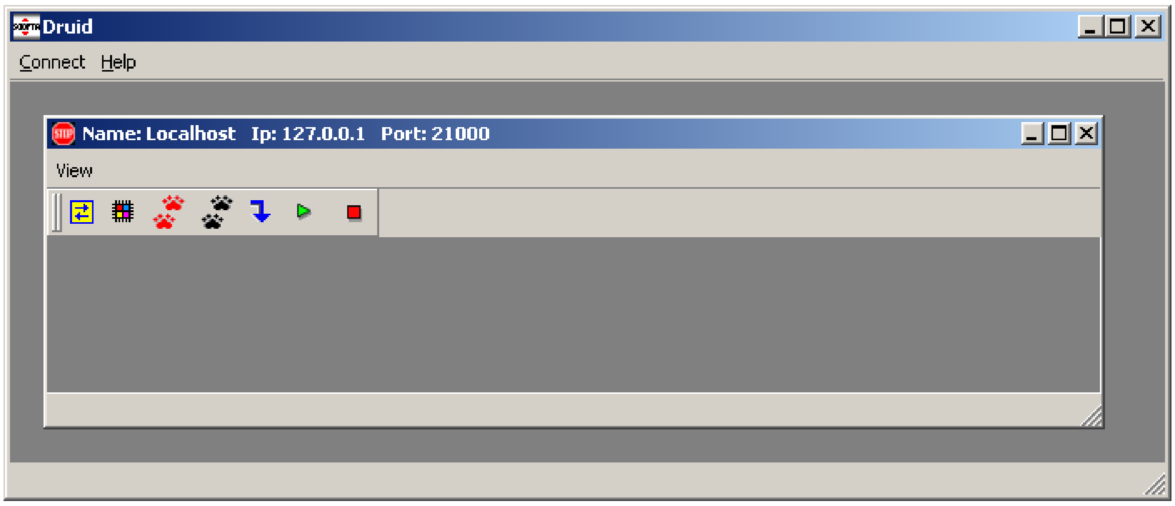
9.2.2 Port Number

Enter here the same UDP port number which was given for the DRUID server. Remember that the default port of the DRUID server is 20000.

9.2.3 IP Address

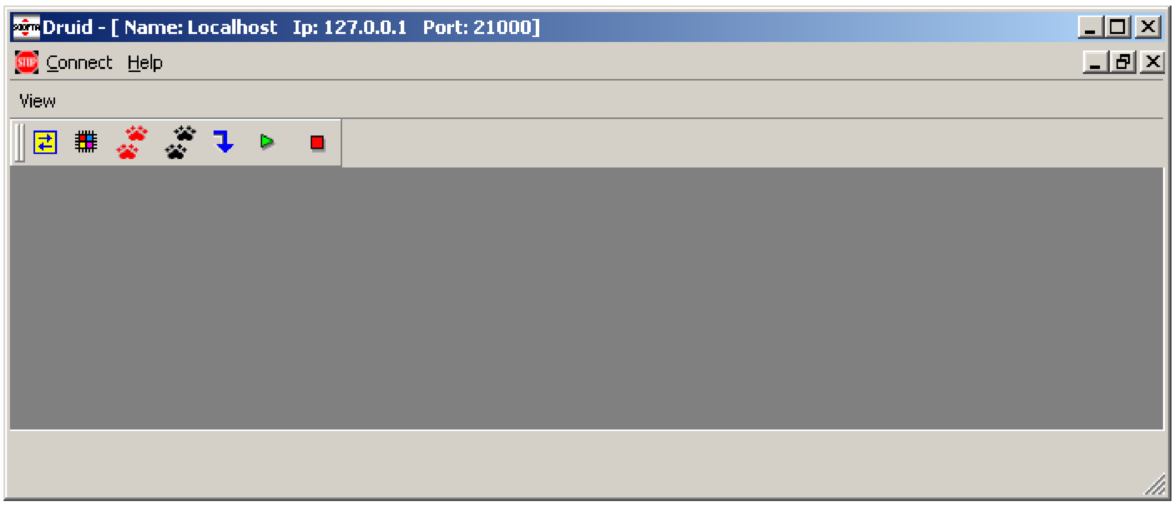
Actually the IP address of the DRUID server is fixed to localhost. Therefore just enter the localhost IP address here (127.0.0.1).

Type OK and a new DRUID window with the given connection parameters will open:



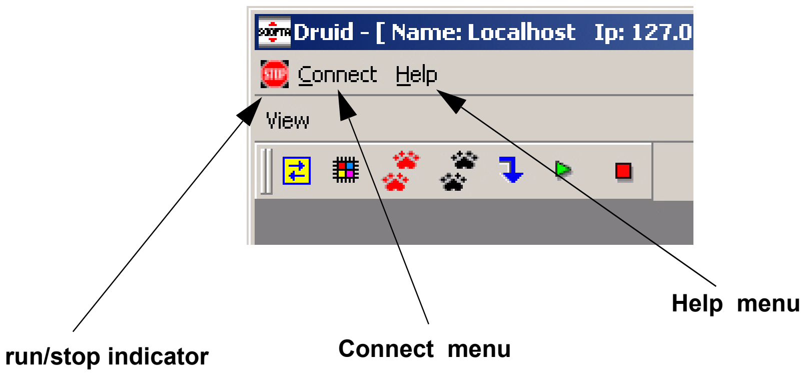
9.3 DRUID Main Window

Maximize the new window and the whole environment will look like this:



9.3.1 Connection and Status Bar

The top level bar consists of the run/stop indicator, the **Connect** menu and the **Help** menu.



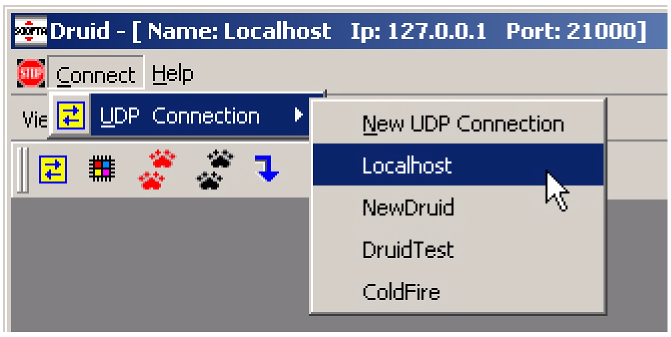
9.3.1.1 Run/Stop Indicator

You can control the SCIOPTA target system from within DRUID. The target can be run or stop manually or it can be stopped if some predefined stop conditions occur.

The run/stop indicator shows always the actual running status of the target system.

9.3.1.2 Connect Menu

The Connect Menu allows you to open an already defined DRUID connection or to define a new connection.



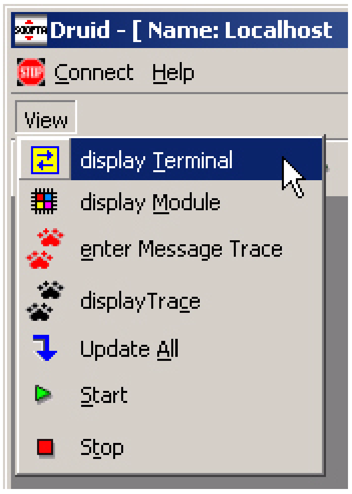
Please consult chapter 9.2 “New Connection” on page 9-2 for more information on how to configure a new connection.

9.3.1.3 Help Menu

The Help Menu gives you access to the SCIOPTA DRUID help system and displays the actual version of DRUID.

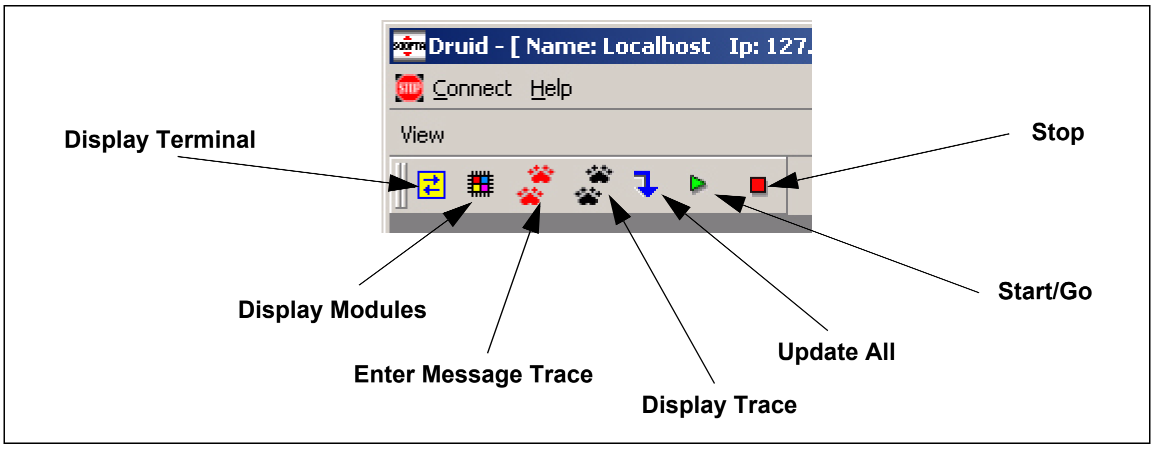
9.3.2 Menu Bar

The menu bar actually consist only of the view menu.



