EstiNet Protocol Module 开发手册

Release Date: May 2, 2018

Produced and maintained by EstiNet Technologies Inc.

阅读本手册建议：

已经有使用过 ESTINET 的 GUI 操作并知道如何执行模拟
已经取得 SOURCE CODE
对 C++不陌生
对网络概论有基本的认识

目标读者：

- 想要在 ESTINET 上开发自己的模块
- 想要修改 ESTINET 上既有的模块

本手册内容：

本手册共分为 9 章，前两章着重运用 GUI 的一些开发组件，让开发者在开发自己的模块时更为便利；后面七章则是着重使用内部模块中常用的 API，以及常用的数据结构。

目 录

第一章：MDF 与 Protocol Stack

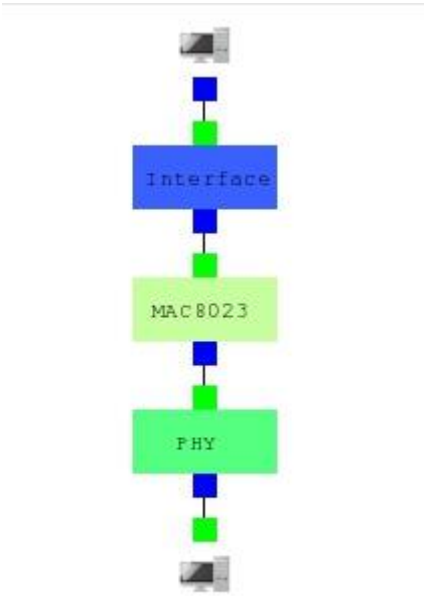
本章重點：
(1)、什麼是 MDF？什麼是 Protocol Stack？
(2)、怎麼改 MDF 及 Protocol Stack？
(3)、模擬引擎(estinetse)中的 API--VBIND、get_nid()
(4)、怎麼編繹 Source Coce？

下载练习：

Chapter1.tar.bz2

在 EstiNet 的仿真引擎中，提供了一个模拟的平台，内含有许多模块(Module)，每个模块都有不同的机制，像模拟有线 IEEE 802.3 的模块，无线的 IEEE 802.11 系列的模块等等。

许多模块所串接起来是一个网络设备的 Protocol Stack，下图所示为 Host1 的 Protocol Stack。(注：每种网络设备的 Protocol Stack 不尽相同)



每个模块中都有一些设定值开放给使用者设定，因此需要一个接口以及描述模块的档案，提供给 GUI 接口开放给用户设定，这个档案称为 MDF。

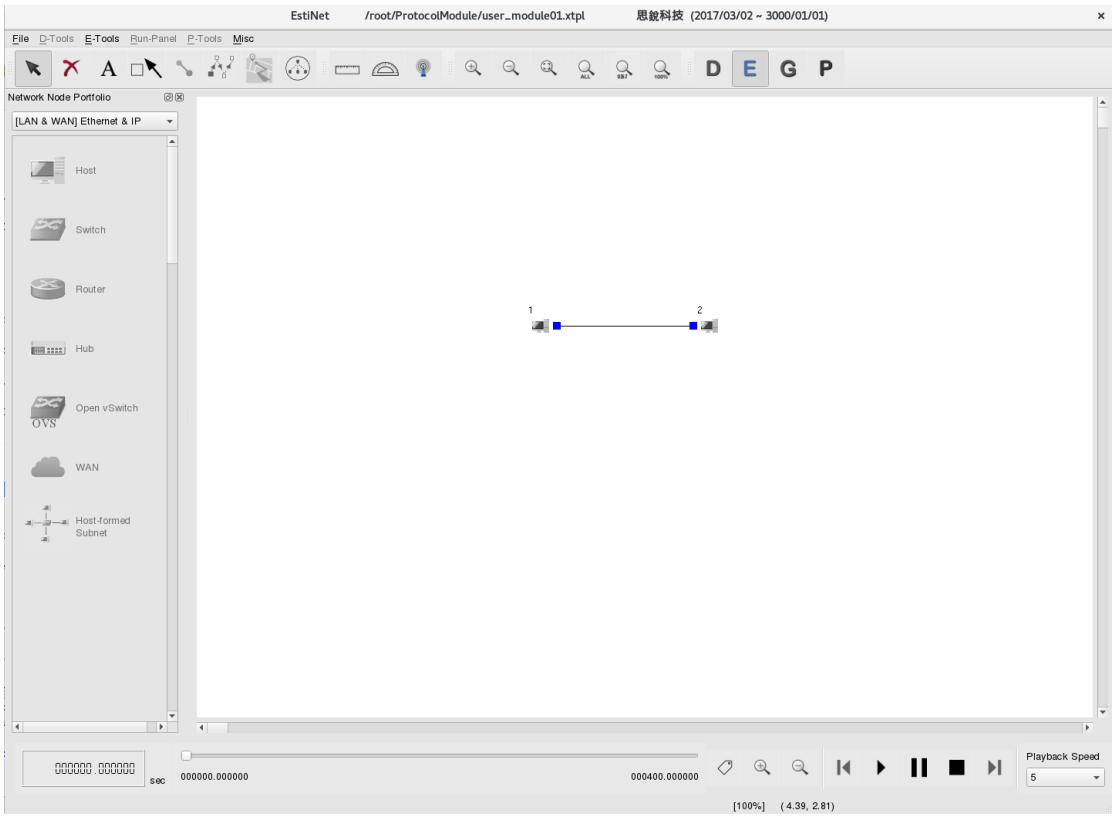
MDF 的全名为 Module Description File，其目的是为了将模块内的参数可以透过 GUI layout 的画面来设定，使用者在 layout 设定完这些参数后，经由 GUI 程序切换到 G Mode 时，就会写入 sim/interface_and_medium_setting/general/目录中的 if_and_medium_conf 配置文件中。

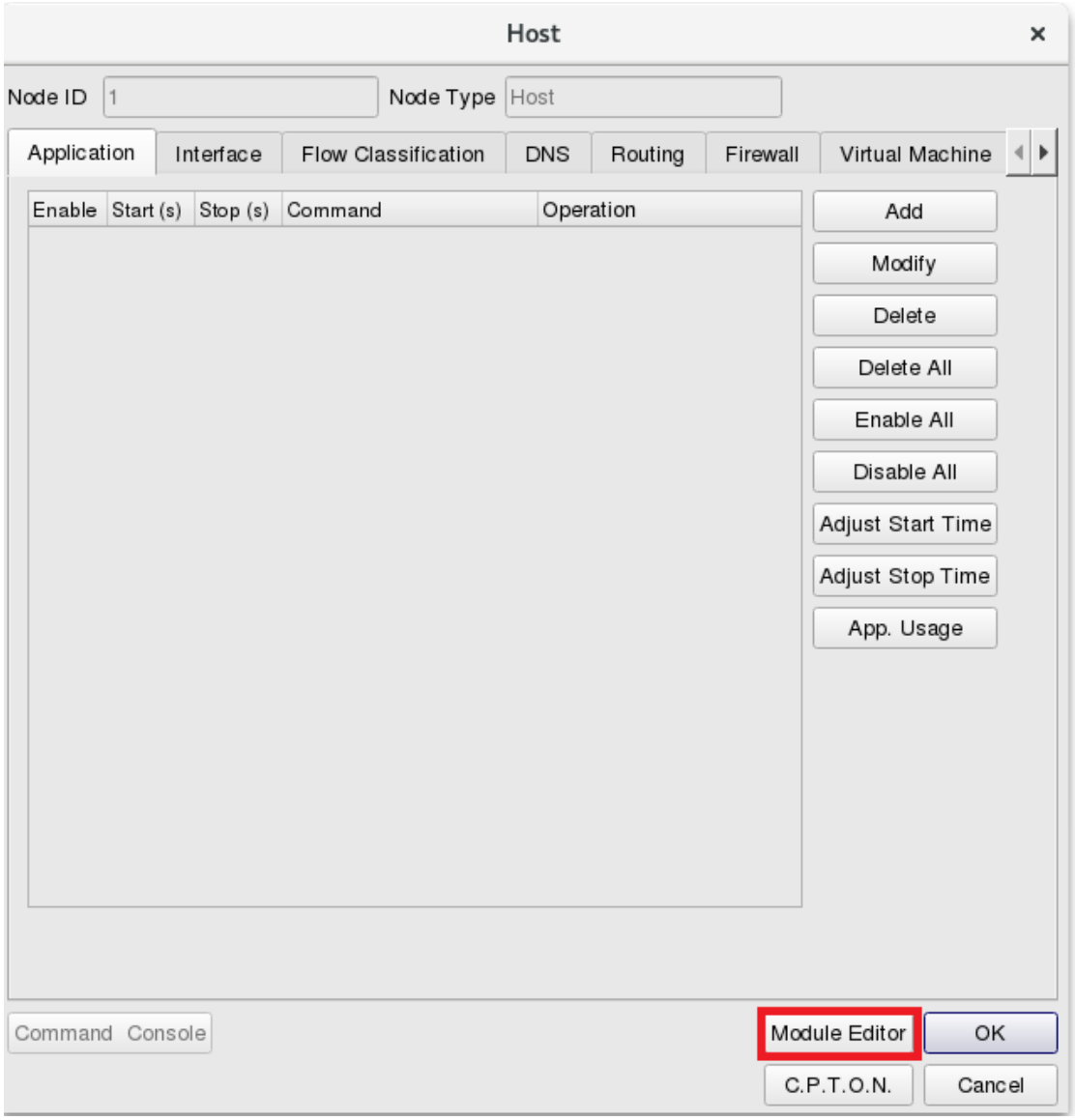
因此 MDF 是对开发模块的人，是一个相当重要的工具。MDF 可以设定自己的 Layout 与参数来便利使用者与开发人员，Layout 常用的对象如 RADIOBOX、TEXTLINE、CHECKBOX、LABLE 等等。

练习 1-1：修改 MDF (搭配 GUI)、修改 Protocol Stack

- 建立范例拓扑

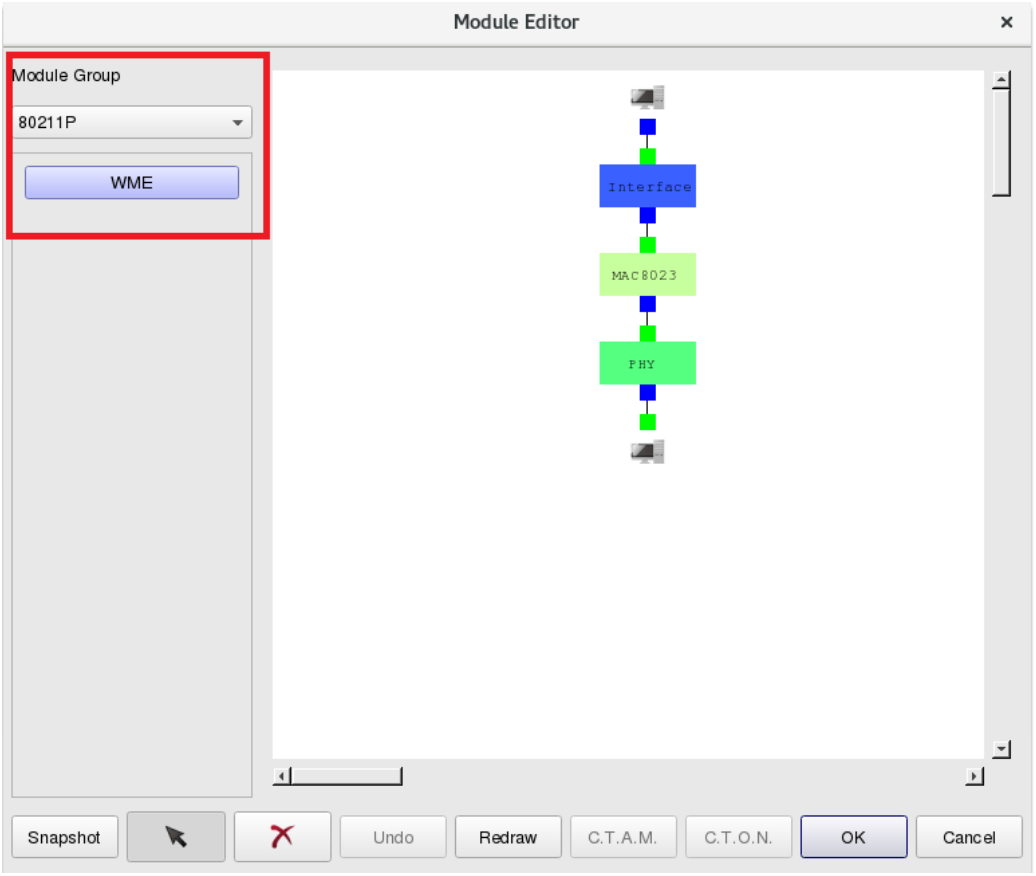
  在 GUI 的 D Mode 建立如下图的拓扑，是 Host1 连接 Host2 的拓扑，接着转换到 E Mode 时存档为 user_moduler01。



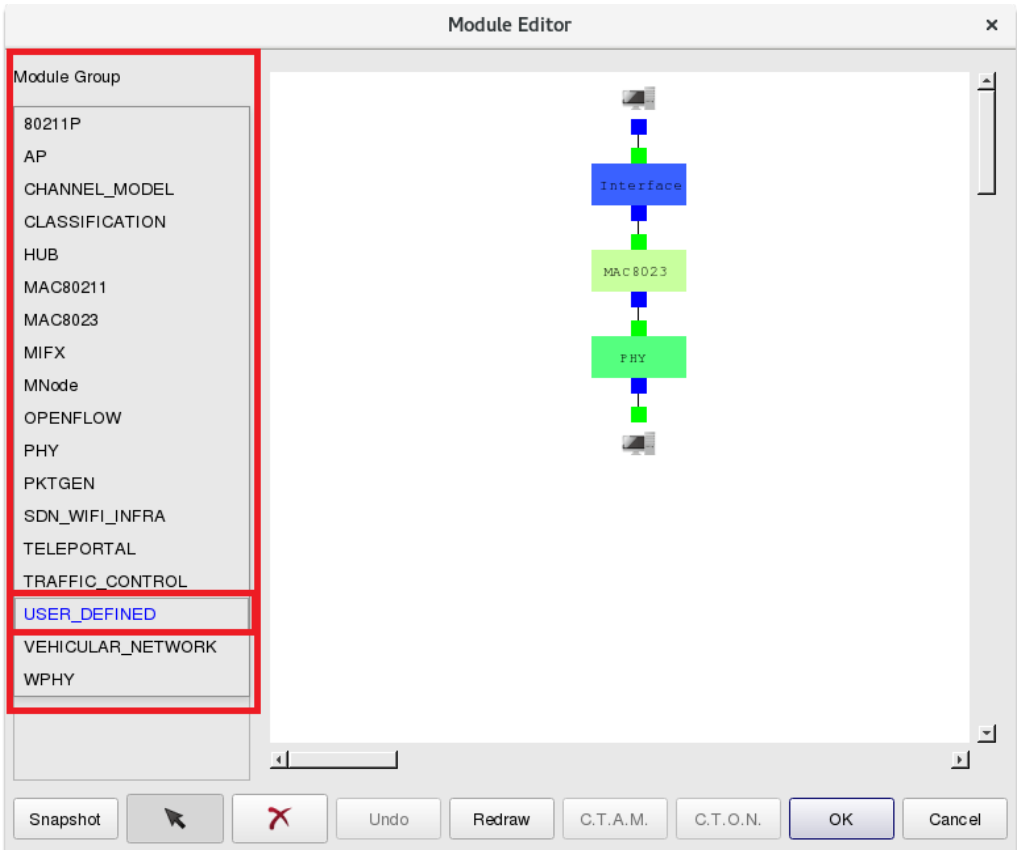


在 E Mode 点选两下 Host1，再点选"Module Editor"按钮，可以看到这个节点中的 Protocol Stack。

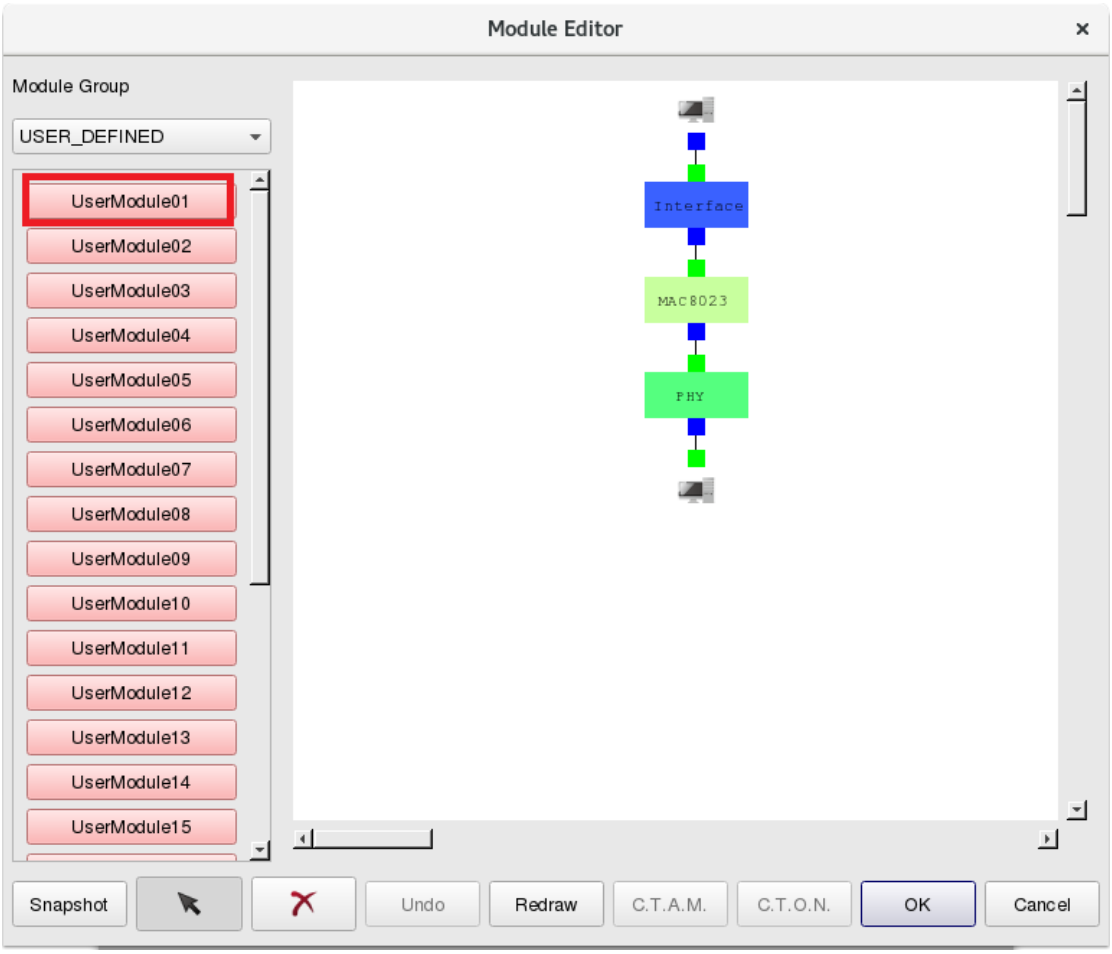
首先加入一个用户可以自定义的模块，准备在这个模块中加入用户的程序代码。所以要将这个模块加入 Protocol Stack 中。首先看到左侧有一个下拉式选单"Module Group"，GUI 负责管理跟分类这些模块，如第一个放的是 80211P 的选项中，里面就有 WME 的模块。