It is a menu with the same entries as in the main DRUID function bar described in the next chapter.

9.3.3 DRUID Main Function Bar



Display Terminal

Opens the terminal window to display the protocol between DRUID (GUI) and DRUID server.

See chapter 9.4 "Display Terminal" on page 9-7.

Display Modules

Opens the display modules window to show all modules in the target. See chapter 9.5 "Display Module" on page 9-8.

Enter Message Trace

Opens the message trace definitions window. Here you can define Trace Options which sets-up what messages to trace.

See chapter 9.13 "Enter Message Trace" on page 9-23.

Display Trace

Opens window with the trace content. See chapter 9.14 "Display Trace" on page 9-28.

Update All

Updates (gets the data from the target system) all windows. See chapter 9.6 "Window Update Policy" on page 9-10.

Start/Go

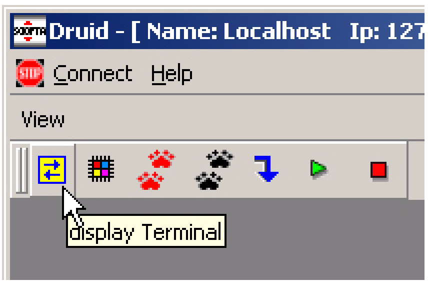
Starts or resumes the target execution. See chapter 9.15 "Start and Stop" on page 9-31.

Stop

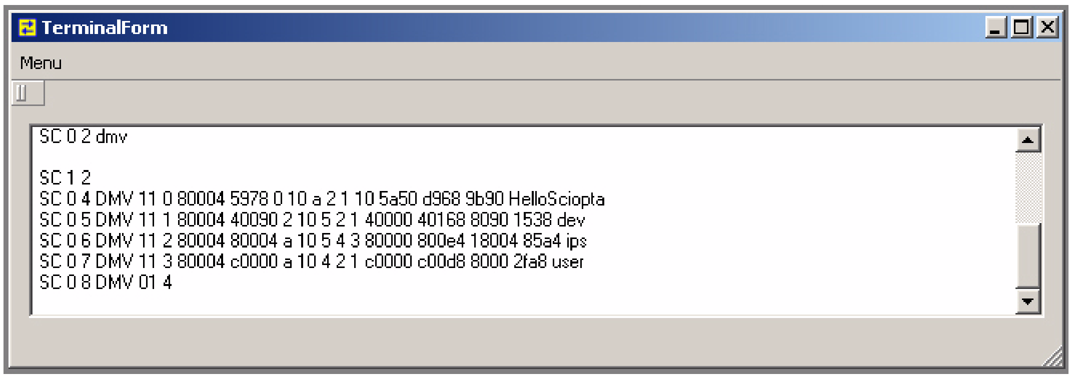
Halts the target execution. See chapter 9.15 "Start and Stop" on page 9-31.

9.4 Display Terminal

The display terminal button opens the UDP terminal window.



Terminal window:



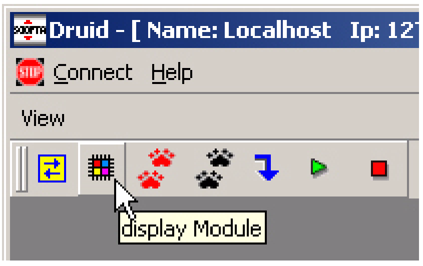
```
SC 0 2 dmv

SC 1 2
SC 0 4 DMV 11 0 80004 5978 0 10 a 2 1 10 5a50 d968 9b90 HelloSciopta
SC 0 5 DMV 11 1 80004 40090 2 10 5 2 1 40000 40168 8090 1538 dev
SC 0 6 DMV 11 2 80004 80004 a 10 5 4 3 80000 800e4 18004 85a4 ips
SC 0 7 DMV 11 3 80004 c0000 a 10 4 2 1 c0000 c00d8 8000 2fa8 user
SC 0 8 DMV 01 4
```

The terminal window shows the data flow between the DRUID GUI and the DRUID server. A specific protocol is used which is not disclosed. This window is only for internal use.

You can check if there is any data flowing to and from the target system (DRUID server) and can get a indication of possible communication problems.

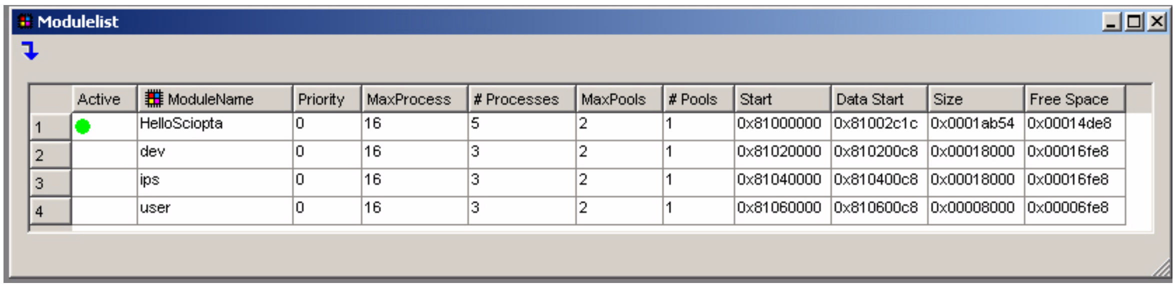
9.5 Display Module

In SCIOPTA processes and pools can be grouped in modules to improve system structure and system protection. To open the Display Module windows and to get information of all modules from the target and to display it you must click on the Display Module Button.



The information will be uploaded from the target and the Module Window opens.

The module window will only show information if the system was initially started and stopped for a first time.



	Active	ModuleName	Priority	MaxProcess	# Processes	MaxPools	# Pools	Start	Data Start	Size	Free Space
1	●	HelloSciopta	0	16	5	2	1	0x81000000	0x81002c1c	0x0001ab54	0x00014de8
2		dev	0	16	3	2	1	0x81020000	0x810200c8	0x00018000	0x00016fe8
3		ips	0	16	3	2	1	0x81040000	0x810400c8	0x00018000	0x00016fe8
4		user	0	16	3	2	1	0x81060000	0x810600c8	0x00008000	0x00006fe8

9.5.1 Module Window Columns

The module window shows a table with all SCIOPTA modules in a system. Eleven columns give detailed information about every module.

9.5.1.1 Active

A green points indicates which module is active, that is in this module the running process is located.

9.5.1.2 Module Name

Name of the module which was given at system configuration (SCONF utility) for static modules or as parameter when a module was dynamically created.

9.5.1.3 Priority

Module priority which can have a value of 0 to 31.

9.5.1.4 MaxProcess

Maximum number of processes of the module which was given at system configuration (SCONF utility) for static modules or as parameter when a module was dynamically created.

9.5.1.5 # Processes

Actual number of static and dynamic processes in the module.

9.5.1.6 MaxPools

Maximum number of pools of the module which was given at system configuration (SCONF utility) for static modules or as parameter when a module was dynamically created.

9.5.1.7 # pools

Actual number of static and dynamic pools in the module.

9.5.1.8 Start

This is start address of the data area of the module which was given at system configuration (SCONF utility) for static modules (or calculated by the linker script) or as parameter when a module was dynamically created.

9.5.1.9 Data Start

This is the start address for the memory area where the module holds the static and dynamic operating system objects such as control blocks, pools and stacks.

9.5.1.10 Size

Size of the module

9.5.1.11 Free Space

Available free data space for the module.

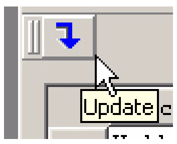
9.6 Window Update Policy

This is the place to give some information about the window update policy in DRUID.

In DRUID you can have different window open containing information about modules, processes, pools and trace. Once uploaded from target DRUID keeps a copy of these windows locally in the host.

If you have uploaded a window content and then started the target and after a while stopped it there is still the local copy displayed in that window. To show that this window may contain invalid data (as the target has executed code in the meantime) all information is now red coloured.

You can now update the window content by selecting the **Update button** in the tool bar of the same window:



The new values will be uploaded and the window will be updated. The window content is coloured black again to show up-to-date data.

There is an **Update button** in each window containing target data. But there is also a global Update button located in the DRUID main function bar which will update all open windows.

Selecting the **Update button** on a window with valid (black coloured) data will not sending any requests to the target system.

Every time you are opening a **new window** new data will uploaded into this window if the old window contains old (red coloured) data.

9.6.1 Comparing System Data

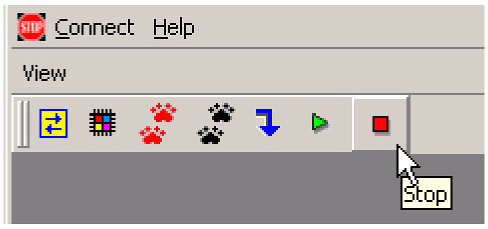
One interesting feature is to compare system data between two system states.