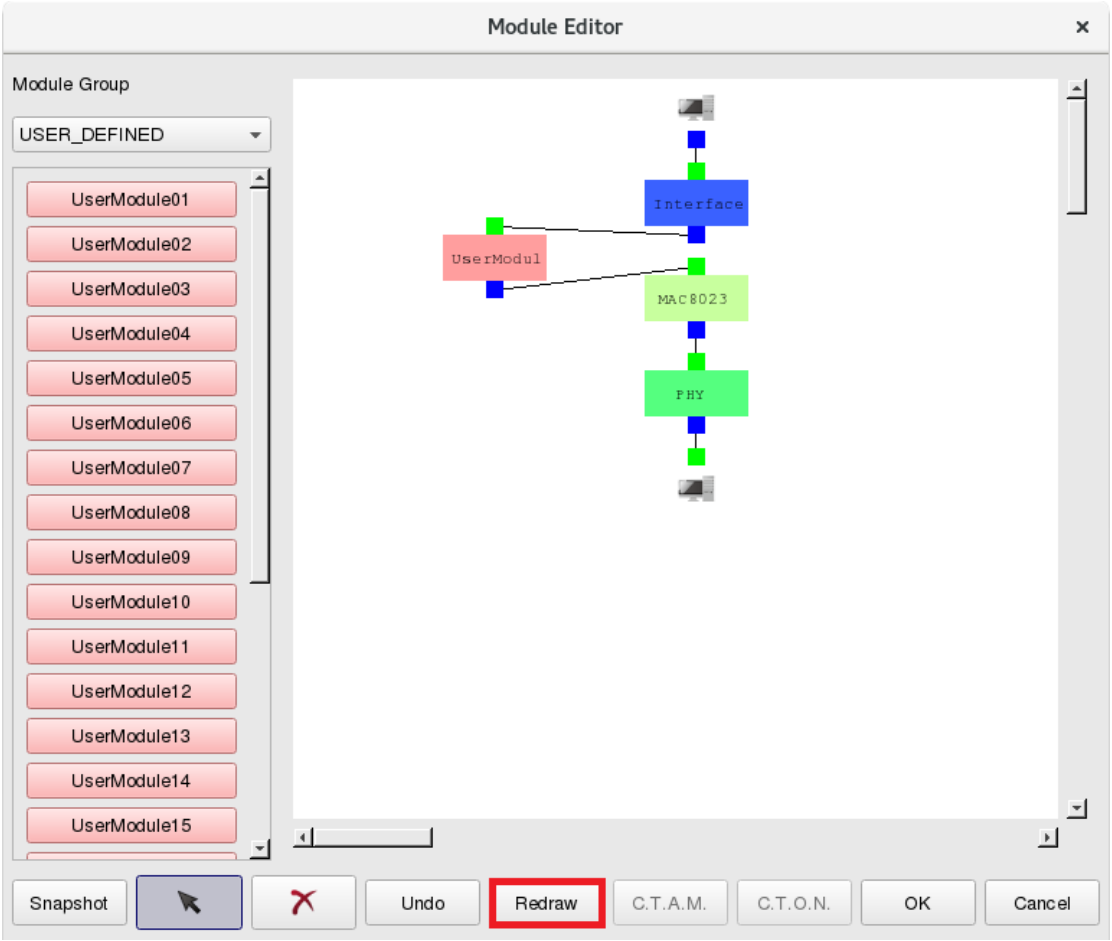
请选择"USER_DEFINED"模块 GROUP。



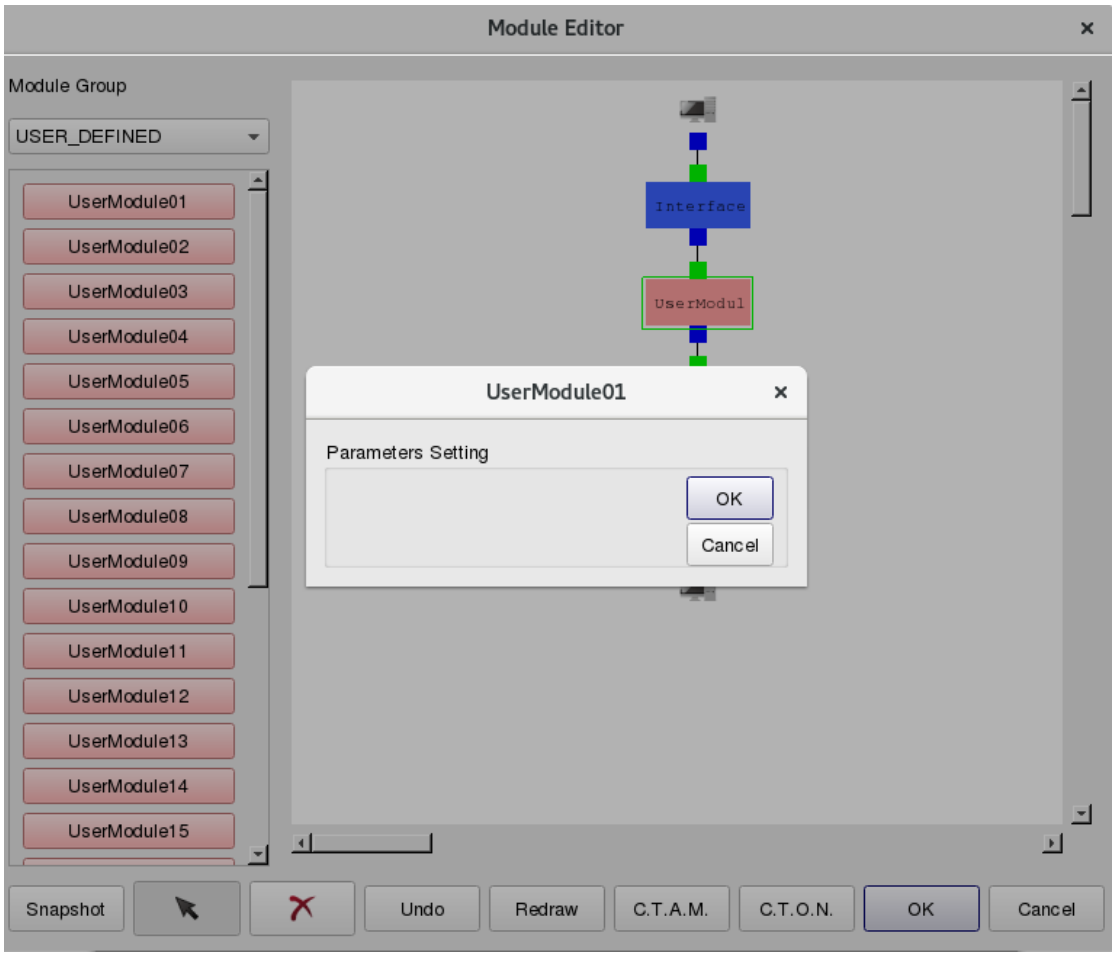
选择"USER_DEFINED"模块 GROUP 之后，可以看到如下图共有 25 个 UserModule 可以让使用者自行定义，这 25 个模块是免费提供给用户使用。因此点选第一个免费提供的"UserModule01"模块。



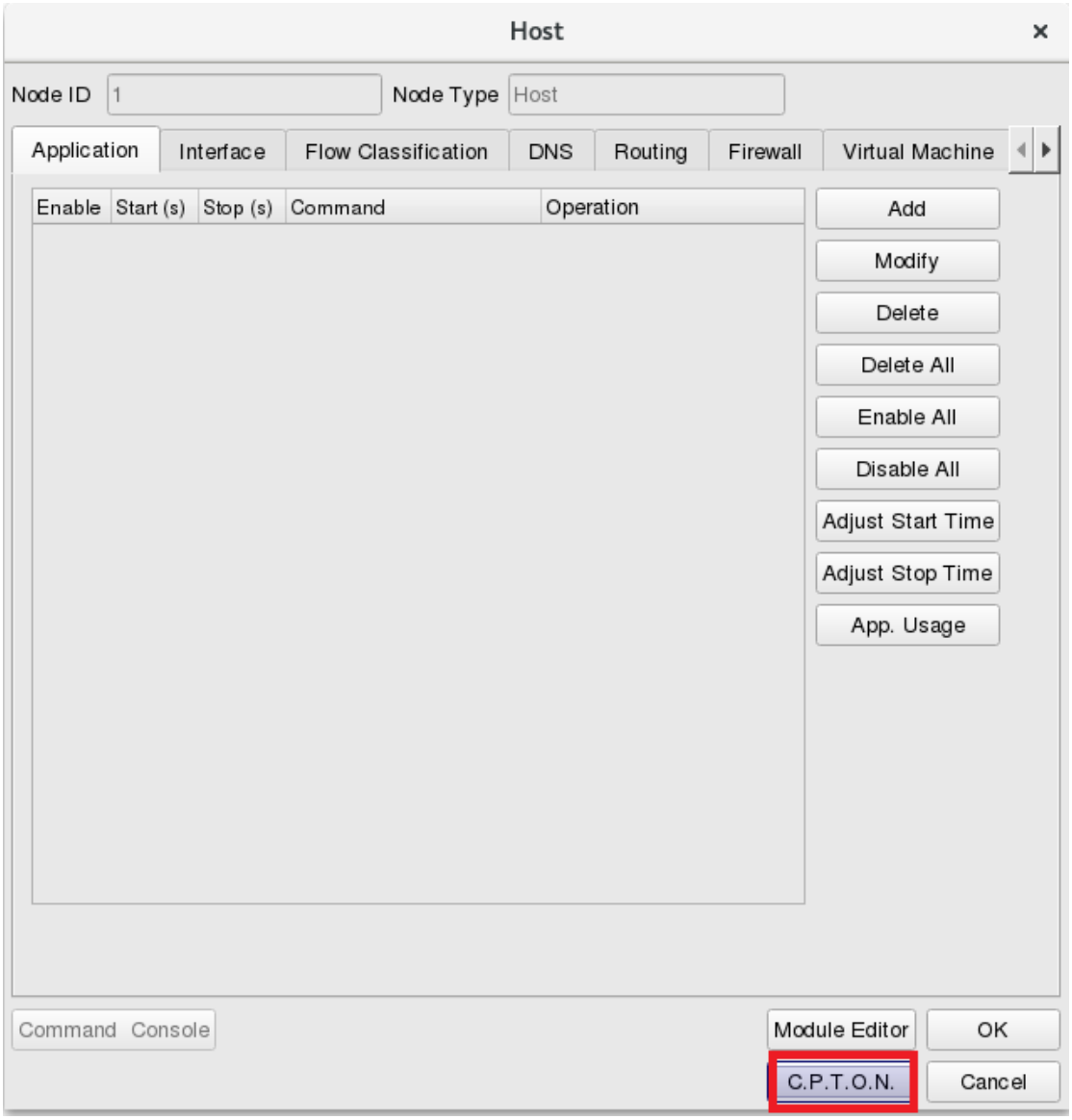
点选后，直接放到白色的画布上，将"UserModule01"模块放入到 Interface 跟 MAC8023 之间，用下面按钮"X"，将 Interface 与 MAC8023 之间的联机删除，再用箭号的按钮，将 UserModule01 的上面绿色的接头接到 Interface 下面的蓝色接头；UserModule01 的下面蓝色的接头接到 MAC8023 上面的绿色接头，就完成将 UserModule01 加入到 HOST1 中的 Protocol Stack 了！(画完后，可以按下"ReDraw"按钮重新整理 Protocol Stack)



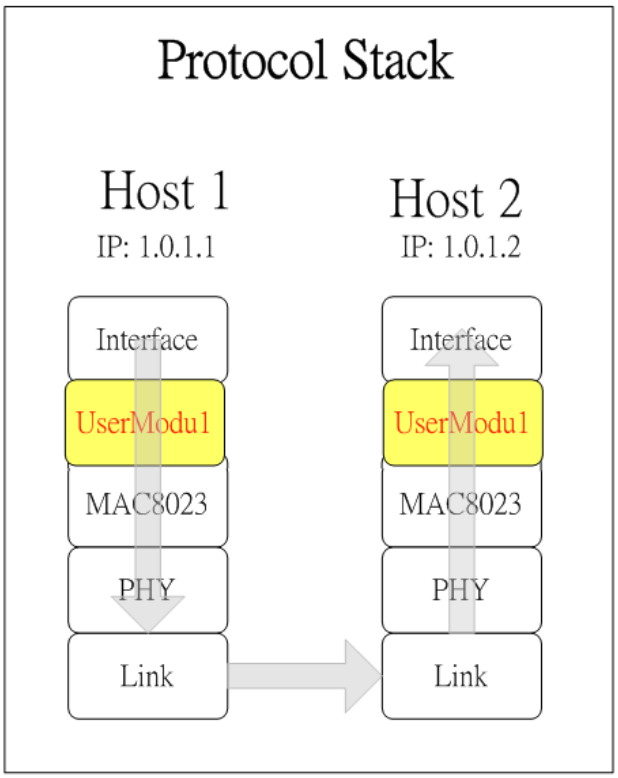
点选两下这个 UserModule01 模块，会发现这个模块目前是空的，稍后会说明如何修改 MDF 来加入 Layout 对象(如 TEXTLINE、CHECKBOX 等)来方便使用者/开发者填入参数。



同样地 Host2 也要是一模一样的 Protocol Stack，因此按下"OK"按钮后回到 Host1 的设定画面，接着按下"C.P.T.O.N."按钮(Copy the Node's Protocol Stack to Other Nodes with the Same Type)，将 Host1 的 Protocol Stack 复制到其它的 Host 上(在此拓扑即为 Host2)。



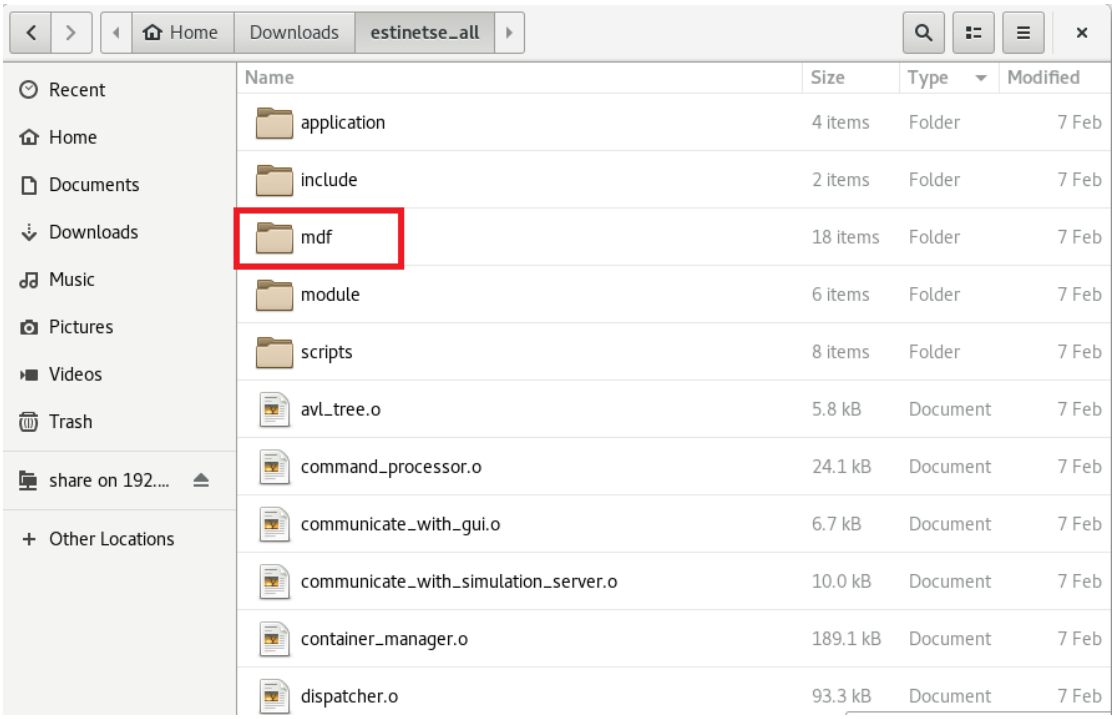
拓扑即设置完成。本手册后面的拓扑也会由此架构延伸，整个的 Protocol Stack 如下图所示：



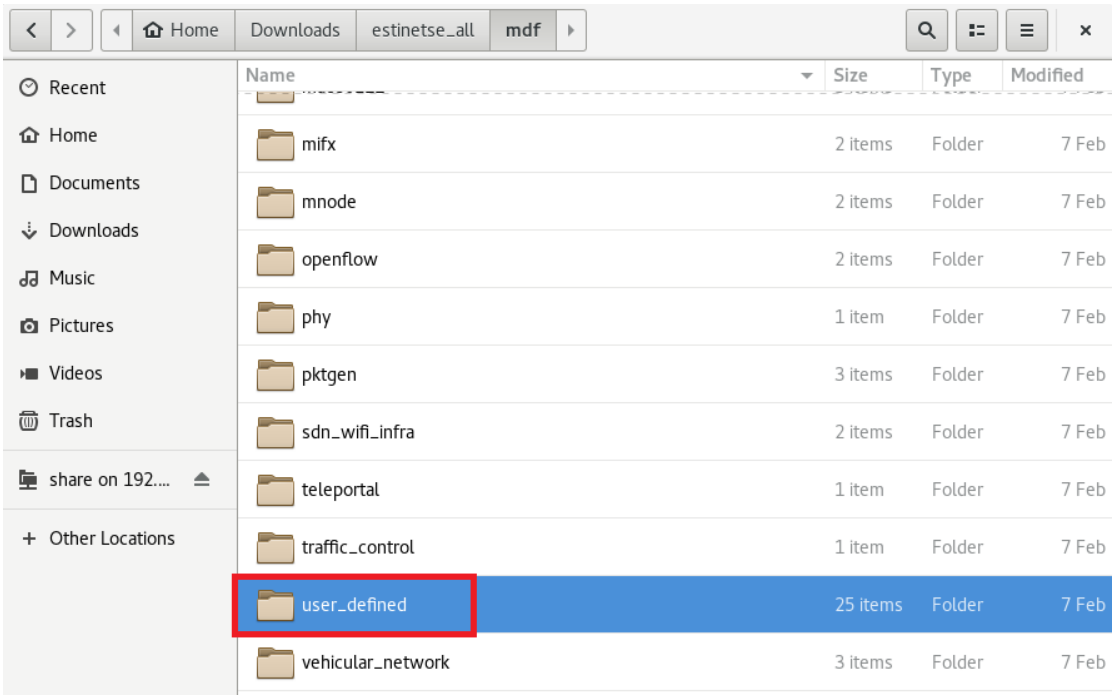
- 介绍 MDF 的结构

  接着要开始来编写 UserModule01 这个模块的 MDF 文件，希望可以加入 TEXTLINE 来填入两个参数，一个是"My Number"；另一个是"My String"。

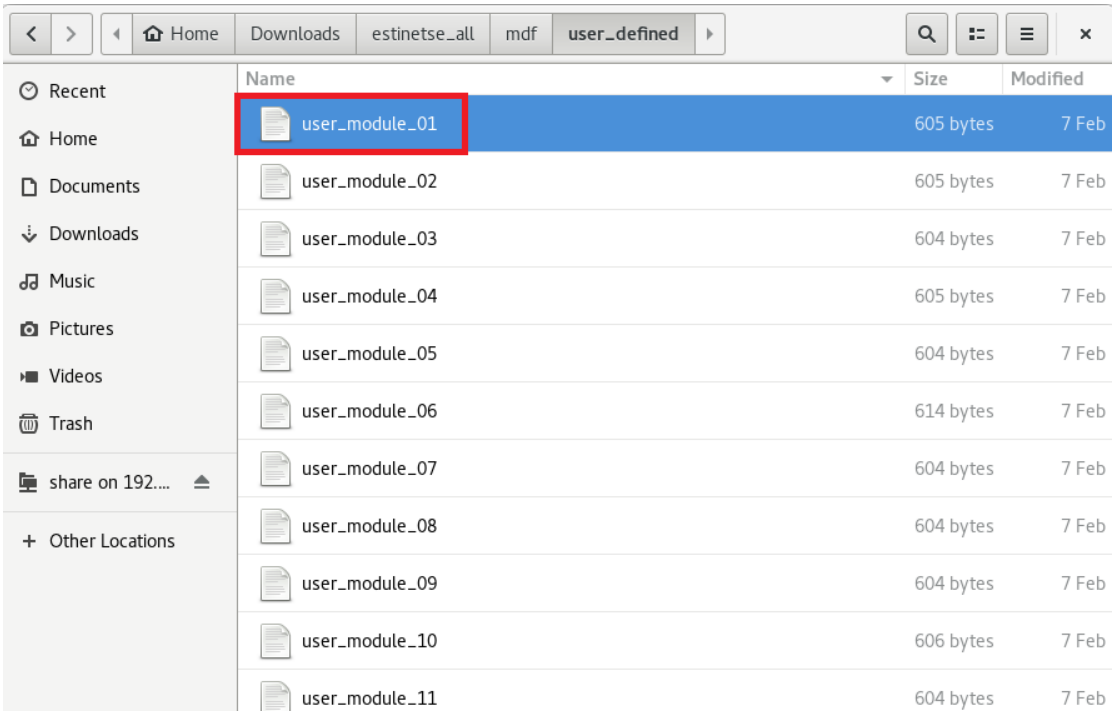
  因此需要修改 MDF 档案，首先开启原始文件目录中(因用户放置的路径都不同，请使用者查看自己所放置的原始码位置)，有一个 mdf 的目录，如下图所示。



进入 mdf 目录后可以看到一个 UserDefined 的目录，如下图所示：



UserDefined 目录里面共有 25 个 UserModule 的 MDF 档案，因拓扑中使用的是 UserModule01 的模块，因此开启此档案来修改。



开启 MDF 后如下图所示，与一般的程序语言的文法并不相同，这是因为 MDF 是专门给 GUI 读的档案。GUI 读取的时机在于开启 Module editor 按钮时，就会将所有的 MDF 按照其 Group Nmae(MDF 档案中的其中一个字段)进行分类摆在左侧；另外当某个模块被点开的时候，GUI 便按照此模块的 MDF 档案的语法执行命令。

MDF 的基本架构：

MDF 档的首尾会被"**ModuleSection**”以及“**EndModuleSection**”所包括。中间主要有三大区块，分别为"**HeaderSection**"、"**InitVariableSection**"、"**ExportSection**"。这三个区块结尾时的关键词分别为"**EndHeaderSection**"、"**EndInitVariableSection**"、"**EndExportSection**"。

HeaderSection 区块是定义模块的名称、以及模块所属的 Group(GUI 分类用)，以及网络的类型、参数的宣告等等。

InitVariableSection 区块是定义 GUI 的对象的呈现，例如按钮、TEXTLINE、RADIOBOX、CHECKBOX、GROUP 对象等等…。

ExportSection 区块则是定义用来当模拟运行时，GUI 可以透过两种所定义的对象跟仿真引擎透过 IPC(inter-process communication)的方式拿取数据或设定数据。在下一章会介绍这个区块的使用方式。

user_module_01

```
ModuleSection
    HeaderSection
        ModuleName      UserModule01

        GroupName       USER_DEFINED
        Introduction        "Empty module for development"

    EndHeaderSection

    InitVariableSection
        Caption         "Parameters Setting"
        FrameSize       320 90

        Begin BUTTON        b_ok
            Caption     "OK"
            Scale       250 17 60 30
            ActiveOn    ALL_MODE
            Action      ok
            Comment     "OK Button"
        End

        Begin BUTTON        b_cancel
            Caption     "Cancel"
            Scale       250 49 60 30
            ActiveOn    ALL_MODE
            Action      cancel
            Comment     "Cancel Button"
        End
    EndInitVariableSection

    ExportSection
        Caption         ""
        FrameSize       0 0
    EndExportSection
EndModuleSection
```

范例中希望新增两个参数，因此首先先在 **HeaderSection** 中宣告两个参数，一个是"MyNumber"；另一个是"MyString"在 HeaderSection 中，如下图红字所示：

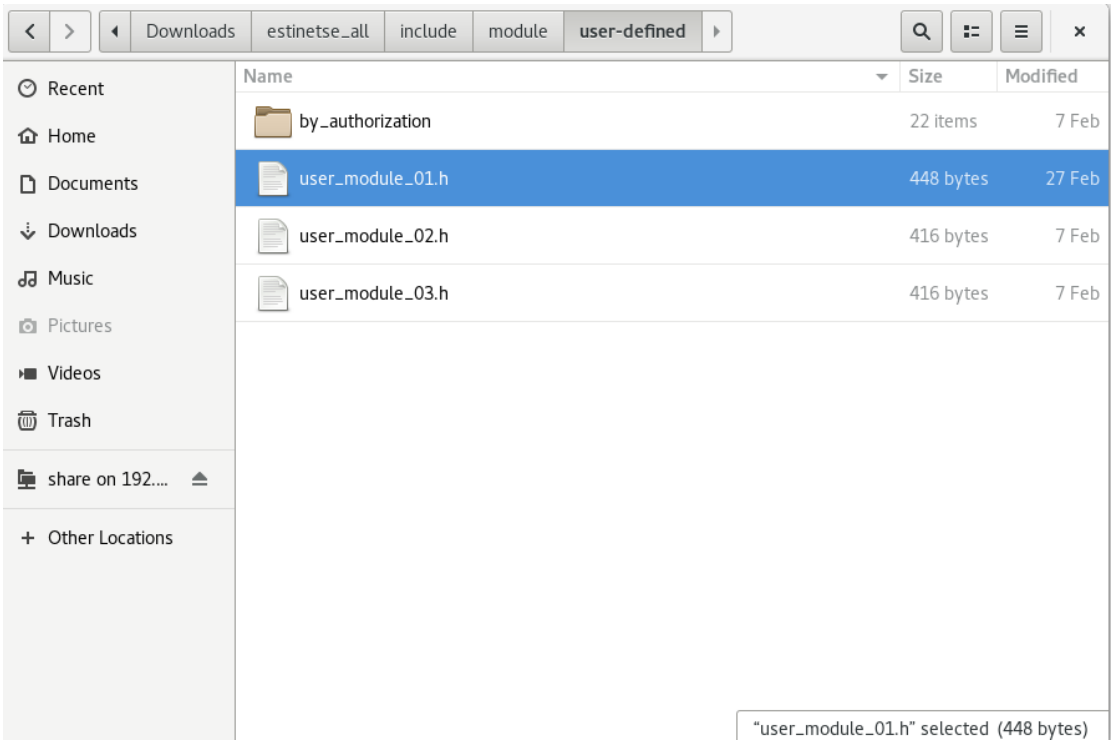
```
ModuleSection
    HeaderSection
        ModuleName          UserModule01
        GroupName           User_Defined
        Introduction        "This is a user-defined module."
        Parameter           myNumber 300 local
        Parameter           myString 111.111.111.111 local
    EndHeaderSection


    InitVariableSection
```

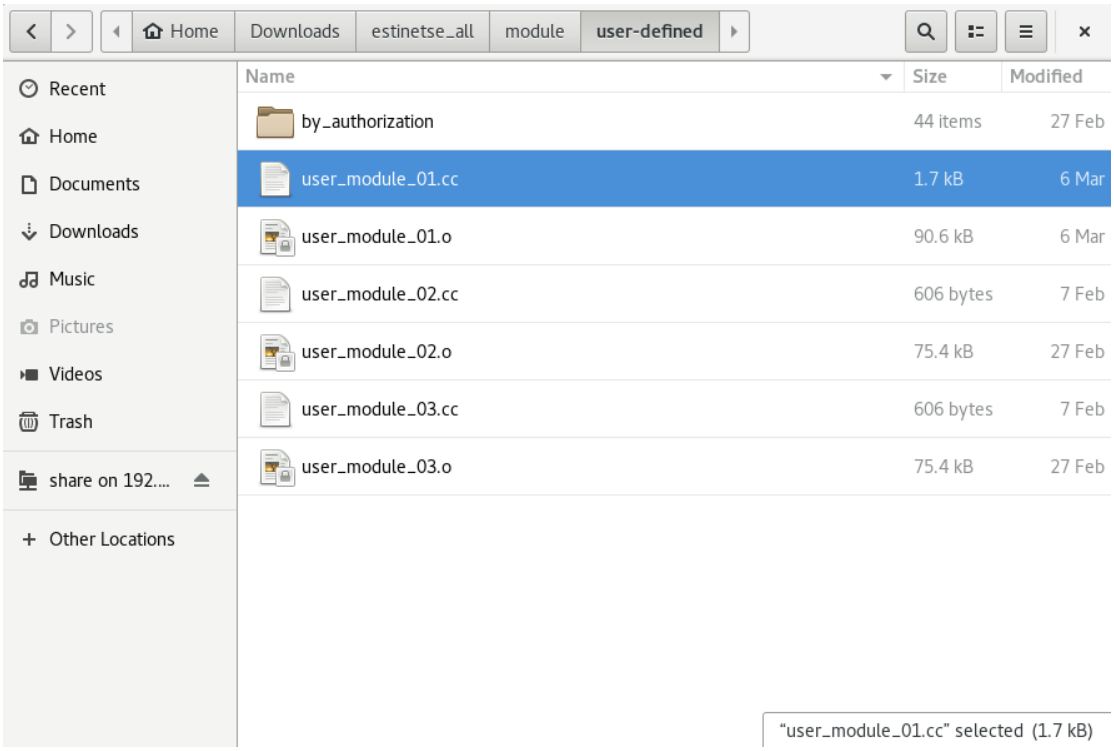
```
            Caption             "Parameters Setting"
            FrameSize           320 90

            Begin BUTTON        b_ok
                Caption         "OK"
                Scale           250 17 60 30
                ActiveOn        ALL_MODE
                Action          ok
                Comment         "OK Button"
            End

            Begin BUTTON        b_cancel
                Caption         "Cancel"
                Scale           250 49 60 30
                ActiveOn        ALL_MODE
                Action          cancel
                Comment         "Cancel Button"
            End
        EndInitVariableSection

        ExportSection
            Caption             ""
            FrameSize       0 0
        EndExportSection
EndModuleSection
```

新增参数一开始的标识符是 Parameter, 接着空白后接参数名称，接着再输入默认值，最后一个标识符是 local，最后一个标识符是 GUI 专用，可以参考附录 A 细节。

完成之后，就完成 MDF 档中新增参数的设定。

接着到仿真引擎中的 USER MODULE 01 模块来设定读取 MDF 中的设定值。每个模块中都分别有 head 檔跟 cc 檔，user_module_01.h 档案位置是在原始码的目录下的 include/module/user-defined/user_module_01.h；另外 user_module_01.cc 档案位置是在原始码的目录下的 module/user-defined/user_module_01.cc
如下两张图所示：





在 user_module_01.h 文件中宣告两个变量准备将 MDF 的两个参数透过 if_and_medium_conf 檔读入进来，宣告一个整数 Number、一个字符串指针 String。如下红字所示：

```
#ifndef __user_module_01_h__
#define __user_module_01_h__

#include <event.h>
#include <object.h>

class UserModule01 : public NslObject {

 private:
      int               Number;
      char              *String;

 public:

      UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name);
      ~UserModule01();

      int          init();
      int          command(int argc, const char *argv[]);
      int          recv(ePacket_ *pkt);
      int          send(ePacket_ *pkt);

};

#endif   /* __user_module_01_h__ */
```

而在 user_module_01.cc 檔的建构子中，透过 vBind_int 和 vBind_char_str 这两个 EstiNet 所提供的 API，将 MDF 的变量值与模块中的变量衔接起来，第一个参数放 MDF 中定义的参数名称，第二个则是在模块中来承接 MDF 参数的 C++变量。用法如下红字所示。而在 init 函数中，将这两个变量印出来(打印加入 ANSI escape codes 控制文字颜色输出为粗体红色，背景黑色)

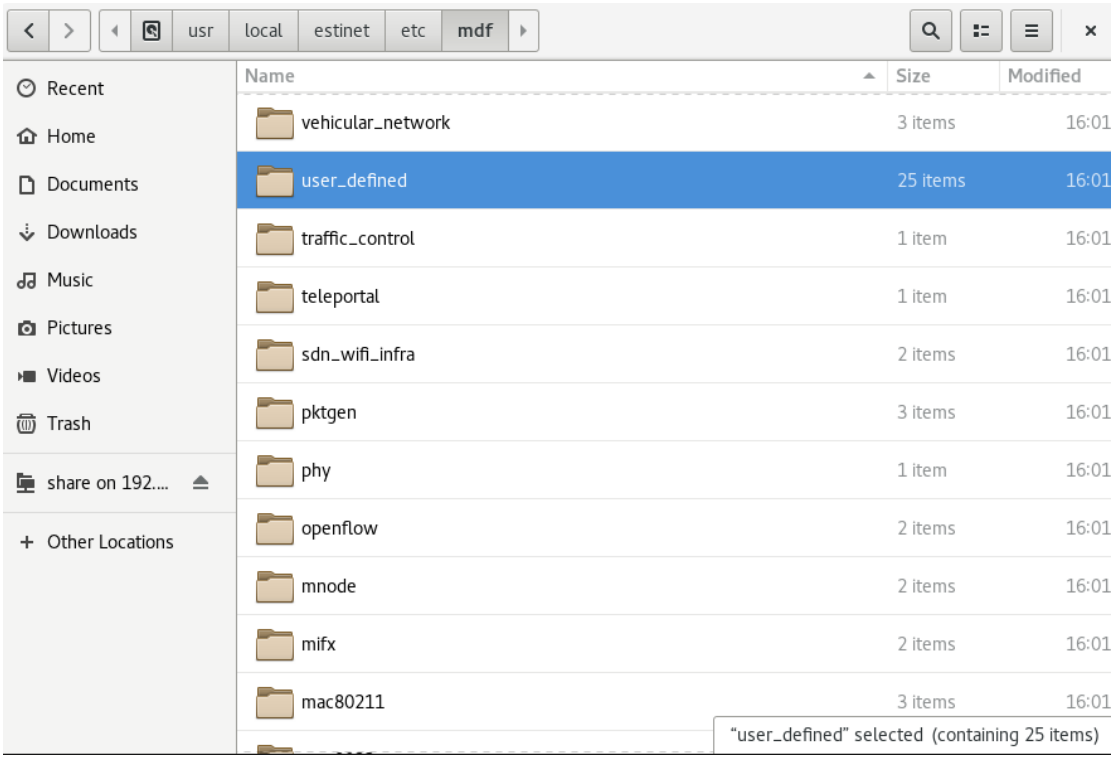
```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>

MODULE_GENERATOR(UserModule01);
```

```
UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char
*name): NslObject(type, id, pl, name) {
        vBind_int("myNumber", &Number);
        vBind_char_str("myString", &String);
}

UserModule01::~UserModule01(){}

int UserModule01::init() {


/* exercise 1 */
  printf("\e[1;31;40m\nExercise 1: myNumber = %d, myString = %s\e[m\n",
     Number, String);

        return(NslObject::init());
}

int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        return(NslObject::send(pkt));
}
```

接着开启终端机到原始码的目录下，执行 **make** 来编绎所有的原始码。

再执行 **make install**，会将仿真引擎(estinetse)的 binary 檔以及刚刚设定后的 mdf 檔，搬到 GUI 默认指定读入的路径(bin 文件会放到

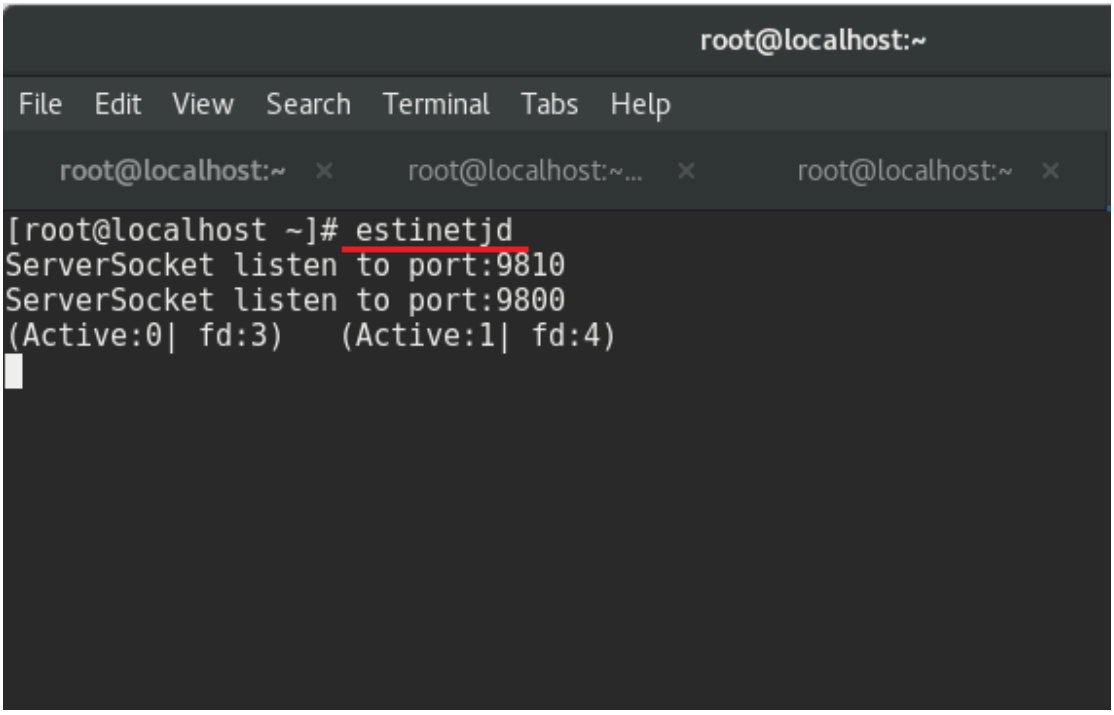
/usr/local/estinet/bin ， mdf 档则是会放到/usr/local/estinet/etc/mdf)，
如下图所示：



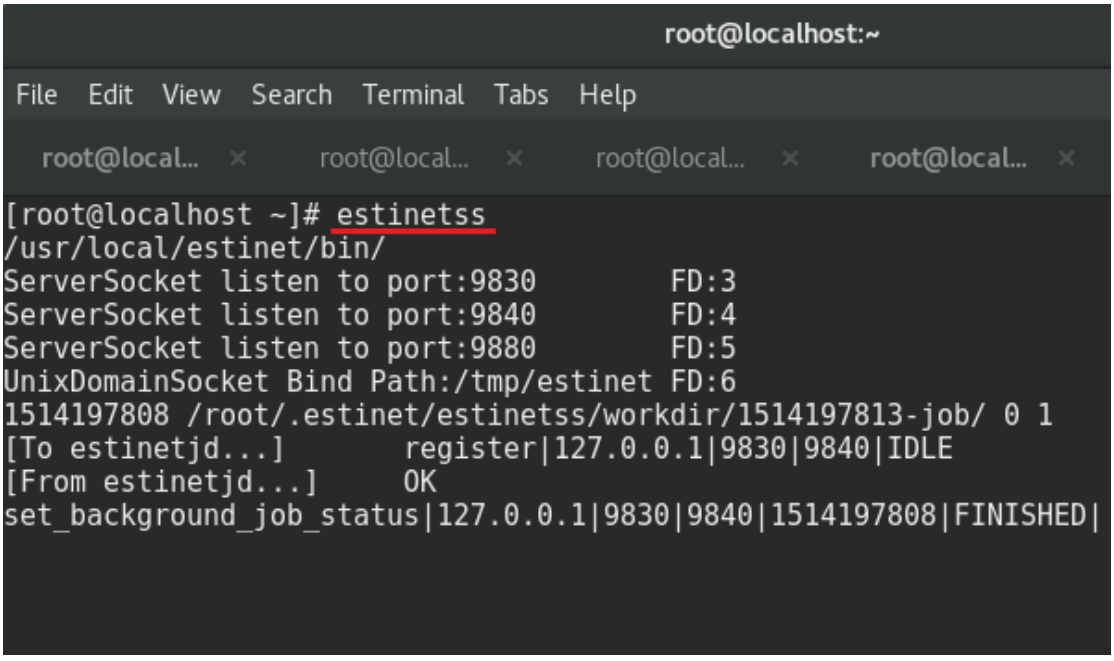
注意：在执行 make install 前，要先关闭 GUI，因为加入 MDF 参数这个动作，是在 GUI 一开始启动时，就会将所有 MDF 的参数给读入，如果中途加入参数，则必须关闭 GUI 再重新启动。如果只是设定 Layout 的宽度、高度或是加入对象等，则可以不用重开 GUI，GUI 只要重新点入模块，就可以看到宽度、高度改变的效果。

完成上述步骤后，编译完成，并且 install 完成。

接着开启三个终端机：
第一个终端机执行 estinetjd，如下图所示：



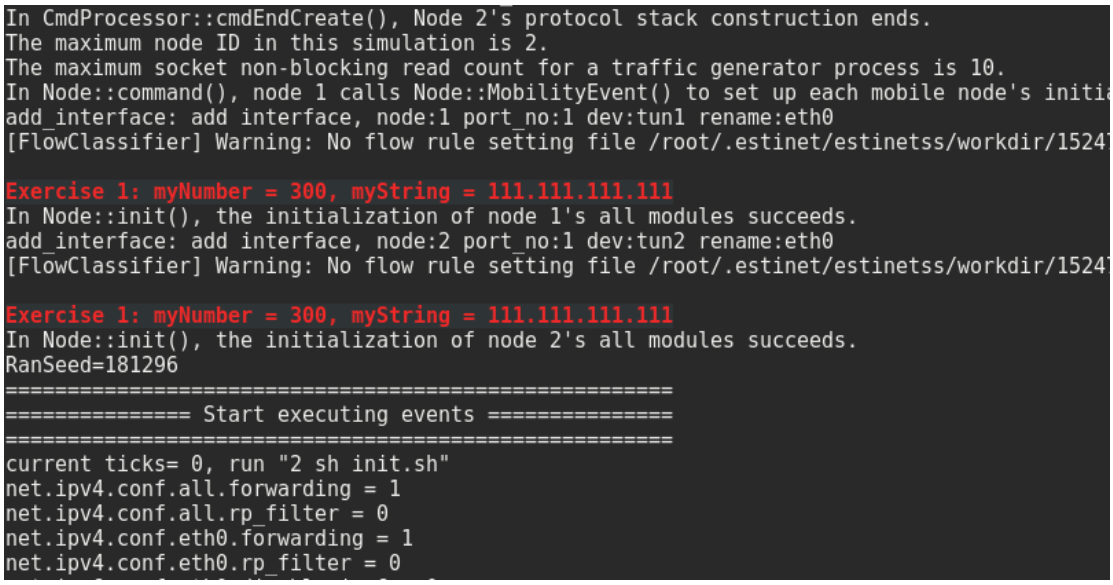
第二个终端机执行指令 **estinetss**



第三个终端机执行 GUI 指令”**estinetgui**”：
首先将刚刚设定 Protocol Stack 的 GUI 存档后关闭。
重新再开启 GUI，在终端机输入 estinetgui，接着在开启此拓扑 (user_moduler01.xtpl)，并切换到 **G** 模式执行仿真。