This can be done the following way:

1. Stop the target system after a first run.
2. Get target data (i.e display module, processes, pools, trace etc.).
3. Restart the target system.
4. Stop the target system again.
5. The colour in the open data window changes to red and shows that the data in the target might have changed.
6. Open a new data window for the same type as the first window.
7. A new window with valid (black coloured) data opens.
8. Now you can compare the data of the two windows which indicates differences between two target system states.

9.7 Running Modes

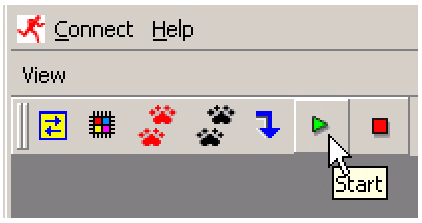
9.7.1 Stop Mode



The usual and most controlled and secure way to do system level debugging with DRUID is to examine the system while the target system is halted.

This is what we are calling **Stop Mode**.

9.7.2 Execution Mode



But you can run all DRUID commands during the target system is running. This is giving you a kind of system snap-shot.

This what we are calling **Execution Mode**.

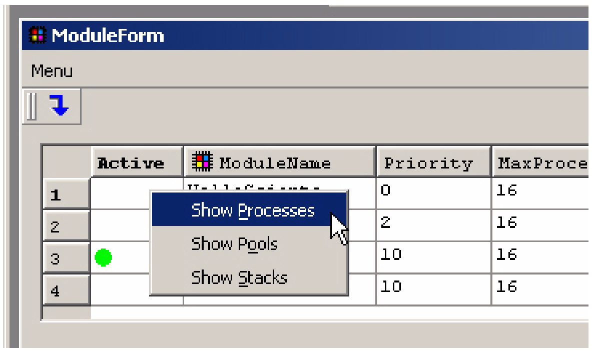
All data in the windows in **Execution Mode** are always red coloured. This is just to show that this data is only valid for a short snap-shot period and remains invalid as the target system continues to execute.

Please use Execution Mode with care. This mode is by far not non-intrusive and might slow-down the target system considerably.

9.8 Show Processes

To show the processes of a specific module you need to have an open **Display Module** window. Please consult chapter **9.5 “Display Module” on page 9-8** for information on how to open the Display Module window.

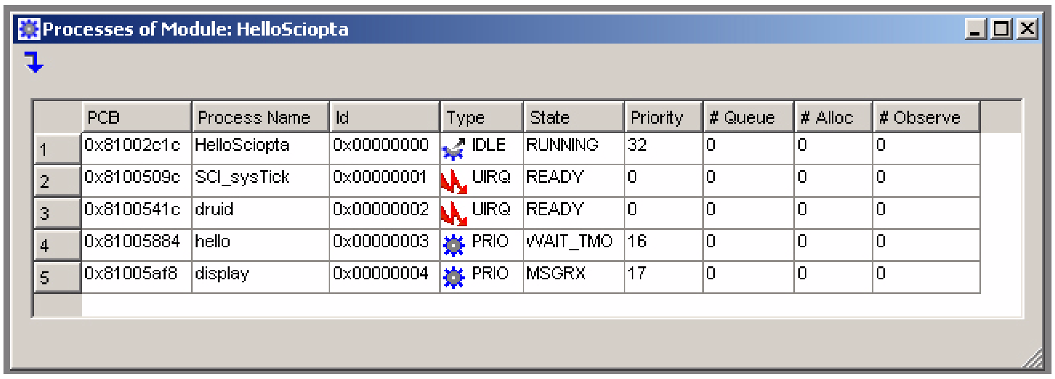
Right click in any cell of a module to open the **Show** window. Select **Show Processes**.



	Active	ModuleName	Priority	MaxProce
1			0	16
2			2	16
3	●		10	16
4			10	16

Double clicking in the cell of a **Module Name**, **MaxProcess** or **Number of Processes column** will also open a **Show Processes** window.

This will open an **Show Processes** window.



	PCB	Process Name	Id	Type	State	Priority	# Queue	# Alloc	# Observe
1	0x81002c1c	HelloSciopta	0x00000000	IDLE	RUNNING	32	0	0	0
2	0x8100509c	SCI_sysTick	0x00000001	UIRQ	READY	0	0	0	0
3	0x8100541c	druid	0x00000002	UIRQ	READY	0	0	0	0
4	0x81005884	hello	0x00000003	PRIO	WAIT_TMO	16	0	0	0
5	0x81005af8	display	0x00000004	PRIO	MSGRX	17	0	0	0

9.8.1 Show Processes Window Columns

The Show Processes window shows a table with all SCIOPTA processes in a SCIOPTA module. Six columns give detailed information about every process.

9.8.1.1 PCB

This is the address of the process control block (PCB).

9.8.1.2 Process Name

Name of the process which was given at system configuration (SCONF utility) for static processes or as parameter when a process was dynamically created.

9.8.1.3 ID

Identity (ID) of the process.

9.8.1.4 Type

Shows the process type. Please Note: The normal coloured icon (as showed below) indicates a statically created process. For a dynamically created process the icon is coloured in green.

- **IDLE** (IDLE or INIT process)

  The init process is the first process in a module Each module has at least one process and this is the init process. At module start the init process gets automatically the highest priority (0). After the init process has done some important work it will change its priority to the lowest level (32) and enter an endless loop. Priority 32 is only allowed for the init process. All other processes are using priority 0 - 31. The init process acts therefore also as idle process which will run when all other processes of a module are in the waiting state.
- **IRQ** (Interrupt process)

  An interrupt is a system event generated by a hardware device. The CPU will suspend the actually running program and activate an interrupt service routine assigned to that interrupt. The programs which handle interrupts are called interrupt processes in SCIOPTA. SCIOPTA is channelling interrupts internally and calls the appropriate interrupt process. The priority of an interrupt process is assigned by hardware of the interrupt source. Whenever an interrupt occurs the assigned interrupt process is called, assuming that no other interrupt of higher priority is running. If the interrupt process with higher priority has completed his work, the interrupt process of lower priority can continue.
- **UIRQ** (User interrupt process)

  Same as an interrupt process, but an user interrupt process is handled outside of the kernel.

- **TIM** (Timer process)

  A timer process in SCIOPTA is a specific interrupt process connected to the tick timer of the operating system. SCIOPTA is calling each timer process periodically derived from the operating system tick counter. When configuring or creating a timer process, the user defines the number of system ticks to expire from one call to the other individually for each process.

- PROI (Prioritized process)

  In a typical SCIOPTA system prioritized processes are the most common used process types. Each prioritized process has a priority and the SCIOPTA scheduler is running ready processes according to these priorities. The process with higher priority before the process with lower priority. If a process has terminated its job for the moment by for example waiting on a message which has not yet been sent or by calling the kernel sleep function, the process is put into the waiting state and is not any longer ready.

9.8.1.5 State

Basically a process running under SCIOPTA is always in the RUNNING, READY or WAITING state. The following process states are defined:

ST_READY	Process is ready. It is ready to run and wants the CPU but there is another process with higher priority running.
ST_MSGRX	Process waits for a message
ST_RUNNING	Process is running
ST_TMO	a time-out occurred
ST_WAIT_TMO	Process is waiting in a time-out

9.8.1.6 Priority

This column shows the process priority. The priority can be from 0..31 (0 is the highest and 31 the lowest). Only the init (or idle) process can have a priority of 32. Interrupt processes are displayed as having a priority of 0.

Please Note:

Each SCIOPTA process and module has a specific priority. The user defines the priorities at system configuration or when creating the module or the process. Process and module priorities can be modified during run-time.

For process scheduling SCIOPTA uses a combination of the module priority and process priority called **effective priority**. The kernel determines the effective priority as follows:

EFFECTIVE PRIORITY = MODULE PRIORITY + PROCESS PRIORITY

This technique assures that the process with highest process priority (0) cannot disturb processes in modules with lower module priority (module protection).

9.8.1.7 # Queue

Number of message in the message queue of the process. These are the messages which are sent to the process but not yet received.

9.8.1.8 # Alloc

Number of owned messages by the process. These are messages which are allocated or received by the process.

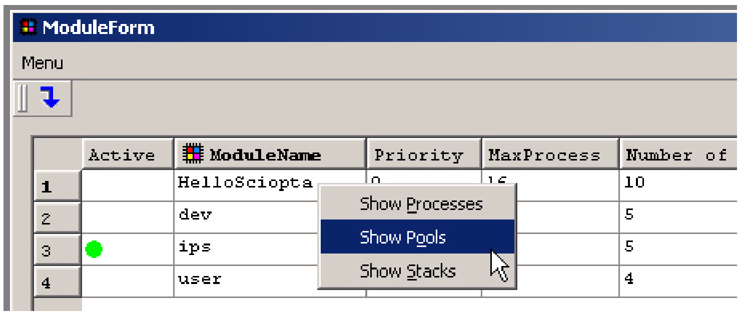
9.8.1.9 # Obsereve

Number of observe messages connected to the process. Please consult the SCIOPTA Kernel, User's Guide for more information about process observation.

9.9 Show Pools

To show the pools of a specific module you need to have an open **Display Module** window. Please consult chapter 9.5 “Display Module” on page 9-8 for information on how to open the Display Module window.