注意 estinetss 的终端机窗口中的结果。

因为有两个节点，且都有放入 User module01 的模块，该模块的 init 函数会打印参数值，因此画面上有两次打印结果，如下图所示。



- 修改 Layout

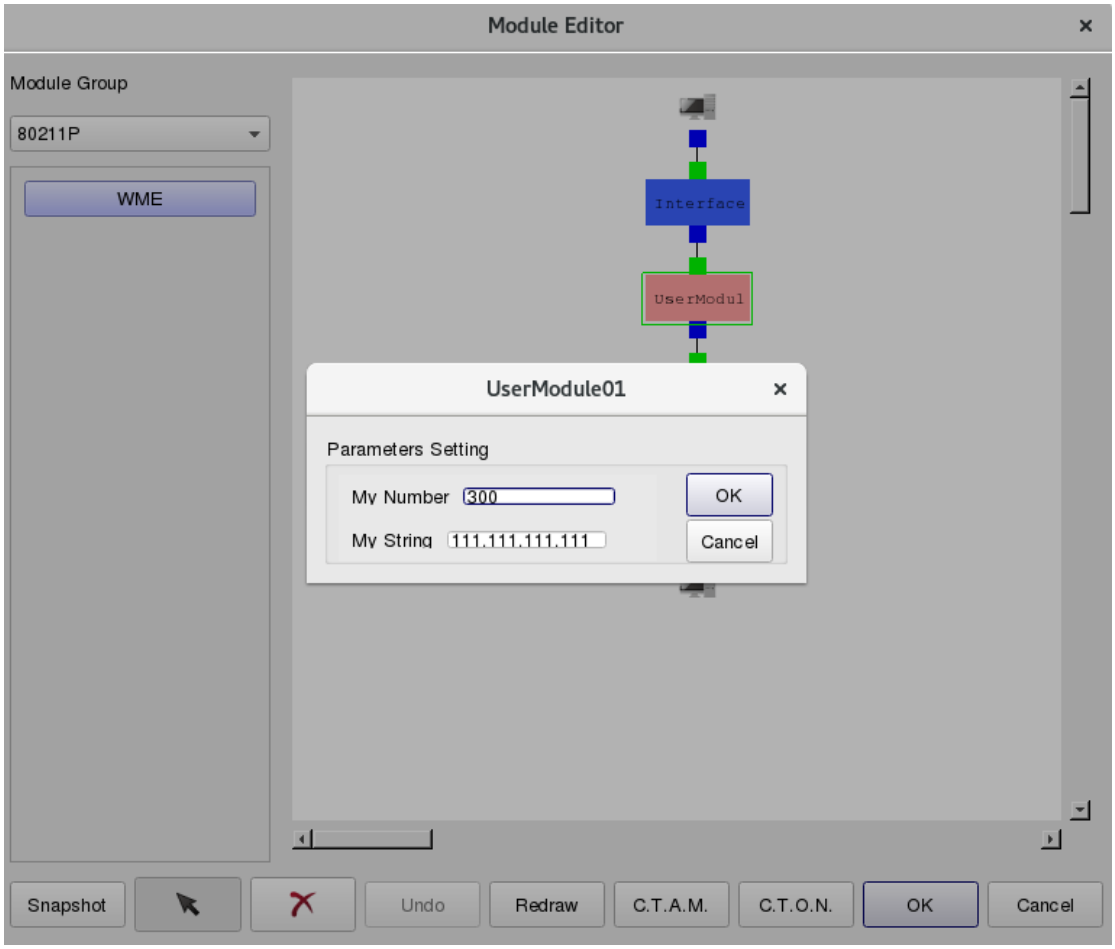
  上面的 estinetss 显示的结果是 MDF 参数的默认值。如果要透过 GUI 对象输入数值的话，就需要在 MDF 中使用 TEXTLINE 跟 LABEL 等对象。

```
ModuleSection
    HeaderSection
        ModuleName          UserModule01
        GroupName           User_Defined
        Introduction        "This is a user-defined module."
        Parameter           myNumber 300 local
        Parameter           myString 111.111.111.111 local
    EndHeaderSection

    InitVariableSection
        Caption             "Parameters Setting"
        FrameSize           320 90

        Begin TEXTLINE      myNumber
            Caption         "My Number "
            Scale           10 18 220 30
            ActiveOn        MODE_EDIT
```

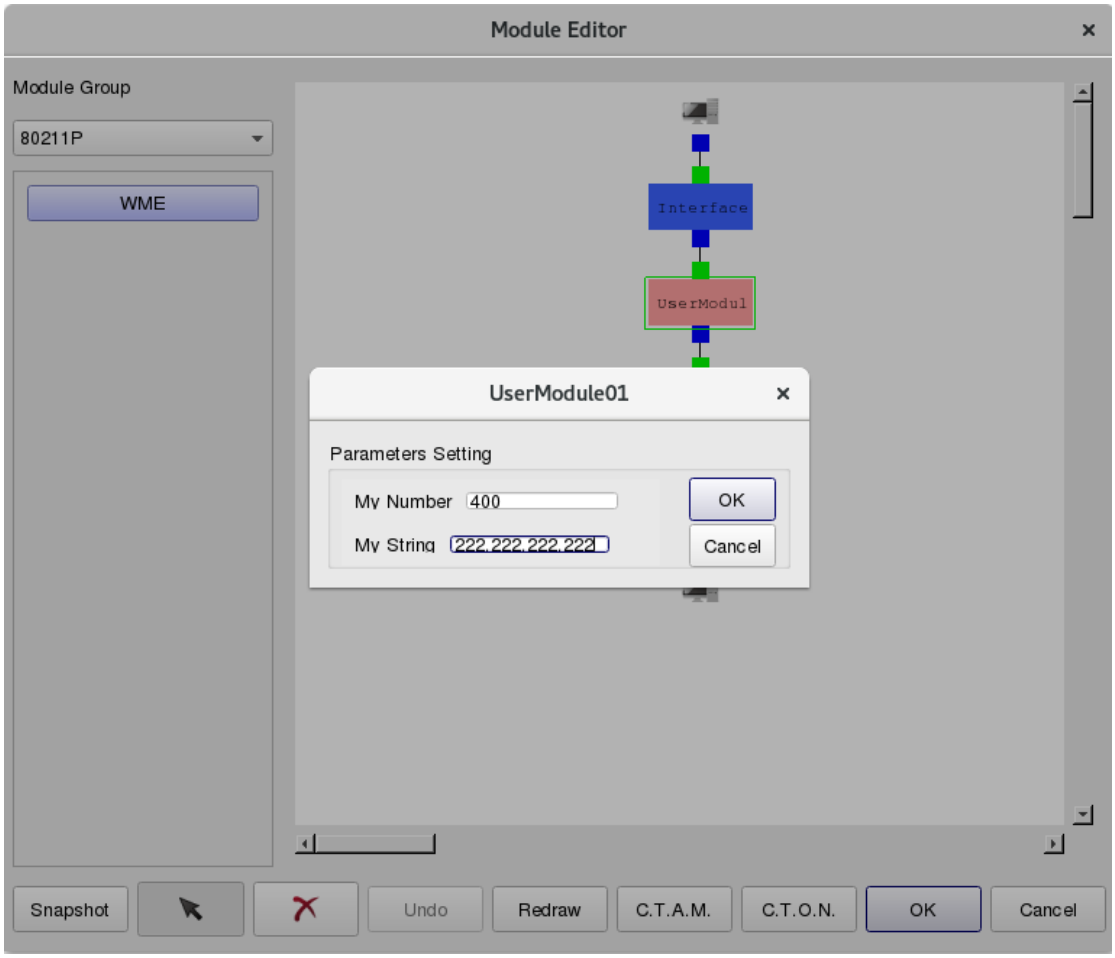
```
                Enabled         TRUE
                Type            INT
                Comment         "An Integer"
            End

            Begin TEXTLINE      myString
                Caption         "My String "
                Scale           10 48 220 30
                ActiveOn        MODE_EDIT
                Enabled         TRUE
                Type            IP
                Comment         "An IP string"
            End

            Begin BUTTON        b_ok
                Caption         "OK"
                Scale           250 17 60 30
                ActiveOn        ALL_MODE
                Action          ok
                Comment         "OK Button"
            End

            Begin BUTTON        b_cancel
                Caption         "Cancel"
                Scale           250 49 60 30
                ActiveOn        ALL_MODE
                Action          cancel
                Comment         "Cancel Button"
            End
        EndInitVariableSection

        ExportSection
            Caption         ""
            FrameSize       0 0
        EndExportSection
EndModuleSection
```

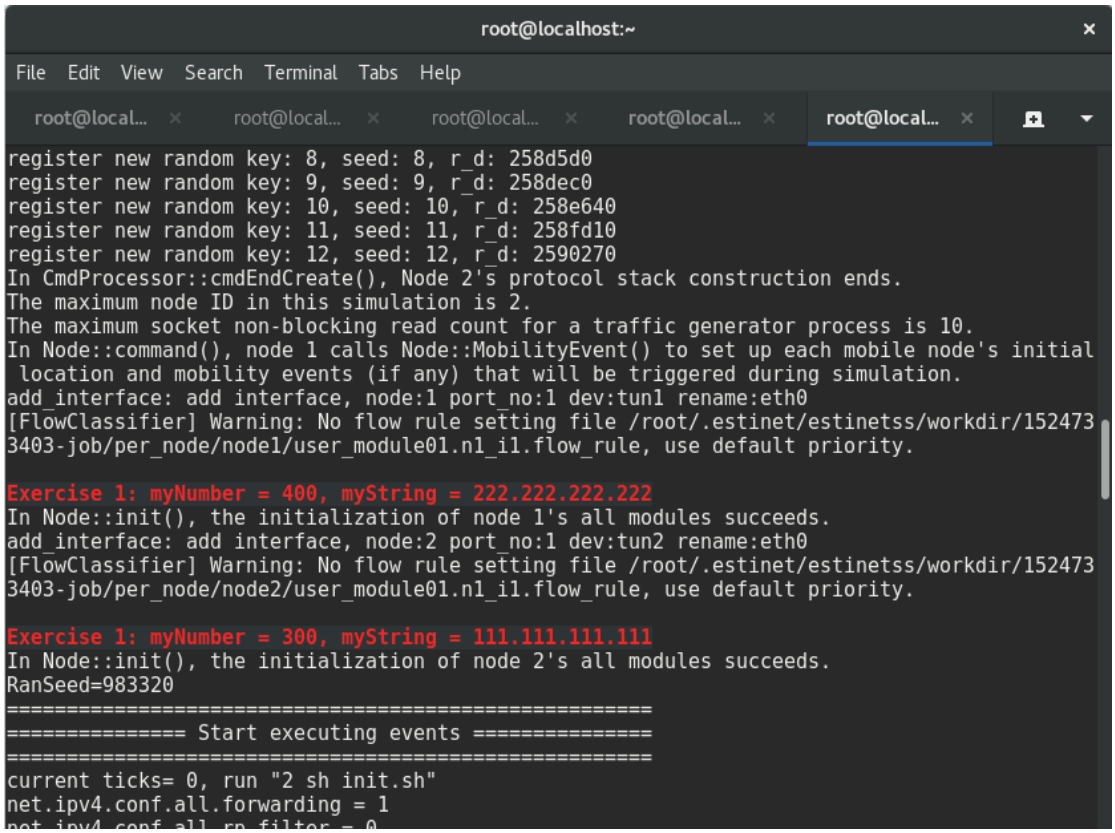
加入 MDF 的两个 TEXTLINE 对象，在 estinetse_all 目录下需要执行 make install，将新改好的 MDF 檔搬移到 GUI 读取的位置。但因没有新加参数，GUI 不用重新开启，可以直接到 Module Editor 看 USER MODELE 01 模块加入对象的样子。

如下图所示：



在 Host1 上的 My Number 输入 400，My string 输入 222.222.222.222 再执行仿真一次，注意 estinetss 窗口，可以看到 Host1 上打印的值改变了，如下图所示：


Module Editor
Module Group
80211P
WME
Interface
UserModul
UserModule01
Parameters Setting
My Number 400
My String 222.222.222.222
OK
Cancel
Snapshot
Undo
Redraw
C.T.A.M.
C.T.O.N.
OK
Cancel


root@localhost:~
File Edit View Search Terminal Tabs Help
root@local...
root@local...
root@local...
root@local...
root@local...
register new random key: 8, seed: 8, r_d: 258d5d0
register new random key: 9, seed: 9, r_d: 258dec0
register new random key: 10, seed: 10, r_d: 258e640
register new random key: 11, seed: 11, r_d: 258fd10
register new random key: 12, seed: 12, r_d: 2590270
In CmdProcessor::cmdEndCreate(), Node 2's protocol stack construction ends.
The maximum node ID in this simulation is 2.
The maximum socket non-blocking read count for a traffic generator process is 10.
In Node::command(), node 1 calls Node::MobilityEvent() to set up each mobile node's initial location and mobility events (if any) that will be triggered during simulation.
add_interface: add interface, node:1 port_no:1 dev:tun1 rename:eth0
[FlowClassifier] Warning: No flow rule setting file /root/.estinet/estinetss/workdir/1524733403-job/per_node/node1/user_module01.n1_i1.flow_rule, use default priority.
Exercise 1: myNumber = 400, myString = 222.222.222.222
In Node::init(), the initialization of node 1's all modules succeeds.
add_interface: add interface, node:2 port_no:1 dev:tun2 rename:eth0
[FlowClassifier] Warning: No flow rule setting file /root/.estinet/estinetss/workdir/1524733403-job/per_node/node2/user_module01.n1_i1.flow_rule, use default priority.
Exercise 1: myNumber = 300, myString = 111.111.111.111
In Node::init(), the initialization of node 2's all modules succeeds.
RanSeed=983320
==========
Start executing events
==========
current ticks= 0, run "2 sh init.sh"
net.ipv4.conf.all.forwarding = 1

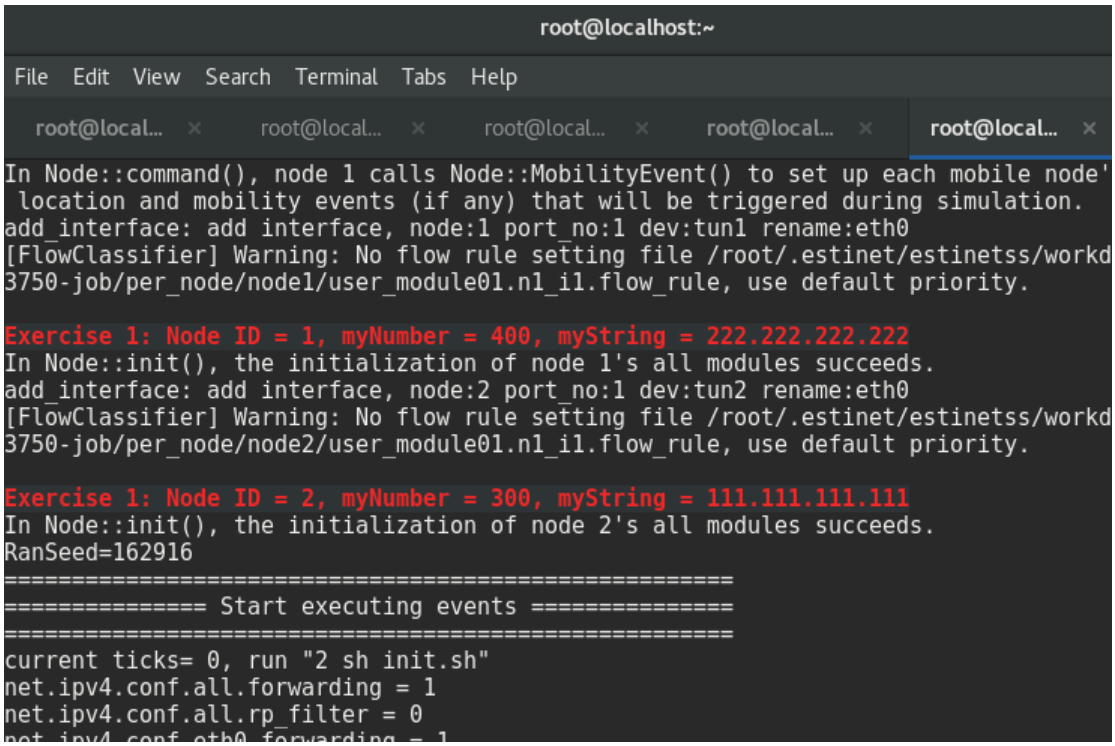
因此可以开发者可以经由 MDF 搭配 GUI，来设定要给仿真引擎的参数。

- 使用 get_nid()

  从打印结果中，其实看不出是哪一个节点印的，可以使用 get_nid()这个 API 来搭配打印：

  ```
  printf("\e[1;31;40m\nExercise 1: Node ID = %d, myNumber = %d, myString = %s\e[m\n", get_nid(), Number, String);
  ```

  打印结果如下图所示：



- TEXTLINE 的细节

  TEXTLINE 其实还有许多细部的参数可以设定，请查看附录 B，有 MDF 所有参数完整的介绍。下面简单介绍常用的部分。

  Caption：

  可以设定此对象的标题

Scale

如上述中，"Scale 10 48 220 30"，四个数值分别代表 X，Y，宽度、高度。左上角的 X,Y 为 0, 0。

ActiveOn

有三个值可以设定，MODE_EDIT, MODE_SIMULATION,ALL_MODE。主要是设定这个对象在哪一个 GUI 的模式下可 active 的。如果设定为 MODE_EDIT，则这个对象在 Edit 模式下是 enable 的。而 MODE_SIMULATION 则是在 G 模式下是 enable 的；ALL_MODE 值则是不论在哪一个模式下都是 enable 的。

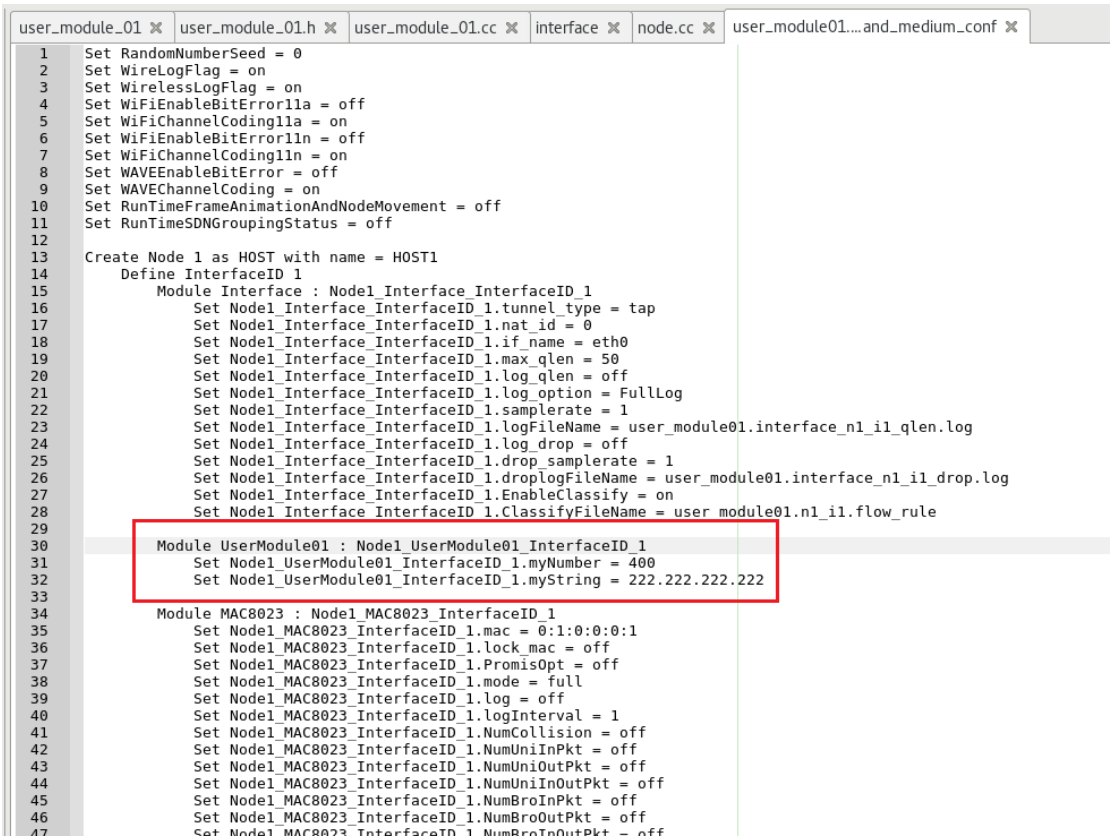
Enabled

这个参数如果是 OFF，则此对象则没有 enable(灰阶)，用户无法使用。如果为 on，或是在使用在其它模式被 active，才会被 enable，让使用者使用。

■ 补充：vBind API

上述练习中，仿真引擎中有使用到 vBind 函数，主要是因为要把 GUI 产出的 if_and_medium_conf 中的模块变量，输入到仿真引擎中的模块中。

上述练习 1-1 中在设定好 MDF 后，在 GUI 执行模拟时，会将 if_and_medium_conf 文件送给仿真引擎，而 if_and_medium_conf 中 UserModule01 的参数值如下所示：



因此，模块要透过 vBind 函式的方式，读入 if_and_medium_conf 这些值，用 vBind，预期他是什么数据型态输入：目前仿真引擎这部分，支持了十种左右的 vBind 数据型别(int、uint8、uint16、bool、float、double、char_str、ip、ipv6、mac)，前者是 MDF 中所变量名，后者是模块中的变量名。下面举例 vBind_int 这个 API。

vBind_int("myNumber", &Number);

- 练习 1-2
  - RADIOBOX & GROUP

    接下来使用 RADIOBOX 这个对象练习 Layout。

    要特别注意的是使用 RADIOBOX，外层还需要搭配 GROUP 对象，因 RADIOBOX 的长宽高度需要跟 GROUP 对象配合。

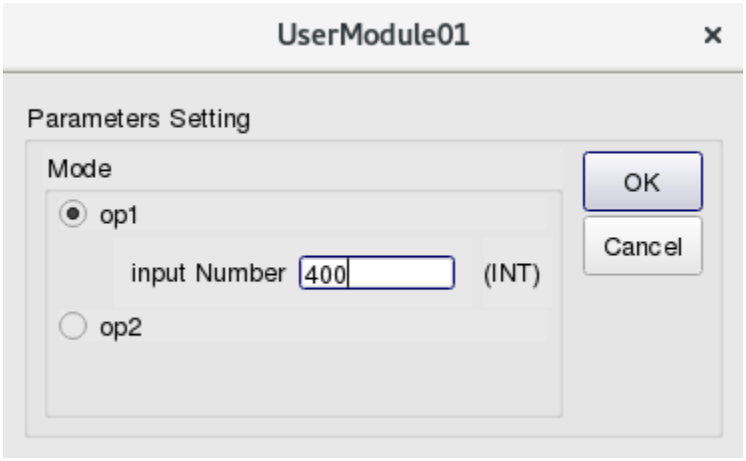
    MDF 设计如下所示：

```
ModuleSection
    HeaderSection
        ModuleName          UserModule01
```

```
        GroupName           User_Defined
        Introduction        "This is a user-defined module."
        Parameter           myNumber 300 local
        Parameter           myString string1 local
EndHeaderSection

InitVariableSection
        Caption             "Parameters Setting"
        FrameSize           350 170

        Begin Group         g_radio
            Caption         "Mode"
            Scale           10 15 260 135
            ActiveOn        MODE_EDIT
            Enabled         TRUE

            Begin RADIOBOX      myString
                Option          "op1"
                Enable          myNumber
                Enable          lable1
                OptValue        "string1"
                VSpace          5
                EndOption

                Option          "op2"
                Disable         myNumber
                Disable         lable1
                OptValue        "string2"
                VSpace          40
                EndOption

                Type            STRING
                Comment         "radiobox test"
            End

           Begin TEXTLINE       myNumber
                Caption         "input Number"
                Scale           35 35 180 35
```

```
                    ActiveOn        MODE_EDIT
                    Enabled         FALSE
                    Type            INT
                    Comment         "for test"
                End

                Begin LABEL         lable1
                    Caption         "(INT)"
                    Scale           220 35 35 35
                    ActiveOn        MODE_EDIT
                    Enabled         FALSE
                End
            End

            Begin BUTTON            b_ok
                Caption             "OK"
                Scale               280 17 60 30
                ActiveOn            ALL_MODE
                Action              ok
                Comment             "OK Button"
            End

            Begin BUTTON            b_cancel
                Caption             "Cancel"
                Scale               280 49 60 30
                ActiveOn            ALL_MODE
                Action              cancel
                Comment             "Cancel Button"
            End
        EndInitVariableSection

        ExportSection
            Caption             ""
            FrameSize           0 0
        EndExportSection
EndModuleSection
```

接着将 MDF 檔 install，开启 GUI 后，执行画面如下所示：



RADIOBOX 及 GROUP 相关参数的细节，请参考附录 B。

- 练习 1-3
  - CHECKBOX

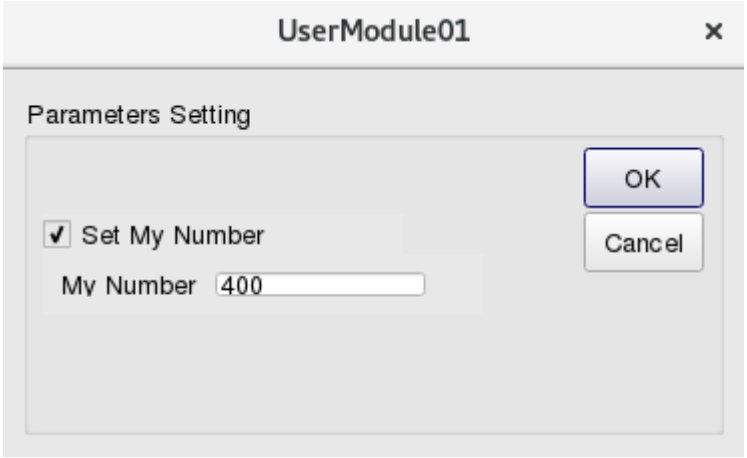
    接下来使用 CHECKBOX，请用户使用下面范例试看看。

```
ModuleSection
    HeaderSection
        ModuleName          UserModule01
        GroupName           User_Defined
        Introduction        "This is a user-defined module."
        Parameter           myNumber 300 local
        Parameter           myString 111.111.111.111 local
    EndHeaderSection

    InitVariableSection
        Caption             "Parameters Setting"
        FrameSize           350 170

        Begin CHECKBOX      check1
            Caption         "Set My Number"
            Scale           10 50 180 20
            ActiveOn        MODE_EDIT
            Enabled         TRUE

            Option          "TRUE"
            OptValue        "on"
```

```
            Enable          myNumber
            EndOption

            Option          "FALSE"
            OptValue        "off"
            Disable         myNumber
            EndOption
            Comment         ""
        End

        Begin TEXTLINE      myNumber
            Caption         "My Number "
            Scale           10 70 220 30
            ActiveOn        MODE_EDIT
            Enabled         FALSE

            Type            INT
            Comment         "An Integer"
        End

        Begin BUTTON        b_ok
            Caption         "OK"
            Scale           280 17 60 30
            ActiveOn        ALL_MODE
            Action          ok
            Comment         "OK Button"
        End

        Begin BUTTON        b_cancel
            Caption         "Cancel"
            Scale           280 49 60 30
            ActiveOn        ALL_MODE
            Action          cancel
            Comment         "Cancel Button"
        End
    EndInitVariableSection

    ExportSection
```

```
        Caption             ""
        FrameSize           0 0
    EndExportSection
EndModuleSection
```

接着将 MDF 檔 install，开启 GUI 后，执行画面如下所示：



CHECKBOX 相关参数的细节，请参考附录 B。

第二章：MDF 与 Run Time Query

本章重點：
(1)、如何使用 RUN TIME QUERY
(2)、第一種 EXPORT SECTION 的物件 -- **ACCESSBUTTON**
(3)、第二種 EXPORT SECTION 的物件 -- **INTERACTIONVIEW**
(4)、介紹 EXPORT 的 API，以及 COMMAND 這個 FUNCTION

下载练习：

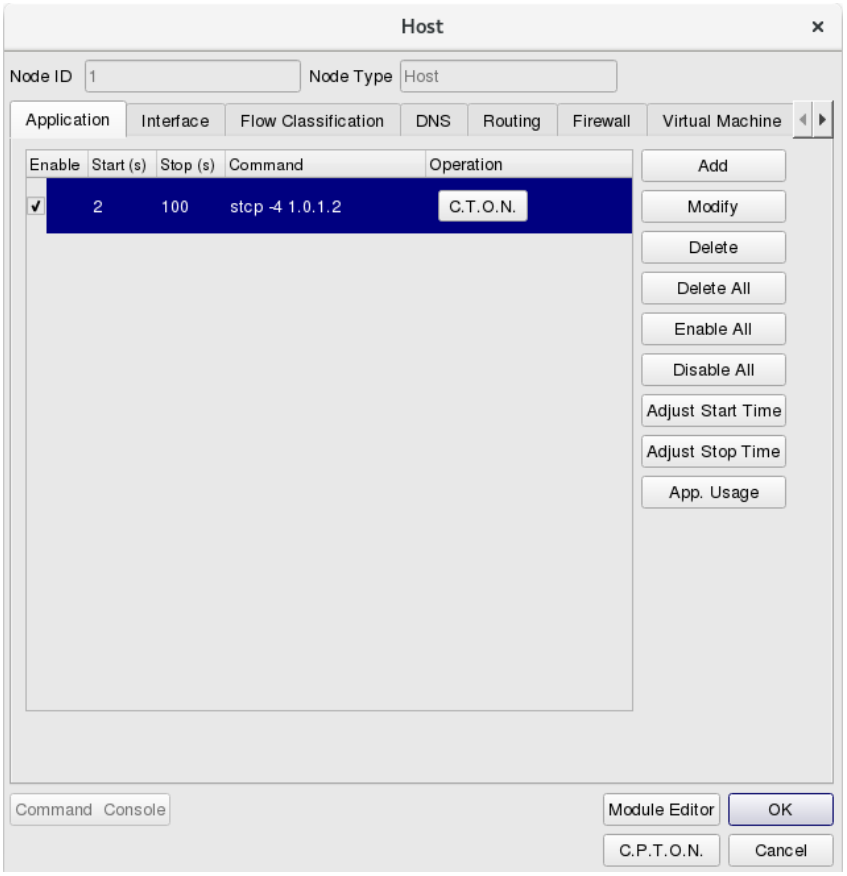

Chapter2.tar.bz2

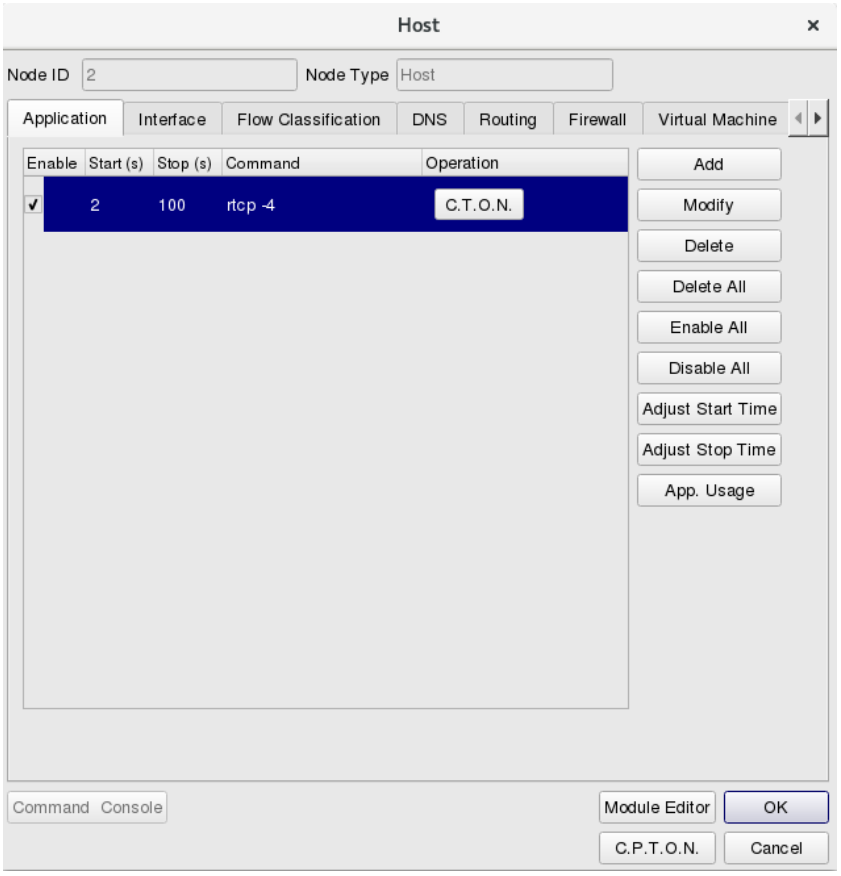
MDF 中可以用来仿真中查询变量的对象有两种，分别是 **ACCESSBUTTON**，另一种是 **INTERACTIONVIEW**。这两种都要搭配模块中的 **COMMAND** 函数以及 **EXPORT** 函数。

- 拓扑设定：

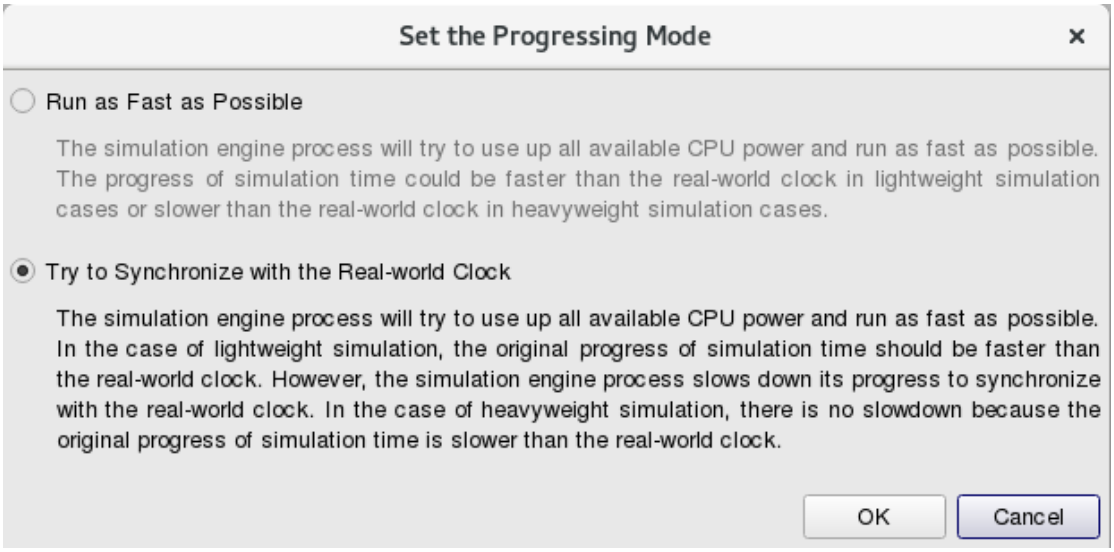
这个章节中同样使用第一章中的 user_module01.xtpl 范例拓扑，并配合 user_module_01 的模块，本章中会介绍可以在模拟执行时，可以读取或设定模块中变量的功能。

首先将范例拓扑中加入通讯流，首先将 HOST 1 中的 application 标签加入 stcp -4 1.0.1.2，而 HOST2 的 application 标签加入 rtcp -4，如下图所示：

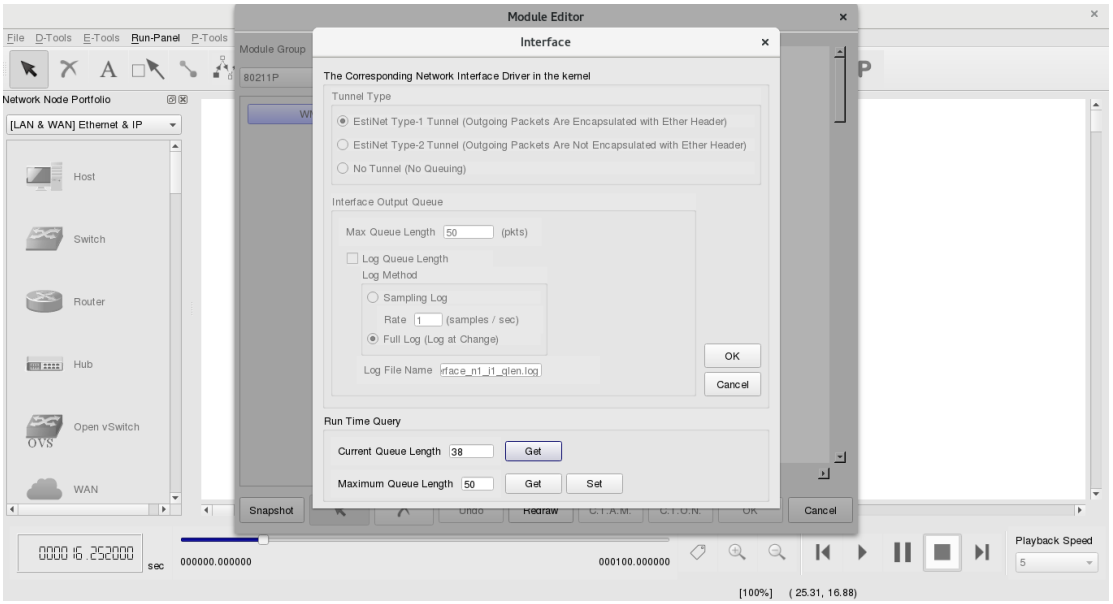

Host
Node ID
1
Node Type
Host
Application
Interface
Flow Classification
DNS
Routing
Firewall
Virtual Machine
Enable
Start (s)
Stop (s)
Command
Operation
2
100
stcp -4 1.0.1.2
C.T.O.N.
Add
Modify
Delete
Delete All
Enable All
Disable All
Adjust Start Time
Adjust Stop Time
App. Usage
Command Console
Module Editor
OK
C.P.T.O.N.
Cancel


Host
Node ID
2
Node Type
Host
Application
Interface
Flow Classification
DNS
Routing
Firewall
Virtual Machine
Enable
Start (s)
Stop (s)
Command
Operation
2
100
rtcp -4
C.T.O.N.
Add
Modify
Delete
Delete All
Enable All
Disable All
Adjust Start Time
Adjust Stop Time
App. Usage
Command Console
Module Editor
OK
C.P.T.O.N.
Cancel

接着为了不要让模拟太快结束，以利观察数值变化，点选上方标题栏 E_Tools 的”Configure Simulation Processes→Simulation→Set the Progressing Mode→Try to Synchronize the Real-World Clock”，让虚拟时间与真实时间一致。



接着切换到 G mode 执行模拟，执行模拟中在 Host1 上点选右键，点选 Module Editor，点选 Interface 模块两下，看到最下方的 Export section 可以 Get 跟 Set 一些 Queue 参数，点选后可以从模块取到目前仿真中的数据，如下图所示。



因此，在练习 2-1 中要在 User module01 中，加入这个 Run Time Query 的功能。

- 练习 2-1：修改 mdf

首先开启 user module 01 的 mdf(开启的位置跟方式在第一章说过，不再重复说明)，接着将最下方的 Export Section 改为下方红字所示：

```
ModuleSection
    HeaderSection
        ModuleName          UserModule01
        GroupName           User_Defined
        Introduction        "This is a user-defined module."
        Parameter           myNumber 300 local
        Parameter           myString 111.111.111.111 local
    EndHeaderSection

    InitVariableSection
        Caption             "Parameters Setting"
        FrameSize           380 100

        Begin TEXTLINE      myNumber
            Caption         "My Number "
            Scale           10 18 220 30
            ActiveOn        MODE_EDIT
            Enabled         TRUE
```

```
            Type            INT
            Comment         "An Integer"
        End

        Begin TEXTLINE      myString
            Caption         "My String "
            Scale           10 48 220 30
            ActiveOn        MODE_EDIT
            Enabled         TRUE
            Type            IP
            Comment         "An IP string"
        End

        Begin BUTTON        b_ok
            Caption         "OK"
            Scale           250 17 60 30
            ActiveOn        ALL_MODE
            Action          ok
            Comment         "OK Button"
        End

        Begin BUTTON        b_cancel
            Caption         "Cancel"
            Scale           250 49 60 30
            ActiveOn        ALL_MODE
            Action          cancel
            Comment         "Cancel Button"
        End
EndInitVariableSection

ExportSection
        Caption             ""
        FrameSize           380 120

        Begin TEXTLINE      text_query_mynum
            Caption         "My Number "
            Scale           10 15 200 35
            ActiveOn        MODE_SIMULATION
```

```
        Enabled         TRUE
        Type            INT
        Comment         ""
End

Begin TEXTLINE          text_query_mystr
        Caption         "My String "
        Scale           10 55 200 35
        ActiveOn        MODE_SIMULATION
        Enabled         TRUE

        Type            INT
        Comment         ""
End

Begin ACCESSBUTTON      ab_get_mynum
        Caption         "Get"
        Scale           215 20 70 25
        ActiveOn        MODE_SIMULATION
        Enabled         TRUE

        Action          GET
        ActionObj       "export-my-number"

        Reference       text_query_mynum
        Comment         "get"
End

Begin ACCESSBUTTON      ab_get_mystr
        Caption         "Get"
        Scale           215 55 70 25
        ActiveOn        MODE_SIMULATION
        Enabled         TRUE

        Action          GET
        ActionObj       "export-my-string"

        Reference       text_query_mystr
```