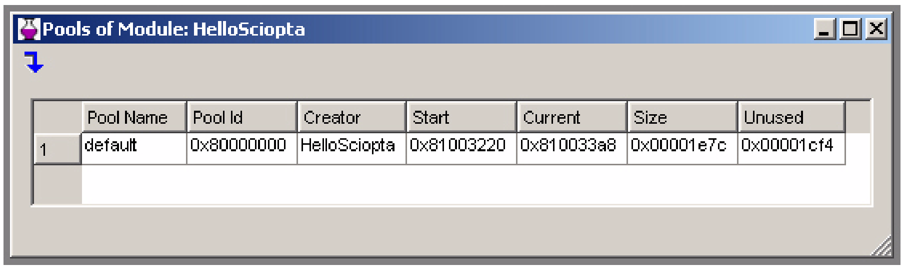
Right click in any cell of a module to open the **Show** window. Select **Show Pools**.



	Active	ModuleName	Priority	MaxProcess	Number of
1		HelloSciopta	0	16	10
2		dev			5
3	●	ips			5
4		user			4

Double clicking in the cell of a **Max Pools** or **Number of Pools column** will also open a **Show Pools** window.

This will open a **Show Pools** window.



	Pool Name	Pool Id	Creator	Start	Current	Size	Unused
1	default	0x80000000	HelloSciopta	0x81003220	0x810033a8	0x00001e7c	0x00001cf4

9.9.1 Show Pools Window Columns

The Show Pools window shows a table with all SCIOPTA pools in a SCIOPTA module. Seven columns give detailed information about every pool.

9.9.1.1 Pool Name

Name of the pool which was given at system configuration (SCONF utility) for static pools or as parameter when a pool was dynamically created.

9.9.1.2 Pool ID

Identity (ID) of the pool.

9.9.1.3 Creator

Module from where the pool was created.

9.9.1.4 Start

Start address in the memory of the pool.

9.9.1.5 Current

Current address of the pool pointer.

9.9.1.6 Size

Size of the pool

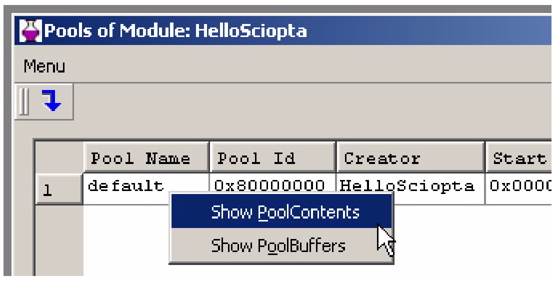
9.9.1.7 Unused

Unused memory of the pool. This is memory which has not yet been used and available for allocating new messages.

9.10 Show Pool Content

To show the pool content of a specific pool you need to have an open **Show Pools** window. Please consult chapter 9.9 “Show Pools” on page 9-16 for information on how to open the Show Pools window.

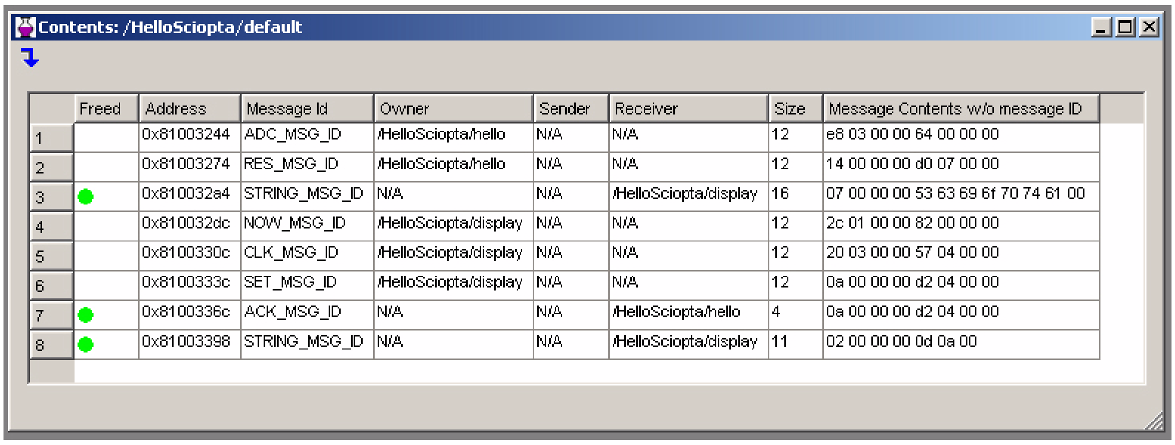
Right click in any cell of a pool to open the **Show** window. Select **Show PoolContents**.



Pools of Module: HelloSciopta

	Pool Name	Pool Id	Creator	Start
1	default	0x80000000	HelloSciopta	0x0000

This will open an **Show PoolContents** window.



Contents: /HelloSciopta/default

	Freed	Address	Message Id	Owner	Sender	Receiver	Size	Message Contents w/o message ID
1		0x81003244	ADC_MSG_ID	/HelloSciopta/hello	N/A	N/A	12	e8 03 00 00 64 00 00 00
2		0x81003274	RES_MSG_ID	/HelloSciopta/hello	N/A	N/A	12	14 00 00 00 d0 07 00 00
3	●	0x810032a4	STRING_MSG_ID	N/A	N/A	/HelloSciopta/display	16	07 00 00 00 53 63 69 6f 70 74 61 00
4		0x810032dc	NOW_MSG_ID	/HelloSciopta/display	N/A	N/A	12	2c 01 00 00 82 00 00 00
5		0x8100330c	CLK_MSG_ID	/HelloSciopta/display	N/A	N/A	12	20 03 00 00 57 04 00 00
6		0x8100333c	SET_MSG_ID	/HelloSciopta/display	N/A	N/A	12	0a 00 00 00 d2 04 00 00
7	●	0x8100336c	ACK_MSG_ID	N/A	N/A	/HelloSciopta/hello	4	0a 00 00 00 d2 04 00 00
8	●	0x81003398	STRING_MSG_ID	N/A	N/A	/HelloSciopta/display	11	02 00 00 00 0d 0a 00

9.10.1 Show Pool Content Window Columns

The Show Pools Content window shows a table with all SCIOPTA messages in a SCIOPTA pool. Eight columns give detailed information about every message.

9.10.1.1 Freed

Shows the status of the message. A green point indicates that the message is freed. In this case the buffer of this message is available for a new (sc_msgAlloc) for a message of this size.

9.10.1.2 Address

Address of the message in the memory space.

9.10.1.3 Message ID

Identity (ID) of the message.

If you have given a name and location of a message name database when the DRUID server was started (-m option) the message name will be displayed. Please consult chapter **8.1.3 "DRUID Server Options" on page 8-2** for more information. Otherwise the message ID in hex format will be displayed.

9.10.1.4 Owner

Process which owns the message.

9.10.1.5 Sender

Last sender process of the message.

9.10.1.6 Receiver

Receiver or addressee process of the message.

9.10.1.7 Size

Returned real size of the message.

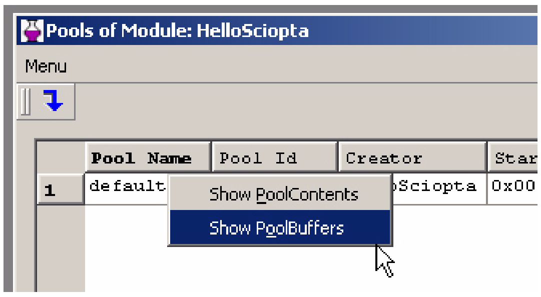
9.10.1.8 Message Content w/o Message ID

Shows the first 16 bytes of data of the message.

9.11 Show Pool Buffers

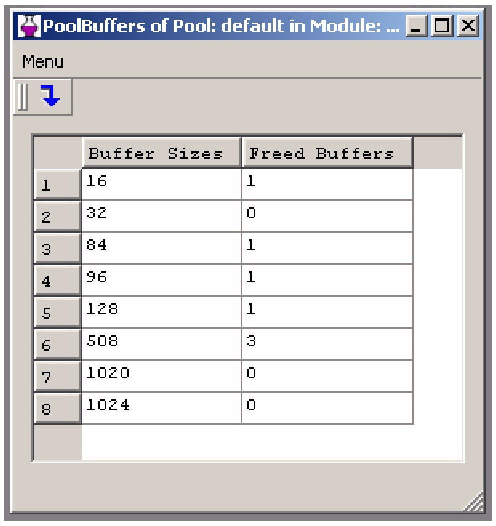
To show the pool buffers of a specific pool you need to have an open **Show Pools** window. Please consult chapter **9.9 "Show Pools" on page 9-16** for information on how to open the Show Pools window.

Right click in any cell of a pool to open the **Show** window. Select **Show PoolBuffers**.



	Pool Name	Pool Id	Creator	Star
1	default		Sciopta	0x00

This will open an **Show PoolBuffers** window.



	Buffer Sizes	Freed Buffers
1	16	1
2	32	0
3	84	1
4	96	1
5	128	1
6	508	3
7	1020	0
8	1024	0

9.11.1 Show Pool Buffers Window Columns

The Show Pools Buffer window shows a table with the different SCIOPTA messages sizes in a SCIOPTA pool. Two columns give detailed information about every message size.

9.11.1.1 Buffer Sizes

Shows the configured size of a buffer type.

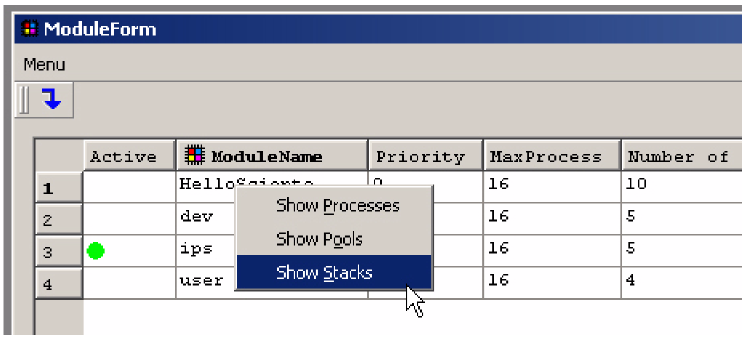
9.11.1.2 Freed Buffers

Shows the number of freed buffers of that size. In this case the buffers are available for a new allocation (sc_msgAlloc system call) of a message of this size.

9.12 Show Stacks

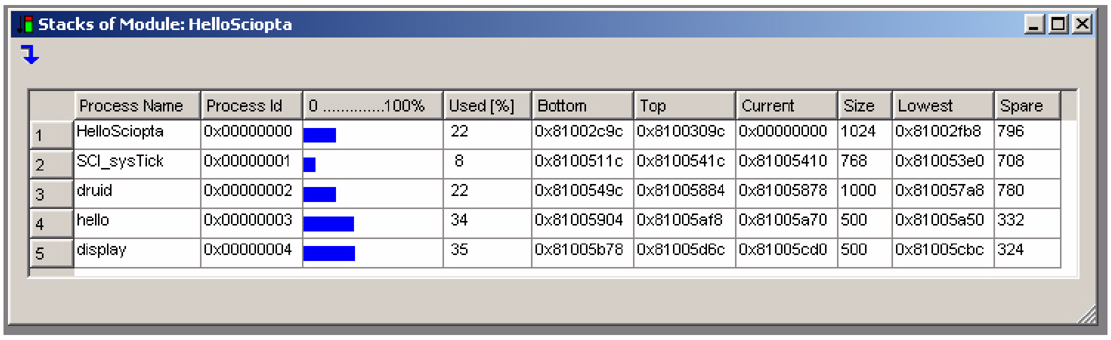
To show the process stacks of a specific module you need to have an open **Display Module** window. Please consult chapter 9.5 "Display Module" on page 9-8 for information on how to open the Display Module window.

Right click in any cell of a module to open the **Show** window. Select **Show Stacks**.



	Active	ModuleName	Priority	MaxProcess	Number of
1		HelloSciopta	0	16	10
2		dev		16	5
3	●	ips		16	5
4		user		16	4

This will open an **Show Stacks** window.



	Process Name	Process Id	0 ....100%	Used [%]	Bottom	Top	Current	Size	Lowest	Spare
1	HelloSciopta	0x00000000		22	0x81002c9c	0x8100309c	0x00000000	1024	0x81002fb8	796
2	SCI_sysTick	0x00000001		8	0x8100511c	0x8100541c	0x81005410	768	0x810053e0	708
3	druid	0x00000002		22	0x8100549c	0x81005884	0x81005878	1000	0x810057a8	780
4	hello	0x00000003		34	0x81005904	0x81005af8	0x81005a70	500	0x81005a50	332
5	display	0x00000004		35	0x81005b78	0x81005d6c	0x81005cd0	500	0x81005cbc	324

9.12.1 Show Stack Window Columns

The Show Stack window shows a table with all SCIOPTA process stacks in a SCIOPTA module. Ten columns give detailed information about every process stack.

9.12.1.1 Process Name

Name of the process which was given at system configuration (SCONF utility) for static processes or as parameter when a process was dynamically created.

9.12.1.2 Process ID

Identity (ID) of the process.

9.12.1.3 0 ....100%

A blue bar indicates the number of maximum bytes ever used for the stack of that process.

9.12.1.4 Used [%]

Same indication as the previous chapter but in the value of percent.

9.12.1.5 Bottom

Address of the lower end of the stack frame.

9.12.1.6 Top

Address of the higher end of the stack frame.

9.12.1.7 Current

Actual value of the process stack pointer.

9.12.1.8 Size

Size of the stack in bytes.

9.12.1.9 Lowest

Indicates the lower ever reached stack pointer of that process.

9.12.1.10 Spare

Worst case still free available stack size.

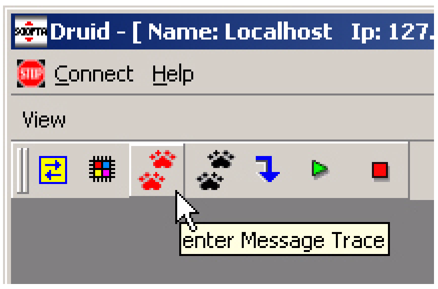
9.13 Enter Message Trace

Message trace is a strong feature of DRUID. The user can configure target memory to be reserved for message trace. You can setup **trace-start** and **trace-stop** conditions and DRUID will trace **selected messages** between start and stop.

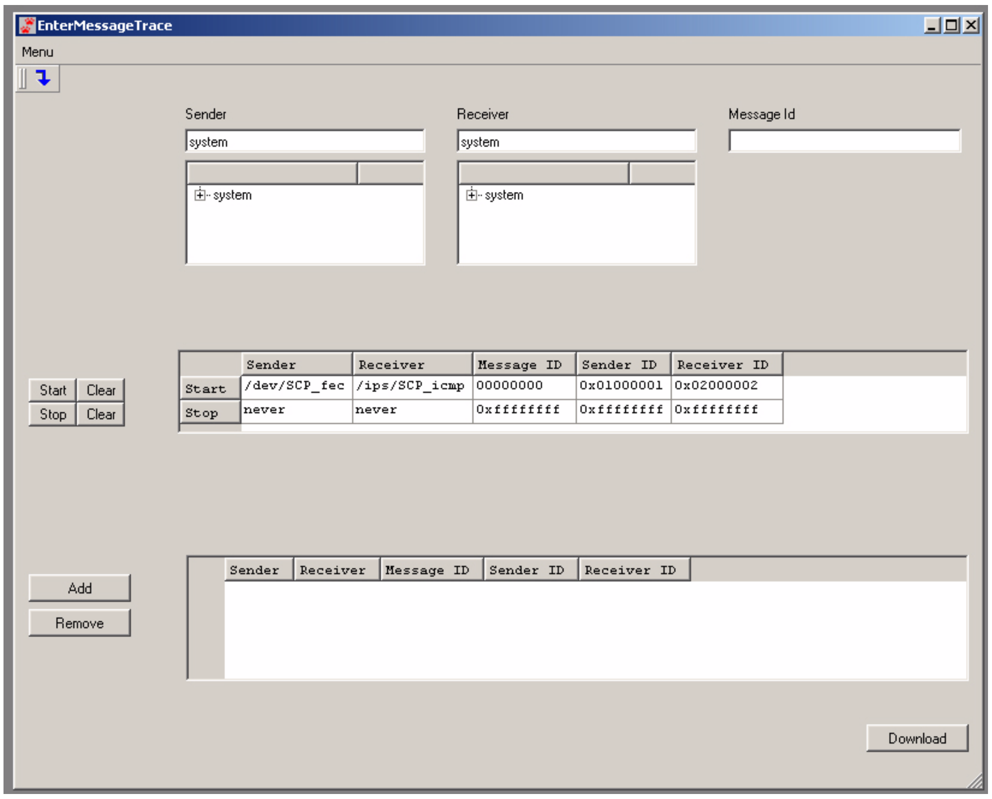
There is a **Trace Option List** where the user defines what messages to trace. For a receiver or a sender the module or process can be defined. The message ID can be given as a further option.

Trace Options and the Trace Option List can be defined and edited using the DRUID EnterMessageTrace window.

To open the EnterMessageTrace window please click on the enter Message Trace Button.



The EnterMessageTrace window opens and the Trace Option List is uploaded from the target if there are Trace Options defined.