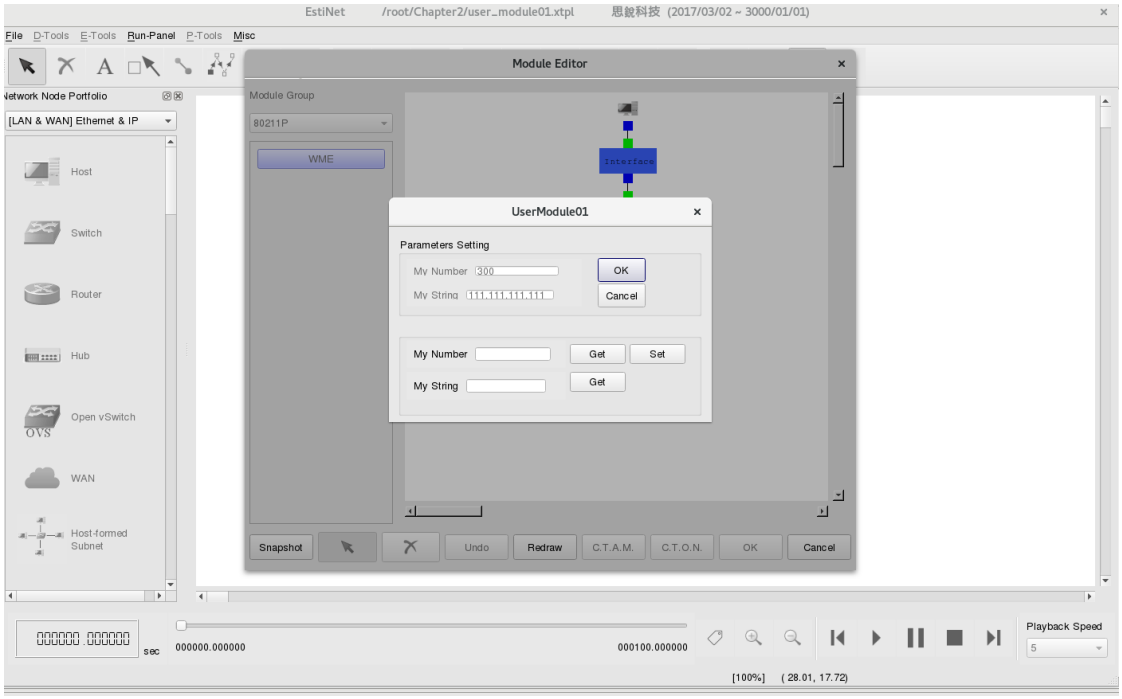
```
            Comment         "get"
        End

        Begin ACCESSBUTTON  ab_set_mynum
            Caption         "Set"
            Scale           290 20 70 25
            ActiveOn        MODE_SIMULATION
            Enabled         TRUE

            Action           SET
            ActionObj       "export-my-number"

            Reference       text_query_mynum
            Comment         "set"
        End
    EndExportSection
EndModuleSection
```

最下面 EXPORT SECTION 区块中，将 CAPTION 加入”RUN TIME QUERY”，FRMAESIZE 把 0 0 改为 380 120(宽度、高度)，接着加入三个 ACCESSBUTTON 的对象，其中两个为 ab_get_mynum 跟 ab_get_mystr 另一个为 ab_set_mystr。

修改后，在原始码目录中使用 make install，将 mdf 安装到 GUI，再开启 GUI 观看修改后的 Layout。

Export section 的参数细节，使用者可以于附录 C 查看。

改完之后的对象可以在开启 GUI 后，在 G MODE，开启 Host1 的 Module Editor 如下图显示：



- 修改 user_module_01.cc 檔：

接着，再回到原始码目录修改 user_module_01.cc，除了在第一个范例需要使用到 VBIND 等 API 外，还需要使用 EXPORT 的 API 以及在 command 函数增加解析 GUI 传送过来的命令的程序代码，如下所示：

在 init()函数中，加入 EXPORT 的 API，EXPORT API 的第一个参数是 GUI 的 MDF ActionObj 所设定的字符串，第二个参数则是设定此字符串读写的权限，其参数值有 E_RONLY(只能读不能写)、E_WONLY (只能写不能读)以及 E_RONLY|E_E_WONLY(可擦写) ，已"export-my-number"为例，此 EXPORT 如同下列红字所示:

```
EXPORT("export-my-number", E_RONLY|E_WONLY);
```

接着修改 command 函式，这个函式每个模块中都有，是在 run time 时，可以读取 GUI 传送过来的命令。因此接收到 GUI 传送过来的命令时，首先需判断命令是属于 GET 或 SET 哪种类型，接着再根据 ActionObj 所设定的字符串取得或设定其对应的参数，如下方 code 红色字体所示:

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>
```

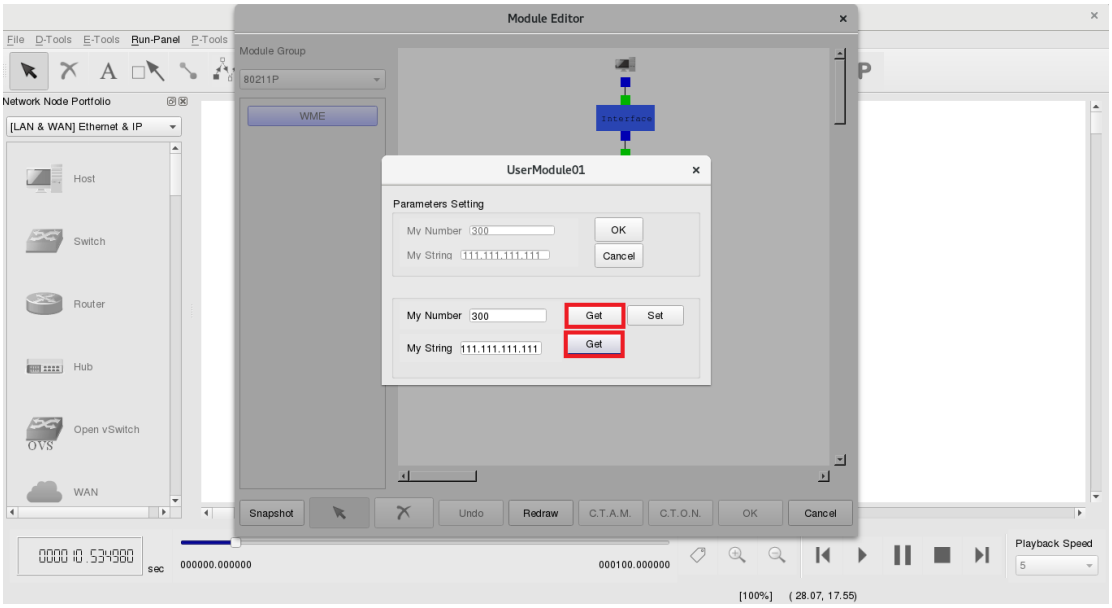
```
MODULE_GENERATOR(UserModule01);

UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name)
        : NslObject(type, id, pl, name) {
        vBind_int("myNumber", &Number);
        vBind_char_str("myString", &String);
}

UserModule01::~UserModule01(){}

int UserModule01::init() {
        EXPORT("export-my-number", E_RONLY|E_WONLY);
        EXPORT("export-my-string", E_RONLY|E_WONLY);
        /* exercise 1 */
        printf("\e[1;31;40m\nExercise 1: Node ID = %d, myNumber = %d, myString = %s\e[m\n",
                        get_nid(), Number, String);
        return(NslObject::init());
}

int UserModule01::command(int argc, const char *argv[]) {

        char                tmpBuf[10];
        struct ExportStr    *ExpStr;
        u_int32_t           row,column;
        u_long *mytunidp;

        /* The Get implementation of Exported Variable "export-my-number" */
             if (!strcmp(argv[0], "Get")&&(argc==2)) {
                        if (!strcmp(argv[1], "export-my-number")) {
                    ExpStr = new ExportStr(1);
                    row = ExpStr->Add_row();
                    column = 1;

                    bzero(tmpBuf, sizeof(tmpBuf));
                                sprintf(tmpBuf, "%d", Number);
                    ExpStr->Insert_cell(row, column, tmpBuf, "\n");
                    EXPORT_GET_SUCCESS(ExpStr);
```

```
                        return 1;
                    }
            }

        /* The Get implementation of Exported Variable "export-my-string" */
            if (!strcmp(argv[0], "Get")&&(argc==2)) {
                    if (!strcmp(argv[1], "export-my-string")) {
                ExpStr = new ExportStr(1);
                row = ExpStr->Add_row();
                column = 1;

                bzero(tmpBuf, sizeof(tmpBuf));
                        sprintf(tmpBuf, "%s", String);
                ExpStr->Insert_cell(row, column, tmpBuf, "\n");
                EXPORT_GET_SUCCESS(ExpStr);
                        return 1;
                    }
            }

        /* The Set implementation of Exported Variable "export-my-number" */
            if (!strcmp(argv[0], "Set")&&(argc==3)) {
                    if (!strcmp(argv[1], "export-my-number")) {
                        Number = atoi(argv[2]);
                EXPORT_SET_SUCCESS();
                        return 1 ;
                    }
            }
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        return(NslObject::send(pkt));
}
```

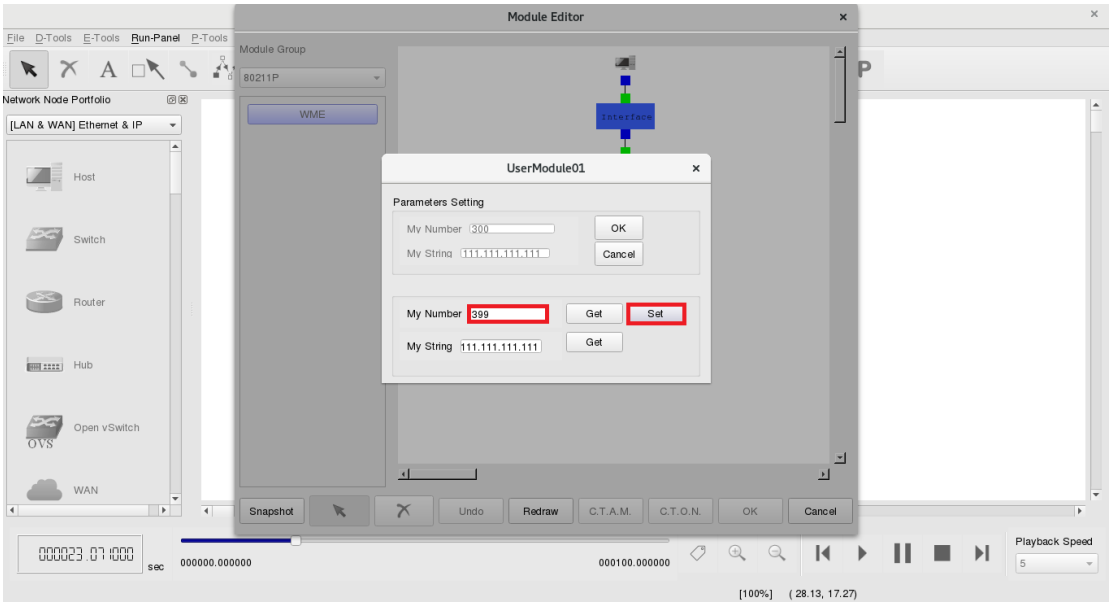
NOTE: VBIND 与 EXPORT 函数的差异
VBIND 是在模拟一开始时，读取 if_and_medium_conf 中 MDF 已设定好的数值。 EXPORT 是在仿真中经由 IPC，在模块中 COMMAND 函数读写 GUI 中的 MDF 数值。每个模块中都有 COMMAND 函数。

完成后在原始码目录进行 make 与 make install。

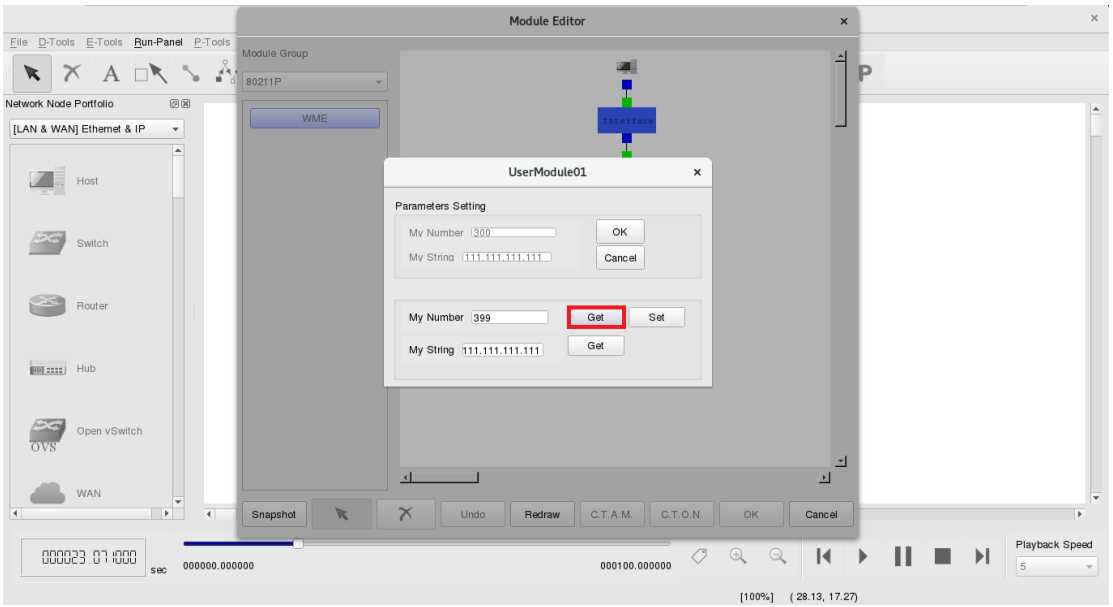
接着进行模拟，模拟时在 HOST1 上右键单击选择”Module Editor”按钮，并在 user module01 点两下，可以按下 Get 按钮来取得目前仿真引擎中的 Number 的数值，也可以 Get 到模块中 String 的数值，如下图所示：



接着修改 My Number 300 为 399，按下 Set 按钮，如下图所示：



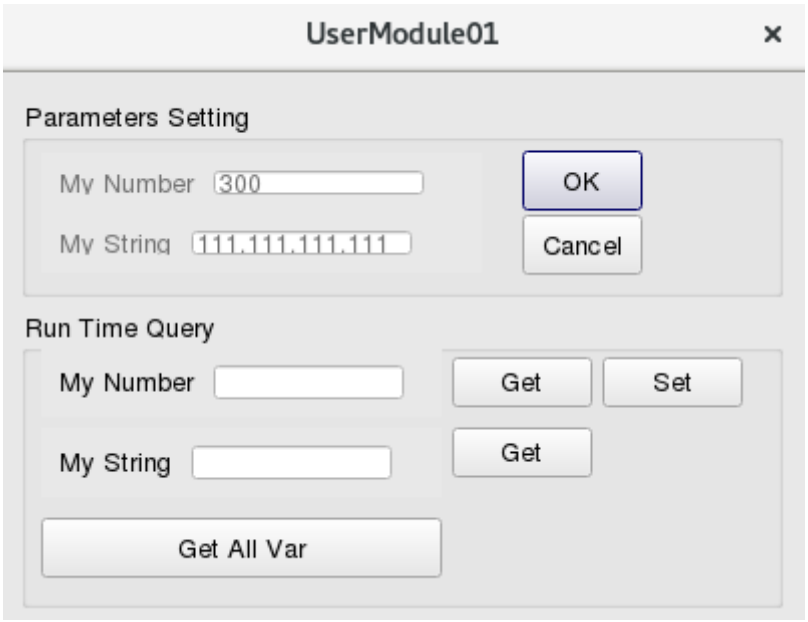
接着再按一次 My Number 的 Get 按钮，发现 My Number 的值已经修改成功，如下图所示：

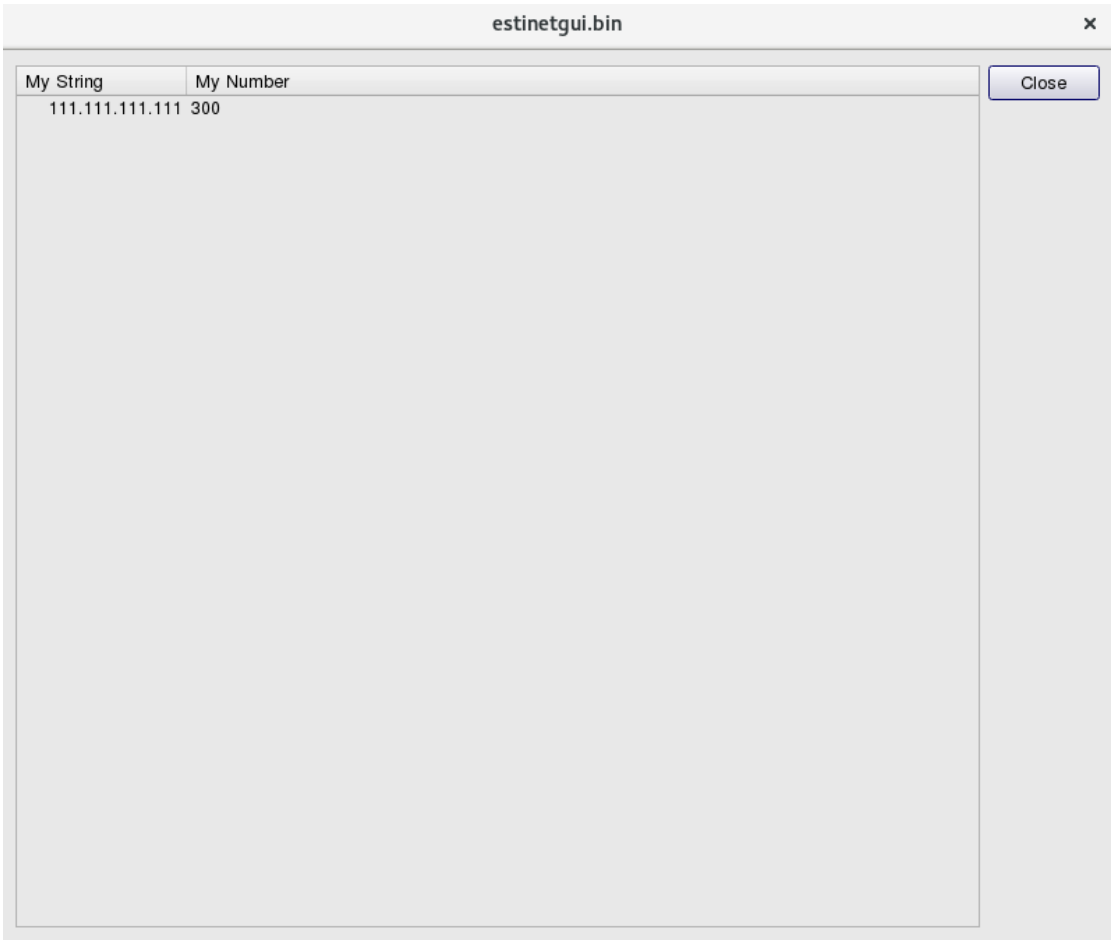


- 另一种 EXPORT 物件
  RUN TIME QUERY 也可以用表格呈现的方式。

- 练习 2-2：

这种呈现方式是使用 GUI 中的 **INTERACTIONVIEW** 对象，因此修改 USER MODULE 01 的 MDF，改成如下图所示：





将 EXPORT SECTION 的区块改写为如下所示：

```
ModuleSection
    HeaderSection
        ModuleName          UserModule01
        GroupName           User_Defined
        Introduction        "This is a user-defined module."
        Parameter           myNumber 300 local
        Parameter           myString 111.111.111.111 local
    EndHeaderSection

    InitVariableSection
        Caption             "Parameters Setting"
        FrameSize           380 100

        Begin TEXTLINE          myNumber
            Caption         "My Number "
            Scale           10 18 220 30
            ActiveOn        MODE_EDIT
            Enabled         TRUE
            Type            INT
            Comment         "An Integer"
        End

        Begin TEXTLINE          myString
            Caption         "My String "
            Scale           10 48 220 30
            ActiveOn        MODE_EDIT
            Enabled         TRUE
            Type            IP
            Comment         "An IP string"
        End

        Begin BUTTON        b_ok
            Caption         "OK"
            Scale           250 17 60 30
            ActiveOn        ALL_MODE
            Action          ok
            Comment         "OK Button"
        End
```

```
        Begin BUTTON        b_cancel
            Caption         "Cancel"
            Scale           250 49 60 30
            ActiveOn        ALL_MODE
            Action          cancel
            Comment         "Cancel Button"
        End
EndInitVariableSection

ExportSection
        Caption         "Run Time Query"
        FrameSize       380 150

        Begin TEXTLINE          text_query_mynum
            Caption         "My Number "
            Scale           10 10 200 35
            ActiveOn        MODE_SIMULATION
            Enabled         TRUE

            Type        INT
            Comment         ""
        End

        Begin TEXTLINE          text_query_mystr
            Caption         "My String "
            Scale           10 50 200 35
            ActiveOn        MODE_SIMULATION
            Enabled         TRUE

            Type        INT
            Comment         ""
        End

        Begin ACCESSBUTTON      ab_get_mynum
            Caption         "Get"
            Scale           215 15 70 25
            ActiveOn        MODE_SIMULATION
```

```
        Enabled         TRUE

        Action          GET
        ActionObj       "export-my-number"

        Reference       text_query_mynum
        Comment         "get"
End

Begin ACCESSBUTTON      ab_get_mystr
        Caption         "Get"
        Scale           215 50 70 25
        ActiveOn        MODE_SIMULATION
        Enabled         TRUE

        Action          GET
        ActionObj       "export-my-string"