	Sender	Receiver	Message ID	Sender ID	Receiver ID
Start	/dev/SCP_fec	/ips/SCP_icmp	00000000	0x01000001	0x02000002
Stop	never	never	0xffffffff	0xffffffff	0xffffffff

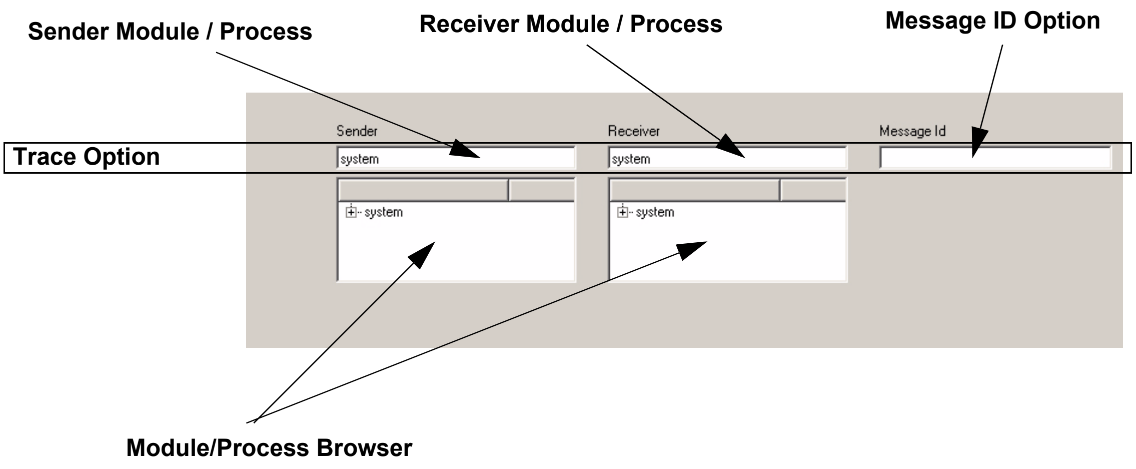
Sender	Receiver	Message ID	Sender ID	Receiver ID

The EnterMessageTrace window is divided into three parts:

- Trace Option editor
- Trace Trigger
- Trace Option List

9.13.1 Trace Option Editor

The Trace Option editor is located in the upper part of the EnterMessageTrace window.



In the Module or Process Browser you have access to all existing modules and processes of a system. To transfer a module or a process to the trace option field above you need to **double-click** on the module or process.

You can also select **system**. In this case the message (or messages) from (or to) the whole system will be traced.

You can do this for the sender and the receiver.

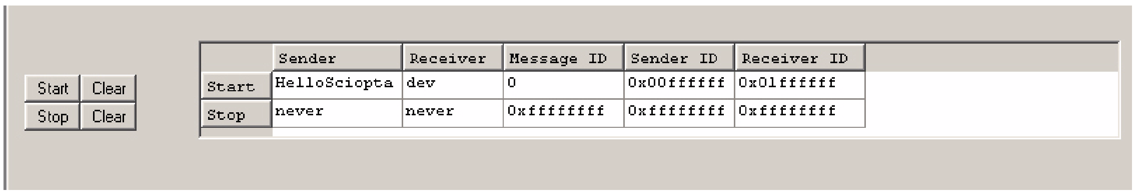
Additionally you can also define a message ID. This will specify a Trace Option for a specific message from a specific module/process to a specific module or process.

A message ID of 0 (or an empty field) will specify **all** messages.

Once satisfied with the Trace Option you can click on the **Add** button of the Trace Option List to add the Trace Option or click on the **Start** or **Stop** button of the Trace Trigger to add the Trace Option as trigger condition.

9.13.2 Trace Trigger

The Trace Trigger field is located in the upper part of the EnterMessageTrace window.



	Sender	Receiver	Message ID	Sender ID	Receiver ID
Start	HelloSciopta	dev	0	0x00ffffff	0x01ffffff
Stop	never	never	0xffffffff	0xffffffff	0xffffffff

This field allows to define a start and a stop trigger condition for the message trace. The trace will be started or stopped if the defined message (or messages) are sent.

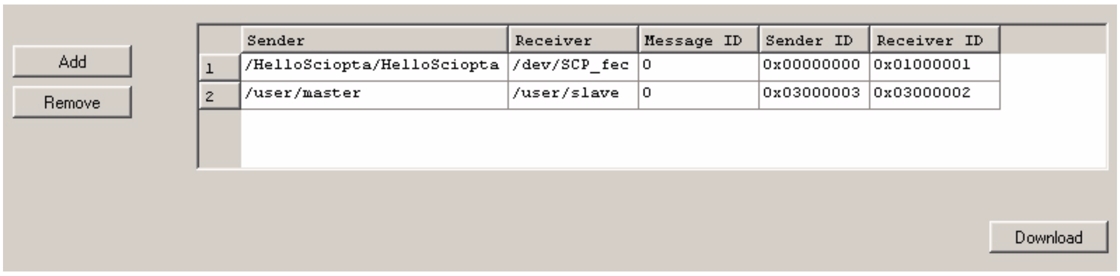
The **Clear** button will set the system to it initial state. Trace will be started at the first message in the system and will never be stopped.

For any other message condition for a start or stop trigger you need to edit it first in the Trace Option Editor (see chapter **9.13.1 "Trace Option Editor" on page 9-24**). By clicking on the **Start** or **Stop** button the Trace Option will be taken-in as a trace trigger.

The **Download** button of the Trace Option List field will download your trigger setting to the target system.

9.13.3 Trace Option List

The Trace Option List field is located in the lower part of the EnterMessageTrace window.



	Sender	Receiver	Message ID	Sender ID	Receiver ID
1	/HelloSciopta/HelloSciopta	/dev/SCP_fec	0	0x00000000	0x01000001
2	/user/master	/user/slave	0	0x03000003	0x03000002

First you need to edit a Trace Option in the Trace Option Editor (see chapter **9.13.1 “Trace Option Editor” on page 9-24**).

Once satisfied with the Trace Option you can click on the **Add** button of the Trace Option List to add the Trace Option to the list.

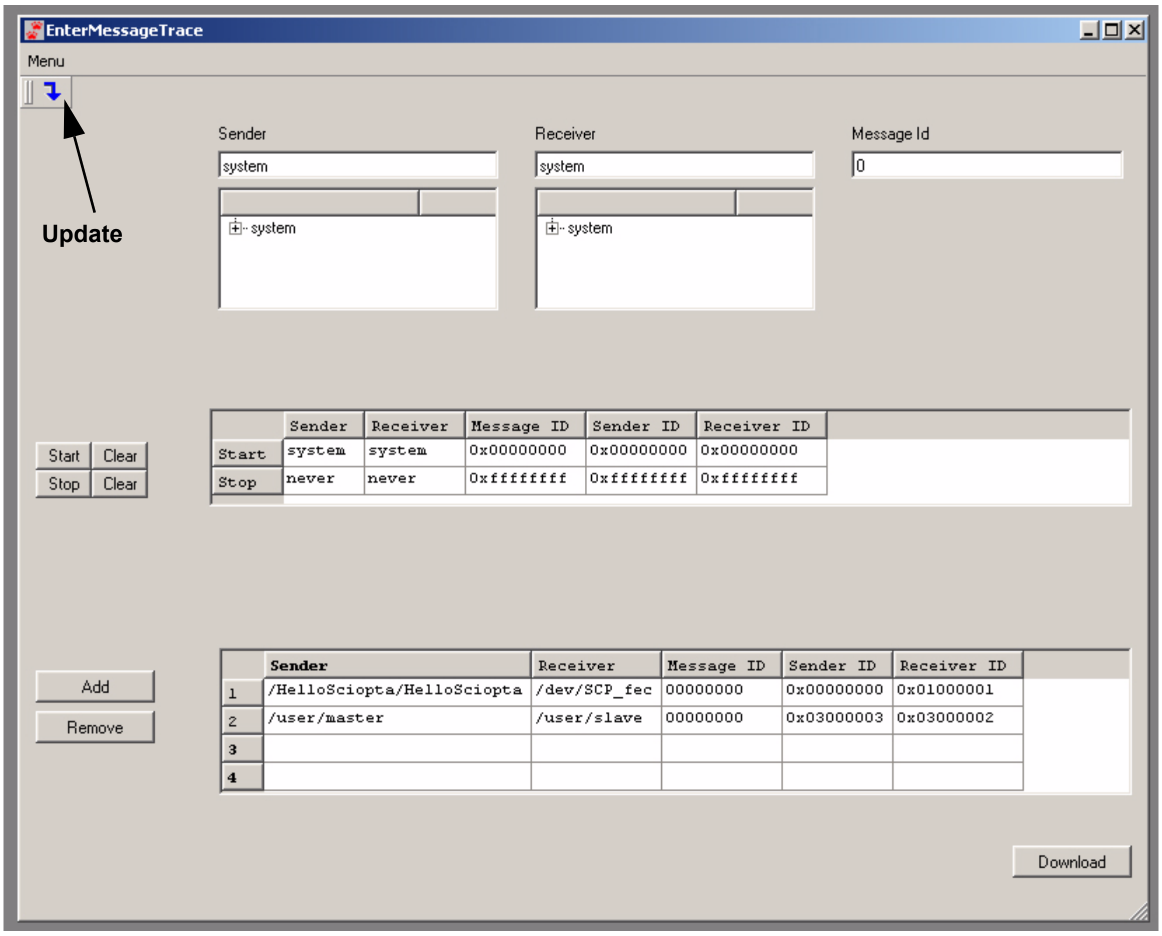
The number of Trace Options is limited by a target definition. This limitation is configurable (please see chapter **5.4.2.1 “CNF_DRUID_OPTION_MAX” on page 5-7**). If the limit is reached you cannot add more Trace Option to the list.

The **Download** button will download the Trace Option List to the target system.

To remove a Trace Option just click into one of the cells of the trace option you want to remove and click on the **Remove** button.

9.13.4 Update Trace Trigger and Trace Option List

The update button in the EnterMessageTrace windows will upload the trace trigger settings and Trace Option List from the target system.