        Reference       text_query_mystr
        Comment         "get"
End

Begin ACCESSBUTTON      ab_set_mynum
        Caption         "Set"
        Scale           290 15 70 25
        ActiveOn        MODE_SIMULATION
        Enabled         TRUE

        Action           SET
        ActionObj       "export-my-number"

        Reference       text_query_mynum
        Comment         "set"
End

Begin INTERACTIONVIEW  iv_get_all
        Caption         "Get All Var"
        Scale           10 100 200 30
```

```
                ActiveOn        MODE_SIMULATION
                Enabled         TRUE

                Action          GET
                ActionObj       "export-all-data"

                Fields          "My String" "My Number"
                Comment         "All Data"
            End

    EndExportSection
EndModuleSection
```

user_module01.cc 修改成：

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>

MODULE_GENERATOR(UserModule01);

UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name)
    : NslObject(type, id, pl, name) {
    vBind_int("myNumber", &Number);
    vBind_char_str("myString", &String);
}

UserModule01::~UserModule01(){}

int UserModule01::init() {
     EXPORT("export-my-number", E_RONLY|E_WONLY);
    EXPORT("export-my-string", E_RONLY|E_WONLY);
    EXPORT("export-all-data", E_RONLY|E_WONLY);
    /* exercise 1 */
    printf("\e[1;31;40m\nExercise 1: Node ID = %d, myNumber = %d, myString = %s\e[m\n",
            get_nid(), Number, String);
    return(NslObject::init());
}
```

```
int UserModule01::command(int argc, const char *argv[]) {

        char                tmpBuf[10];
        struct ExportStr    *ExpStr;
        u_int32_t           row,column;
        u_long *mytunidp;

        /* The Get implementation of Exported Variable "export-my-number" */
            if (!strcmp(argv[0], "Get")&&(argc==2)) {
                        if (!strcmp(argv[1], "export-my-number")) {
                    ExpStr = new ExportStr(1);
                    row = ExpStr->Add_row();
                    column = 1;

                    bzero(tmpBuf, sizeof(tmpBuf));
                                sprintf(tmpBuf, "%d", Number);
                    ExpStr->Insert_cell(row, column, tmpBuf, "\n");
                    EXPORT_GET_SUCCESS(ExpStr);
                                return 1;
                      }
            }

        /* The Get implementation of Exported Variable "export-my-string" */
            if (!strcmp(argv[0], "Get")&&(argc==2)) {
                        if (!strcmp(argv[1], "export-my-string")) {
                    ExpStr = new ExportStr(1);
                    row = ExpStr->Add_row();
                    column = 1;

                    bzero(tmpBuf, sizeof(tmpBuf));
                                sprintf(tmpBuf, "%s", String);
                    ExpStr->Insert_cell(row, column, tmpBuf, "\n");
                    EXPORT_GET_SUCCESS(ExpStr);
                                return 1;
                      }
            }
```

```
        /* The Set implementation of Exported Variable "export-my-number" */
            if (!strcmp(argv[0], "Set")&&(argc==3)) {
                        if (!strcmp(argv[1], "export-my-number")) {
                                    Number = atoi(argv[2]);
                        EXPORT_SET_SUCCESS();
                                    return 1 ;
                        }
             }

            /* The Set implementation of Exported Variable "export-all-data" */
            if (!strcmp(argv[0], "Get")&&(argc==2)) {
                        if (!strcmp(argv[1], "export-all-data")) {

                        ExpStr = new ExportStr(2);

                        bzero(tmpBuf, sizeof(tmpBuf));
                        sprintf(tmpBuf, "String\t\tNumber\n");
                        ExpStr->Insert_comment(tmpBuf);

                        row = ExpStr->Add_row();
                        column = 1;

                        bzero(tmpBuf, sizeof(tmpBuf));
                              sprintf(tmpBuf, "%s", String);
                        ExpStr->Insert_cell(row, column++, tmpBuf, "\t\t");

                        bzero(tmpBuf, sizeof(tmpBuf));
                        sprintf(tmpBuf, "%d", Number);
                        ExpStr->Insert_cell(row, column, tmpBuf, "\n");

                        EXPORT_GET_SUCCESS(ExpStr);
                                    return 1 ;
                        }
            }

        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
```

```
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        return(NslObject::send(pkt));
}
```

修改完毕后，在原始码目录中执行 make、make install，并执行模拟，即可看到上图中的呈现效果。

第三章：模块间互相分享数据

本章重點：
(1)、介紹 REG_VAR、GET_REG_VAR 的使用範例
(2)、介紹 INSTANCELOOKUP 的使用範例

下载练习：


Chapter3.tar.bz2

这一章说明模块之间分享变量或是函式给其它模块使用，例如模块中若需要取得 phy 模块的数据来运算(如带宽数据等)。在仿真器的模块平台运作中，每个节点上都有自己的 protocol stack，因此会有许多的模块运作，在节点结构中提供一个公用的数据结构为”var-register-table”，可以让 protocol stack 中的模块，注册变量到此结构中，而其它模块就可以到这个数据结构来取得此分享的变量数据。

- 注册变数
  REG_VAR()是注册模块中的变量到节点公用数据结构--"var-register-table"，让这个节点上 protocol stack 的所有模块，都可以读取所注册的变量。
  例如在本练习范例中，会在 phy 模块中取得 USER MODULE 01 所注册的变量。
  用法：REG_VAR(vname, var)
  第一个参数是一个指标，指向一个识别的名称，这个名称将用来识别这些注册的变量；而第二个参数则是一个指标指向这个分享的变量。

- 取得其它模块中的注册变量
  GET_REG_VAR()是取得这个节点中的其它模块所注册的变量。
  用法：GET_REG_VAR(interface id, vname, type)
  第一个参数是 interface id，是指 protocol stack 上的 interface。我们可以使用 API(get_ifid())来取得 protocol stack 上的 interface id，更精准的指向某个模块所分享出来的变量。

- get_ifid()用法

get_ifid()会回传目前模块所连接到的 interface id。

- 练习 3-1：REG_VAR()与 GET_REG_VAR()

  首先使用第一章中的范例拓扑，并配合 USER MODULE 01 的模块，修改 user_module01.cc 的档案。将 Number 变量分享给其它模块使用，在建构子中加入注册的信息，如下方红字所示：

  user_module01.cc 修改成：

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>

MODULE_GENERATOR(UserModule01);

UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct iflist* ifl, const char
*name): NslObject(type, id, ifl, name) {
        vBind_int("myNumber", &Number);
        vBind_char_str("myString", &String);
        REG_VAR("shared-number", &Number);
}

UserModule01::~UserModule01(){}

int UserModule01::init() {
        return(NslObject::init());
}

int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        return(NslObject::send(pkt));
}
```

接着，在 protocol stack 的 PHY 模块，取得在 user_module01 模块中所分享出来变量值。修改原始码目录中的 module/8023/phy/phy.cc，主要只在 init()函数中，打印 user_module01 模块所分享出来的变量值，如下方红字。

```
int phy::init() {
        NslObject::init();
        int *get_share_data = GET_REG_VAR(get_ifid(), "shared-number", int *);
        if (get_share_data)
        {
                printf("\e[1;36;46m\nExercise3-1: Get Shared Number in
UserModule01= %d\e[m\n", *get_share_data);
        }
 }
```

执行结果，如下图所示，phy 模块中，可以经由 **GET_REG_VAR** 跟取得 UserModule01 中注册的变量。

```
e's initial location and mobility events (if any) that will be triggered during
simulation.
add_interface: add interface, node:1 port_no:1 dev:tun1 rename:eth0
[FlowClassifier] Warning: No flow rule setting file /root/.estinet/estinetss/wor
kdir/1525186664-job/per_node/node1/user_module01.n1_i1.flow_rule, use default pr
iority.

Exercise3-1: Get Shared Number in UserModule01= 300
In Node::init(), the initialization of node 1's all modules succeeds.
add_interface: add interface, node:2 port_no:1 dev:tun2 rename:eth0
[FlowClassifier] Warning: No flow rule setting file /root/.estinet/estinetss/wor
kdir/1525186664-job/per_node/node2/user_module01.n1_i1.flow_rule, use default pr
iority.

Exercise3-1: Get Shared Number in UserModule01= 300
In Node::init(), the initialization of node 2's all modules succeeds.
RanSeed=485297
=====================================================
=============== Start executing events ===============
=====================================================
current ticks= 0, run "2 sh init.sh"
```

■ 练习 3-2：REG_VAR()与 GET_REG_VAR()

在 phy 模块的程序中，可以看到也有注册分享给其它模块的变量如下：

```
REG_VAR("DataRate", &bw_);
```

这个变量即为带宽的数据，因此练习在 UserModule01 中 send 函式中，取得 phy 模块的 DataRate 数据。

参考解答：

```
int UserModule01::send(ePacket_ *pkt) {

printf("\e[1;33;43m\nExercise 3-2: Get Shared DataRate in Phy Module = %f\e[m\n",
*(GET_REG_VAR(get_ifid(), "DataRate", double *)));
    return(NslObject::send(pkt));
}
```

```
net.ipv6.conf.eth0.autoconf = 0
Current Time:    0.00 sec   Event#: <Insert:22, Dequeue:5, Rest:17>
current ticks= 100100000, run "1 sh init_daemon.sh"
current ticks= 100200000, run "2 sh init_daemon.sh"

Exercise 3-2: Get Shared DataRate in Phy Module = 10000000.000000

Exercise 3-2: Get Shared DataRate in Phy Module = 10000000.000000

Exercise 3-2: Get Shared DataRate in Phy Module = 10000000.000000

Exercise 3-2: Get Shared DataRate in Phy Module = 10000000.000000

Exercise 3-2: Get Shared DataRate in Phy Module = 10000000.000000

Exercise 3-2: Get Shared DataRate in Phy Module = 10000000.000000

Exercise 3-2: Get Shared DataRate in Phy Module = 10000000.000000

Exercise 3-2: Get Shared DataRate in Phy Module = 10000000.000000
Current Time:    1.00 sec   Event#: <Insert:54, Dequeue:54, Rest:17>
```

注：在 user_module01 模块的 init 函式执行上述红字的话，取得的数据会是 0，因为 phy 的 init 的执行比 user_module01 的 init 执行较晚。因此选择 send()函数中打印(但记得要有通讯流，如 ipv6 广播封包或 stcp/rtcp，才会执行到 send 函式)

另外，还有一种方式是模块可以使用其它模块的函式，这个做法需要使用到 InstanceLookup 这个 API。

- InstanceLookup 的用法

首先第一个参数要放这个节点的 ID，可以用 get_nid()这个 API 来取得目前该节点的 ID；第二个参数则是使用 protocol stack 目前连接的 interface id，可以使用 get_ifid()来取得；接着第三个参数则是放 phy 模块注册在整个仿真引擎中的名字，可以透过此名称的指到此对象，后面就可以依照 c++的用法，接成员变量及函式，如下例：

```
((phy *)InstanceLookup(get_nid(), get_ifid(), "Phy"))->Debugger();
```

- 练习 3-3：InstanceLookup()函数

首先改写 user_module01.cc 的程序代码，加入下面红字的程序代码，将 phy.h include 进来，主要等一下要用到 phy 对象中的函式

接着在 send()函式中，使用 InstanceLookup 这个 API，来执行 phy 对象中的 Debugger 函式。

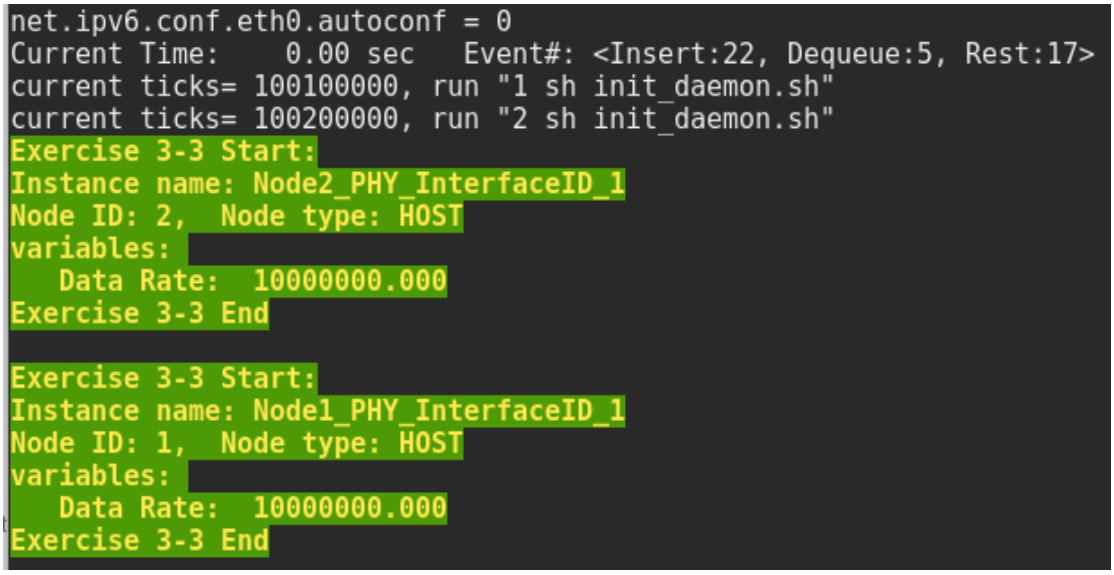
```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>
#include <module/8023/phy/phy.h>

MODULE_GENERATOR(UserModule01);

UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct iflist* ifl, const char
*name): NslObject(type, id, ifl, name) {
        vBind_int("myNumber", &Number);
        vBind_char_str("myString", &String);
        REG_VAR("shared-number", &Number);
}

UserModule01::~UserModule01(){}

int UserModule01::init() {
        return(NslObject::init());
}
```

```
int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        ((phy *)InstanceLookup(get_nid(), get_ifid(), "PHY"))->Debugger();
        return(NslObject::send(pkt));
}
```

接着为了让 PHY 模块中的 Debugger()函式打印到终端机上比较明显，将此函式加入 ascii code 的显示效果：

```
int phy::Debugger() {
        printf("\e[1;33;42mExercise 3-3 Start:\n");
        NslObject::Debugger();
        printf("      Data Rate: %13.3lf\n", bw_);
        printf("Exercise 3-3 End\e[m\n\n");
        return(1);
}
```

make 以及 make install 后，并执行模拟，可以看到如下图绿色区块的效果



因此可以透过此 API，来执行其它模块中的函式。

第四章：RUN TIME MESSAGE

本章重點：
(1)、介紹 RUN TIME MESSAGE -- WARNING 的使用範例 1
(2)、介紹 RUN TIME MESSAGE -- INFOMATION 的使用範例 2
(3)、介紹 RUN TIME MESSAGE – FATAL ERROR 的使用範例 3
(4)、模組的 send()與 recv()函式架構

下载练习：


Chapter4.tar.bz2

Run Time Message 可以在原始码中预先安排一些错误提示、警告提示、或是一些信息提示的程序代码，使得模拟进行时只要执行到这些程序代码，会将这些讯息经由 IPC 送到 GUI，让 GUI 跳出窗口与用户进行互动。

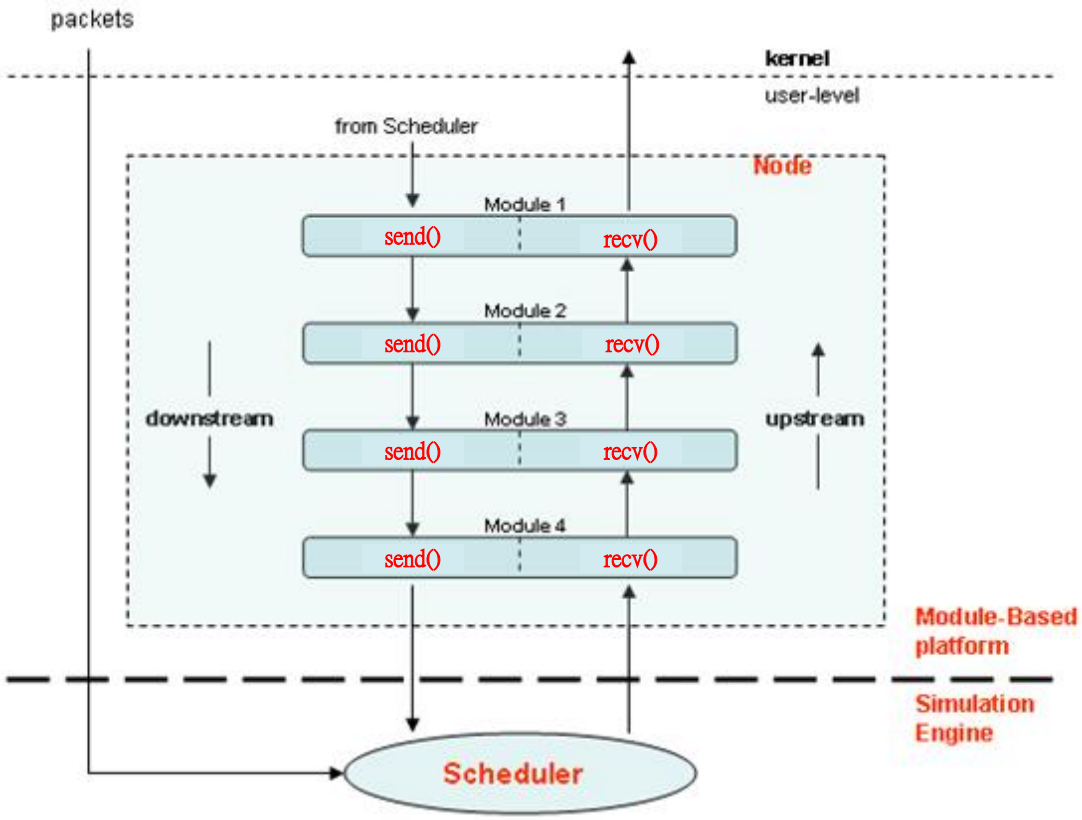
Run Time Message 在仿真引擎中分为三种：

- INFORMATION
- WARNING
- FATAL ERROR

使用三个范例程序代码来展示。

使用第一章中的范例拓扑(user_module01.xtpl)。
预期都在 user_module01.cc 的 send()函式中，各加入一行程序代码展示。

注：由于封包进入模块时，会由 send()进入，处理完成后再送给下一个模块的 send()，直到送到最底层的模块，才会传送给其它节点。接收时则是由每个模块的 recv()由最底层的模块，逐步往上送给每一个模块处理，最后才由 interface 模块将封包经由 kernel 再导向应用程序。如下图所示：



因此使用通讯流来触发 user_module01 模块中的 send()函式，并使用下面三种 run time message 种类来展示：

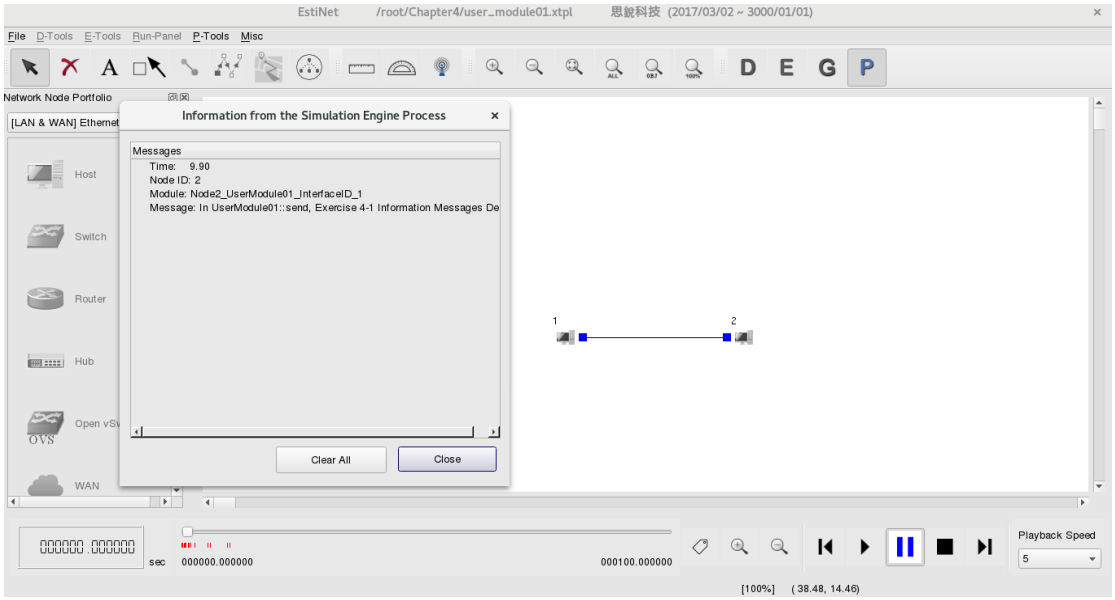
- 练习 4-1：INFORMATION

  sendRuntimeMsg 这个 API 是用来展示 RUN TIME MESSAGE 的 API，第一个参数是指 RUN TIME MESSAGE 的 TYPE，在本章的练习 4-1 中，使用”**RTMSG_INFORMATION**”的 TYPE，第二个参数是放 Node id，第三个参数是模块的名称，可以用 get_name()这个 API 取得，接着第四个参数是我们要呈现在 GUI 窗口中的讯息。如下红字所示：