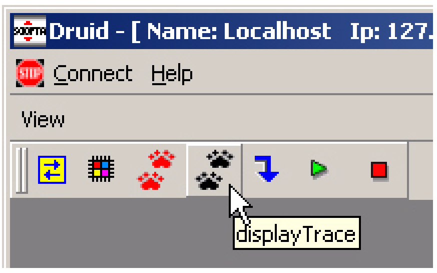
	Sender	Receiver	Message ID	Sender ID	Receiver ID
Start	system	system	0x00000000	0x00000000	0x00000000
Stop	never	never	0xffffffff	0xffffffff	0xffffffff

	Sender	Receiver	Message ID	Sender ID	Receiver ID
1	/HelloSciopta/HelloSciopta	/dev/SCP_fec	00000000	0x00000000	0x01000001
2	/user/master	/user/slave	00000000	0x03000003	0x03000002
3					
4					

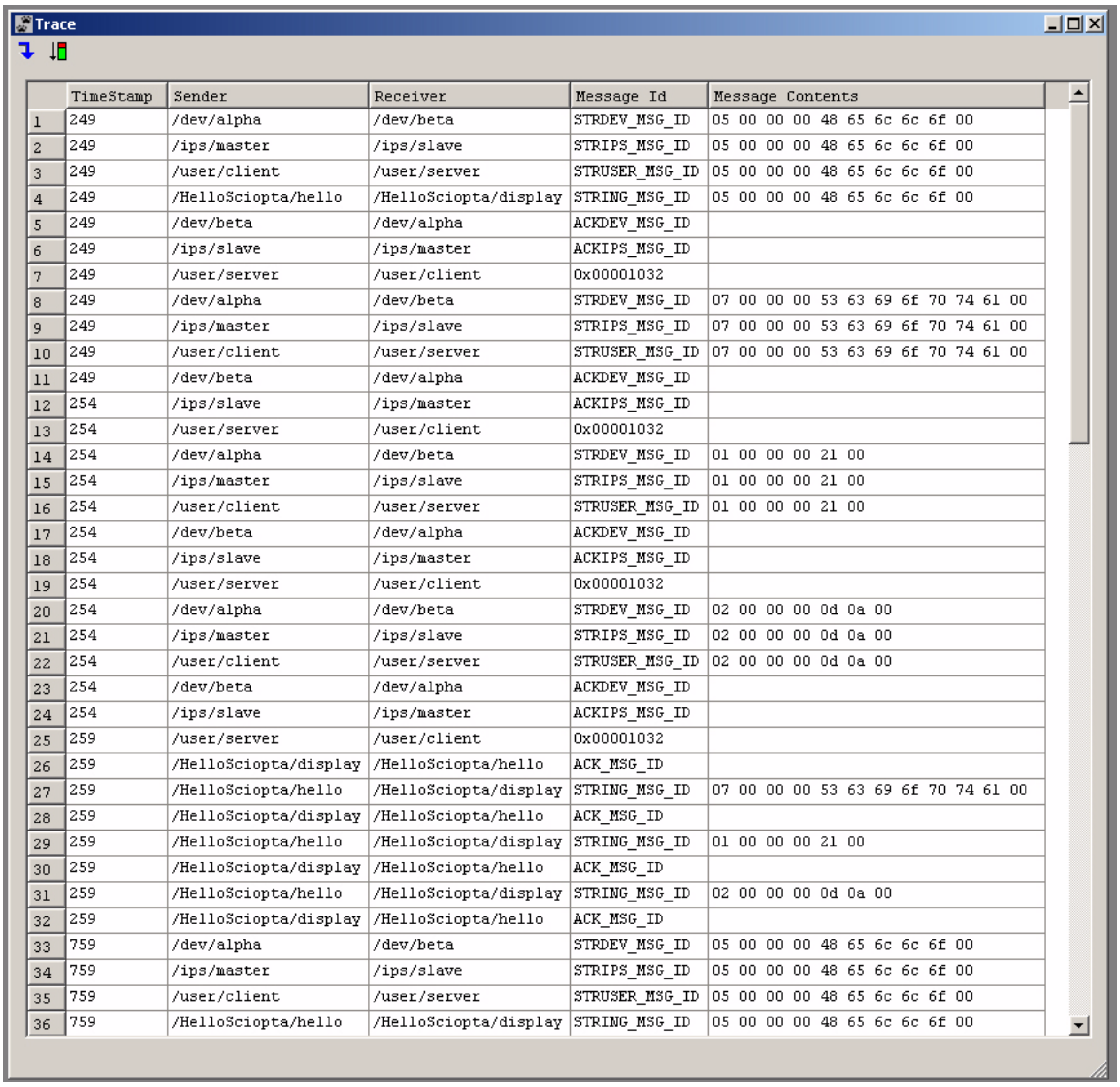
The global update button has the same effect.

9.14 Display Trace

To open the EnterMessageTrace window please click on the enter Display Trace Button.



The Display Trace window opens and the Trace Content is uploaded from the target.



	TimeStamp	Sender	Receiver	Message Id	Message Contents
1	249	/dev/alpha	/dev/beta	STRDEV_MSG_ID	05 00 00 00 48 65 6c 6c 6f 00
2	249	/ips/master	/ips/slave	STRIPS_MSG_ID	05 00 00 00 48 65 6c 6c 6f 00
3	249	/user/client	/user/server	STRUSER_MSG_ID	05 00 00 00 48 65 6c 6c 6f 00
4	249	/HelloSciopta/hello	/HelloSciopta/display	STRING_MSG_ID	05 00 00 00 48 65 6c 6c 6f 00
5	249	/dev/beta	/dev/alpha	ACKDEV_MSG_ID	
6	249	/ips/slave	/ips/master	ACKIPS_MSG_ID	
7	249	/user/server	/user/client	0x00001032	
8	249	/dev/alpha	/dev/beta	STRDEV_MSG_ID	07 00 00 00 53 63 69 6f 70 74 61 00
9	249	/ips/master	/ips/slave	STRIPS_MSG_ID	07 00 00 00 53 63 69 6f 70 74 61 00
10	249	/user/client	/user/server	STRUSER_MSG_ID	07 00 00 00 53 63 69 6f 70 74 61 00
11	249	/dev/beta	/dev/alpha	ACKDEV_MSG_ID	
12	254	/ips/slave	/ips/master	ACKIPS_MSG_ID	
13	254	/user/server	/user/client	0x00001032	
14	254	/dev/alpha	/dev/beta	STRDEV_MSG_ID	01 00 00 00 21 00
15	254	/ips/master	/ips/slave	STRIPS_MSG_ID	01 00 00 00 21 00
16	254	/user/client	/user/server	STRUSER_MSG_ID	01 00 00 00 21 00
17	254	/dev/beta	/dev/alpha	ACKDEV_MSG_ID	
18	254	/ips/slave	/ips/master	ACKIPS_MSG_ID	
19	254	/user/server	/user/client	0x00001032	
20	254	/dev/alpha	/dev/beta	STRDEV_MSG_ID	02 00 00 00 0d 0a 00
21	254	/ips/master	/ips/slave	STRIPS_MSG_ID	02 00 00 00 0d 0a 00
22	254	/user/client	/user/server	STRUSER_MSG_ID	02 00 00 00 0d 0a 00
23	254	/dev/beta	/dev/alpha	ACKDEV_MSG_ID	
24	254	/ips/slave	/ips/master	ACKIPS_MSG_ID	
25	259	/user/server	/user/client	0x00001032	
26	259	/HelloSciopta/display	/HelloSciopta/hello	ACK_MSG_ID	
27	259	/HelloSciopta/hello	/HelloSciopta/display	STRING_MSG_ID	07 00 00 00 53 63 69 6f 70 74 61 00
28	259	/HelloSciopta/display	/HelloSciopta/hello	ACK_MSG_ID	
29	259	/HelloSciopta/hello	/HelloSciopta/display	STRING_MSG_ID	01 00 00 00 21 00
30	259	/HelloSciopta/display	/HelloSciopta/hello	ACK_MSG_ID	
31	259	/HelloSciopta/hello	/HelloSciopta/display	STRING_MSG_ID	02 00 00 00 0d 0a 00
32	259	/HelloSciopta/display	/HelloSciopta/hello	ACK_MSG_ID	
33	759	/dev/alpha	/dev/beta	STRDEV_MSG_ID	05 00 00 00 48 65 6c 6c 6f 00
34	759	/ips/master	/ips/slave	STRIPS_MSG_ID	05 00 00 00 48 65 6c 6c 6f 00
35	759	/user/client	/user/server	STRUSER_MSG_ID	05 00 00 00 48 65 6c 6c 6f 00
36	759	/HelloSciopta/hello	/HelloSciopta/display	STRING_MSG_ID	05 00 00 00 48 65 6c 6c 6f 00

9.14.1 Display Trace Window Columns

The Display Trace window shows the message trace content.

9.14.1.1 TimeStamp

Message trace time stamp in number of system ticks.

9.14.1.2 Sender

Identity (ID) of the sender process.

9.14.1.3 Receiver

Identity (ID) of the receiver process.

9.14.1.4 Message ID

Identity (ID) of the message.

If you have given a name and location of a message name database when the DRUID server was started (-m option) the message name will be displayed. Please consult chapter **8.1.3 “DRUID Server Options” on page 8-2** for more information. Otherwise the message ID in hex format will be displayed.

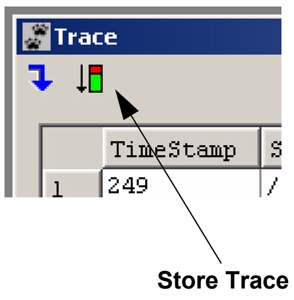
9.14.1.5 Message Contents

A configurable number of first data bytes are displayed.

The number of displayed first message data bytes is limited by a target definition. This limitation is configurable (please see chapter **5.4.2.3 “CNF_DRUID_MSG_DATA_MAX” on page 5-7**).

9.14.2 Store Trace

The store trace button allows to store the message trace content into a file.



The file name is fixed to: sc_traceLog.txt.

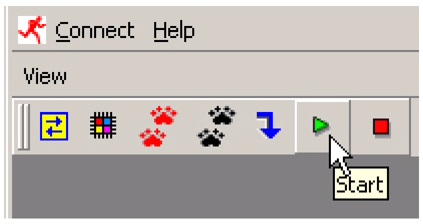
The file will be stored at the following fixed location:

..\Documents and Settings\<user_name>\

In the next version the storage location will be user selectable.

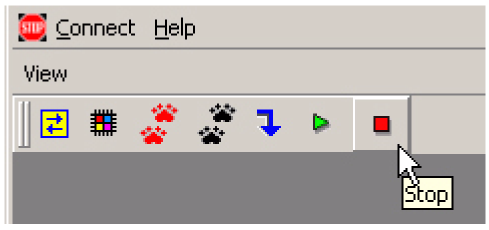
9.15 Start and Stop

9.15.1 Start



Clicking on the **Start** button will start or resume the target execution.

9.15.2 Stop



Clicking on the Stop button will stop the target execution.

10 SCIOPTA ARM Releases

10.1 SCIOPTA ARM Release Notes

On the **SCIOPTA ARM** CD in the subfolder **doc** you will find a text file named **RelNotes_ARM.txt**

This file contains a description of the changes of the actual version compared to the last delivered version. It allows you to decide if you want to install and use the new version.

10.2 Installed Files

On the **SCIOPTA ARM** CD in the subfolder **doc** you will find a text file named **revisions.txt**

This file contains a list of all installed files including the following information for each file:

- File name
- SCIOPTA document number
- File version
- File description

11 Manual Versions

11.1 Manual Version 2.1

- Chapter 2 Installation, Main Installation Window screen shot changed.

11.2 Manual Version 2.0

- The following manuals have been combinded in this new SCIOPTA ARM - DRUID, User's Guide:
  - SCIOPTA - DRUID, System Level Debugger User's Guide Version 1.1
  - SCIOPTA - ARM Target Manual

11.3 Former SCIOPTA - DRUID, User's Guide Versions

11.3.1 Manual Version 1.1

- Back front page, Litronic AG became SCIOPTA Systems AG.
- Chapter 3.2.2.2 Process Hooks, added.
- Chapter 3.2.2.3 Pool Hooks, added.
- Chapter 3.3 Board Setup File, added.
- Chapter 3.4 Trace Memory Setup, CNF_DRUID_MA_MAX replaced by CNF_DRUID_OPTION_MAX, CNF_DRUID_MTD_MAX replaced by CNF_DRUID_TRACE_MEM_MAX and CNF_DRUID_MTD_DATA_MAX replaced by CNF_DRUID_MSG_DATA_MAX.
- Chapter 3.5.3 Board Setup File, header renamed.
- Chapter 4.1.2 Starting DRUID Server, starting now only from command shell described. DRUID server is not any longer installed on the desktop and the menu, due to use of command line parameters.
- Chapter 4.1.3 DRUID Server Options, <baudrate> is now -v switch, new switches: -m, -e and -b.
- Chapter 5.5 Display Module, third paragraph added.
- Chapter 5.5. Display Module, columns rearranged and renamed.
- Chapter 5.8, Show Processes, some columns rearranged and renamed. New columns: #Queue, #Alloc and #Observe added.
- Chapter 5.9. Show Pool, some columns rearranged and renamed. New column: Unused.
- Chapter 5.10.1.3 Message ID, information on message name display added.
- Chapter 5.12 Show Stacks, some columns renamed.
- Chapter 5.14.1.1 TimeStamp, added.
- Chapter 5.14.2 Store Trace, added.

11.3.2 Manual Version 1.0

- Initial version.

11.4 Former SCIOPTA ARM - Target Manual Versions

11.4.1 Manual Version 2.2

- Back front page, Litronic AG became SCIOPTA Systems AG.
- Chapter 2.2 The SCIOPTA ARM Delivery and chapter 2.4.1 Main Installation Window, tiny kernel added.
- Chapter 3 Getting Started, in the example folder, additional directories for boards have been introduced.
- Chapter 3 Getting Started, the Eclipse project files and the file **copy_files.bat** are now stored in the “\phyCore2294” board sub-directory of the example folder.
- Chapter 3 Getting Started, the SCIOPTA SCONF configuration file is now called **hello.xml** (was hello_phyCore2294.xml before).
- Chapter 5.8.3 Assembling with IAR Systems Embedded Workbench, added.
- Chapter 5.10.3 Compiling with IAR Systems Embedded Workbench, added.
- Chapter 5.12.3 Linking with IAR Systems Embedded Workbench, added.
- Chapter 5.13.1.1 Memory Regions, last paragraph added.
- Chapter 5.13.1.2 Module Sizes, name is now **<module_name>_size** (was <module_name>_free before).
- Chapter 5.13.3 IAR Systems Embedded Workbench Linker Script, added.
- Chapter 5.14 Data Memory Map, redesigned and now one memory map for all environments.
- Chapter 5.14.4 IAR Systems Embedded Workbench©, added.
- Chapter 6 Board Support Packages, file lists modified for SCIOPTA ARM version 1.7.2.5
- Chapter 6.3 ATMEL AT96SAM7S-EK Board, added.
- Chapter 6.4 ATMEL AT96SAM7X-EK Board, added.
- Chapter 6.5 IAR Systems STR711-SK Board, added.

11.4.2 Manual Version 2.1

- Chapter 1.1 About this Manual, SCIOPTA product list updated.
- Chapter 2.4.1 Main Installation Window, Third Party Products, new version for GNU Tool Chain (version 1.4) and MSys Build Shell (version 1.0.10).
- Chapter 2.4.7 GNU Tool Chain Installation, new GCC Installation version 1.4 including new gcc version 3.4.4, new binutils version 2.16.1 and new newlib version 1.13.1. The installer creates now two directories (and not three).
- Chapter 2.4.8 MSYS Build Shell, new version 1.0.10.
- Chapter 3, Getting Started: Equipment, new versions for GNU GCC and MSys.
- Chapter 3, Getting Started: List of copied files (after executed copy_files.bat) removed.
- Chapter 3.5.1 Description (Web Server), paragraph rewritten.
- Chapter 3.13.2.1 Equipment, serial cable connection correctly described.
- Chapter 3.13.2.2 Step-By-Step Tutorial, DRUID and DRUID server setup rewritten.
- Chapter 5.16 Integrated Development Environments, new chapter.

11.4.3 Manual Version 2.0

- Manual rewritten.
- Own manual version, moved to version 2.0

11.4.4 Manual Version 1.7.2

- Installation: all IPS Applications such as Web Server, TFTP etc. in one product.
- Getting started now for all products.
- Chapter 4, Configuration now moved into Kernel User's Guide.
- New BSP added: Phytec phyCORE-LPC2294.
- Uninstallation now separately for every SCIOPTA product.
- Eclipse included in the SCIOPTA delivery.
- New process SCP_proxy introduced in Getting Started - DHCP Client Example.
- IPS libraries now in three verisons (standard, small and full).

11.4.5 Manual Version 1.7.0

- All **union sc_msg *** changed to **sc_msg_t** to support SCIOPTA 16 Bit systems (NEAR pointer).
- All **union sc_msg **** changed to **sc_msgptr_t** to support SCIOPTA 16 Bit systems (NEAR pointer).
- All **sdd_obj_t *** changed to **sdd_obj_t NEARPTR** to support SCIOPTA 16 Bit systems.
- All **sdd_netbuf_t *** changed to **sdd_netbuf_t NEARPTR** to support SCIOPTA 16 Bit systems.
- All **sdd_objInfo_t *** changed to **sdd_objInfo_t NEARPTR** to support SCIOPTA 16 Bit systems.
- All **ips_dev_t *** changed to **ips_dev_t NEARPTR** to support SCIOPTA 16 Bit systems.
- All **ipv4_arp_t *** changed to **ipv4_arp_t NEARPTR** to support SCIOPTA 16 Bit systems.
- All **ipv4_route_t *** changed to **ipv4_route_t NEARPTR** to support SCIOPTA 16 Bit systems.
- IAR support added in the kernel.
- Web server modifiied.
- TFTP server added (in addition to client).
- DHCP server added (in addition to client).
- DRUID System Level Debugger added.

12 Index