```
int UserModule01::send(ePacket_ *pkt) {
        sendRuntimeMsg(RTMSG_INFORMATION, get_nid(), get_name(), "In UserModule01::send,
Exercise 4-1 Information Messages Demo");
        return(NslObject::send(pkt));
}
```

make 后再执行 make install，完成编译以及安装。接着执行仿真，就会看到有一个窗口不断的印出讯息，这是因为通讯流不断的有数据，每一笔封包进入 send

函式，就会触发一次这个功能，GUI 会将所有的 INFORMATION 集中在同一个窗口中。



■ 练习 4-2：WARNING

接下来继续修改 user_module01.cc 的程序代码。将练习 4-1 中 sendRuntimeMsg 的第一个参数 TYPE 改为”**RTMSG_WARNING**”，并将第四个参数中的讯息加以改变，如下红字所示：

```
int UserModule01::send(ePacket_ *pkt) {
        sendRuntimeMsg(RTMSG_WARNING, get_nid(), get_name(), "In UserModule01::send,
Exercise 4-2 Warning Messages Demo");
        return(NslObject::send(pkt));
}
```

make 后再执行 make install，完成编译以及安装。接着执行模拟时，可以看到模拟会暂停，并跳出一个互动窗口，询问用户是否要”Continue”或是”Stop”。
如果按”Stop”会直接结束模拟，或是按”Continue”的话，则因为下一个封包马上又触发这个 WARNING 的讯息。用户可以决定此类讯息所出现的时机。

■ 练习 4-3：FATAL ERROR

接下来继续修改 user_module01.cc 的 send 函式，sendRuntimeMsg 第一个参数的 TYPE 为 **RTMSG_FATAL_ERROR**，并将第四个参数中的讯息稍加修改，如下红字所示：

```
int UserModule01::send(ePacket_ *pkt) {
        sendRuntimeMsg(RTMSG_FATAL_ERROR, get_nid(), get_name(), "In
UserModule01::send, Exercise 4-3 Error Messages Demo");
    return(NslObject::send(pkt));
}
```

make 后再执行 make install，完成编译以及安装。接着执行仿真只出现一个互动选项，要求用户只能结束模拟，这在遇到一些错误情况无法再进行模拟时，开发者可以运用此讯息要求结束仿真。

这三种 RUN TIME MESSAGE 都可以设定，可让使用者经由 GUI 知道与仿真引擎的互动信息。三种 RUN TIME MESSAGE 的用途不太相同，开发者可以依自身需求决定使用哪一种 TYPE 的 RUN TIME MESSAGE。

第五章：不需要使用 GUI 的模拟执行方式

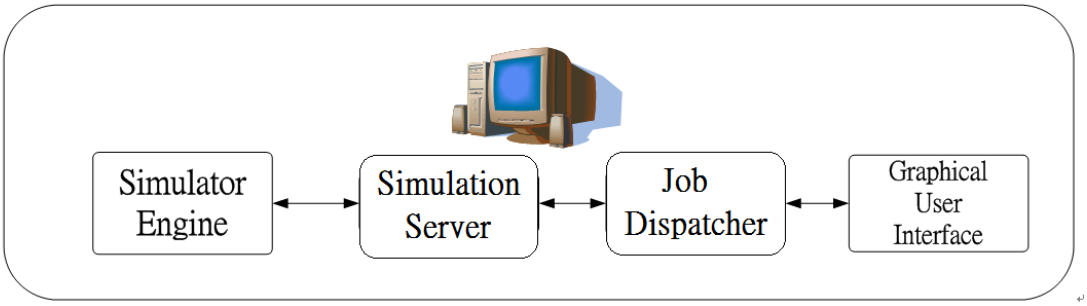
本章重點：
(1)、開發的架構
(2)、關掉 IPC 的使用方式
(3)、查看 Frame Trace File

下载练习：

Chapter5.tar.bz2

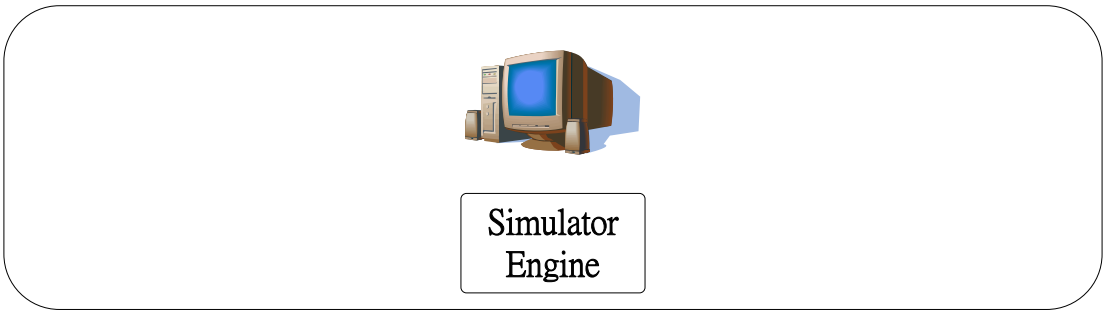
在一些开发中的情况，开发者希望一个更单纯的执行环境，希望跟 GUI 之间的 IPC 关闭，纯粹只有仿真引擎的执行方式。因此本章中将说明如何在没有 GUI 的情况下执行模拟。

一般常用的单机架构中，会像下图架构：(ESTINET 也是分布式架构，于附录 D 说明)



如同在第一章中，在同一台计算机中执行 estinetjd、estinetss、estinetgui 来执行模拟。GUI 负责倒出模拟的配置文件后，将这些配置文件送给 estinetjd 来分配工作，而 estinetjd 再透过 estinetss 来找到闲置的仿真引擎来读取这些配置文件来进行模拟。但这样在开发过程中，有时模块开发中，不见得 GUI 都有支持开发中的节点类型，或是开发人员希望有一个更单纯的模拟环境时，开发者可以选择使用关掉 IPC，只留下仿真引擎存在。

若关掉 IPC 的情况下，架构会更单纯如下图：



开发人员常在这样的情况下，设计 Protocol Module 或 Debug，在开发模块及协议中，更为方便。

因此，使用原本的 user_ moduler01 拓扑(user_moduler01.xtpl)做为说明。

一般来说，模拟使用 GUI 执行过后会有三个目录以及跟一个 GUI 用的档案：user_ moduler01.xtpl 以及 user_ moduler01.gui_data 目录是 GUI 在储存一些 GUI 对象信息的数据跟数据目录，是为 GUI 所使用。而 user_moduler01.sim 目录则是会送交一些配置文件(例如 if_and_medium_conf 文件)给仿真引擎，而 user_moduler01.results 目录是仿真引擎执行的执行结果。

关键步骤就是要将 sim 目录中的配置文件送给仿真引擎。

在不需要使用 GUI 来执行模拟的步骤如下：
首先，要设定一个环境变量”ESTINET_WORKDIR”，设定此目录到 sim 目录，如下图所示：

export ESTINET_WORKDIR=/root/Chapter5/user_module01.sim/

```
[root@localhost Chapter5]# ls user_module01.sim/.for_se_direct_access/
per_node                                                                    user_module01.obstacle
runtime_fatal_error_message_log   user_module01.if_and_medium_conf        user_module01.run
user_module01.application                                                   user_module01.traffic_light_log
user_module01.car_moving_path_log  user_module01.node_type_and_virtualization
[root@localhost Chapter5]# export ESTINET_WORKDIR=/root/Chapter5/user_module01.sim/
[root@localhost Chapter5]# estinetse -d $ESTINET_WORKDIR/.for_se_direct_access/user_module01.if_and_medium_conf
docker daemon is running.
docker images: estinet10/fedora24:v1
turn off docker seccomp setting
clean old container
The maximum number of kernel-supported tunnel interfaces is 40960.
The maximum queue length of kernel-supported data tunnel interfaces is 1000.
The maximum queue length of kernel-supported event tunnel interfaces is 50000.
The limited maximum number of file descriptors is 1048576.
The limited maximum number of forked processes is unlimited.
The limited maximum size of a core dump file is unlimited.
In HeapObject::HeapObject(), the event heap's capacity is 5000000.
Trying to connect to license server 1, please wait
LogID : 36998
Get capablilty_v2 command
```

接着执行 estinetse 下-d 参数表示，关闭与 GUI 之间的 IPC，后面再接上 if_and_medium_conf 的档案路径：
$ESTINET_WORKDIR/.for_se_direct_access/user_moduler01. if_and_medium_conf 即可。

estinetse -d $ESTINET_WORKDIR/.for_se_direct_access/user_module01.if_and_medium_conf

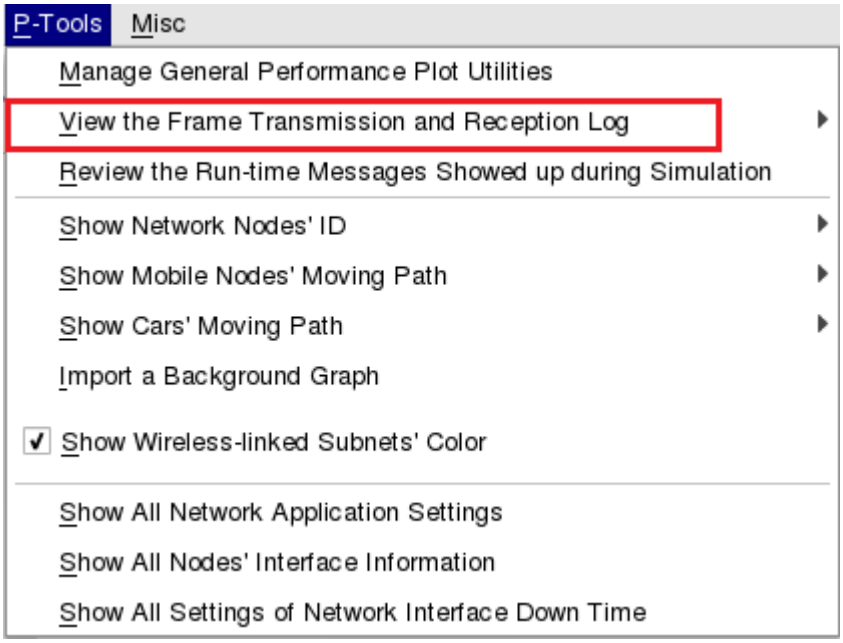
按下 enter 后，开始执行模拟。

```
Current Time:    3.00 sec   Event#: <Insert:2318, Dequeue:2317, Rest:21>
6664     3          1182 Kbyte/sec ==>  9.460160 Mbit/sec
Current Time:    4.00 sec   Event#: <Insert:2299, Dequeue:2299, Rest:21>
6664     4          1183 Kbyte/sec ==>  9.464128 Mbit/sec
Current Time:    5.00 sec   Event#: <Insert:2299, Dequeue:2299, Rest:21>
6664     5          1183 Kbyte/sec ==>  9.464128 Mbit/sec
Current Time:    6.00 sec   Event#: <Insert:2301, Dequeue:2301, Rest:21>
6664     6          1183 Kbyte/sec ==>  9.464128 Mbit/sec
Current Time:    7.00 sec   Event#: <Insert:2300, Dequeue:2300, Rest:21>
6664     7          1183 Kbyte/sec ==>  9.464128 Mbit/sec
Current Time:    8.00 sec   Event#: <Insert:2299, Dequeue:2299, Rest:21>
6664     8          1183 Kbyte/sec ==>  9.464128 Mbit/sec
Current Time:    9.00 sec   Event#: <Insert:2299, Dequeue:2299, Rest:21>
6664     9          1183 Kbyte/sec ==>  9.464128 Mbit/sec
Current Time:   10.00 sec   Event#: <Insert:2300, Dequeue:2300, Rest:21>
6664     10         1183 Kbyte/sec ==>  9.464128 Mbit/sec
Current Time:   11.00 sec   Event#: <Insert:2300, Dequeue:2300, Rest:21>
6664     11         1183 Kbyte/sec ==>  9.464128 Mbit/sec
Current Time:   12.00 sec   Event#: <Insert:2298, Dequeue:2298, Rest:21>
6664     12         1183 Kbyte/sec ==>  9.464128 Mbit/sec
```

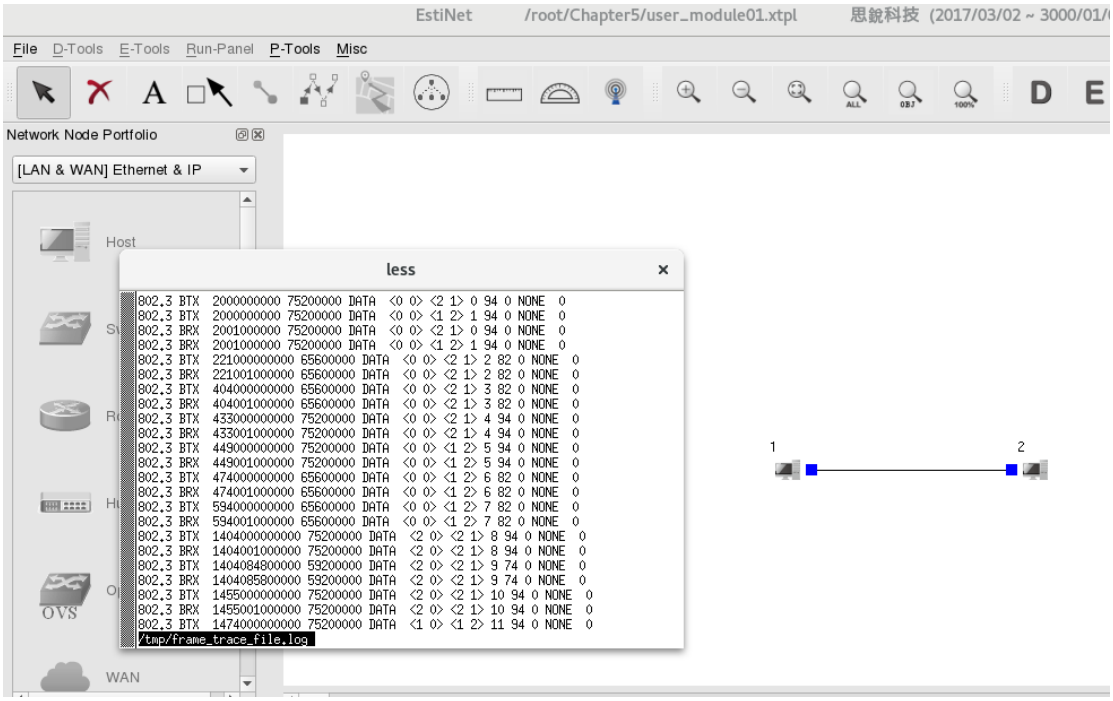
模拟结束后，会在$ESTINET_WORKDIR 中，存下模拟结果 frame_trace_file 档案，如下图所示，ESTINET_WORKDIR 在 sim 目录中，因此 frame_trace_file 文件也就存在这里的.for_se_direct_access 了(平常透过 GUI 会存在 results 目录中，但关掉 ipc 的方式会存在 ESTINET_WORKDIR 中)。
所谓的 frame_trace_file 文件，在仿真器中，主要是记录 mac 层中的封包的活动记录。

```
[root@localhost user_module01.sim]# ls
application_setting          networking_setting  user_module01.node_type_and_virtualization
interface_and_medium_setting  particular_field    user_module01.run
[root@localhost user_module01.sim]# cd .for_se_direct_access/
[root@localhost .for_se_direct_access]# ls
per_node                                user_module01.frame_trace_file          user_module01.obstacle
runtime_fatal_error_message_log         user_module01.if_and_medium_conf        user_module01.run
user_module01.application               user_module01.moving_path               user_module01.traffic_light_log
user_module01.car_moving_path_log       user_module01.node_type_and_virtualization
```

frame_trace_file 檔在 GUI 中，可以用 view frame trace file 的方式来读取，如下图所示，按上面标题栏的 P_Tools→View the Frame Transmission and Reception Log→Open the Frame Trace File，选取 frame_trace_file 档案

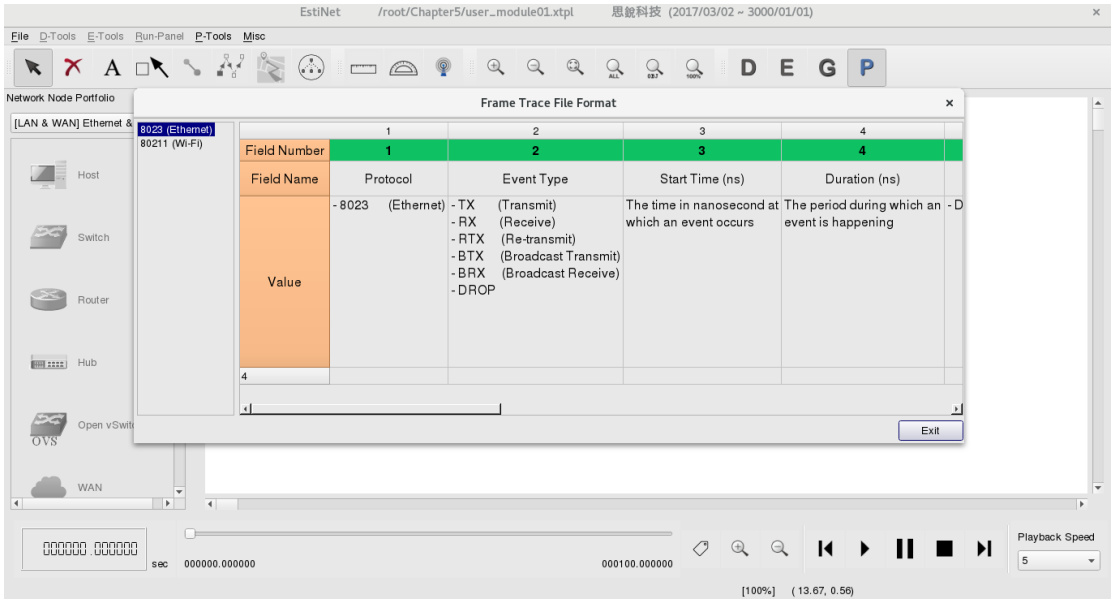


就会看到 frame_trace_file 的资料



至于 frame_trace_file 上每一行中的每一栏所代表的意义，可以看 GUI 中，按上面标题栏的 P_Tools→View the Frame Transmission and Reception Log/Show the Frame Trace File's Format

按下后可以看到每一栏所代表的意义。



但如果在不想开 GUI 的情况下看 frame_trace_file 檔，也可以使用 estinet_printftr 工具来检视或汇出。

estinet_printftr /root/Chapter5/user_module01.sim/.for_se_direct_access/user_module01.frame_trace_file

```
[root@localhost Chapter5]# estinet_printftr /root/Chapter5/user_module01.sim/.for_se_direct_access/user_module01.frame_trac
e_file
```

```
802.3 TX  99991082400000 1214400000 DATA  <1 2> <1 2> 120132 1518 0 NONE  0
802.3 RX  99991083400000 1214400000 DATA  <1 2> <1 2> 120132 1518 0 NONE  0
802.3 TX  99992306400000 1214400000 DATA  <1 2> <1 2> 120133 1518 0 NONE  0
802.3 RX  99992307400000 1214400000 DATA  <1 2> <1 2> 120133 1518 0 NONE  0
802.3 TX  99993521800000 56000000 DATA  <2 1> <2 1> 120191 70 0 NONE  0
802.3 RX  99993522800000 56000000 DATA  <2 1> <2 1> 120191 70 0 NONE  0
802.3 TX  99993530400000 1214400000 DATA  <1 2> <1 2> 120134 1518 0 NONE  0
802.3 RX  99993531400000 1214400000 DATA  <1 2> <1 2> 120134 1518 0 NONE  0
802.3 TX  99994754400000 1214400000 DATA  <1 2> <1 2> 120135 1518 0 NONE  0
802.3 RX  99994755400000 1214400000 DATA  <1 2> <1 2> 120135 1518 0 NONE  0
802.3 TX  99995969800000 56000000 DATA  <2 1> <2 1> 120192 70 0 NONE  0
802.3 RX  99995970800000 56000000 DATA  <2 1> <2 1> 120192 70 0 NONE  0
802.3 TX  99995978400000 1214400000 DATA  <1 2> <1 2> 120136 1518 0 NONE  0
802.3 RX  99995979400000 1214400000 DATA  <1 2> <1 2> 120136 1518 0 NONE  0
802.3 TX  99997202400000 1214400000 DATA  <1 2> <1 2> 120137 1518 0 NONE  0
802.3 RX  99997203400000 1214400000 DATA  <1 2> <1 2> 120137 1518 0 NONE  0
802.3 TX  99998417800000 56000000 DATA  <2 1> <2 1> 120193 70 0 NONE  0
802.3 RX  99998418800000 56000000 DATA  <2 1> <2 1> 120193 70 0 NONE  0
802.3 TX  99998426400000 1214400000 DATA  <1 2> <1 2> 120138 1518 0 NONE  0
802.3 RX  99998427400000 1214400000 DATA  <1 2> <1 2> 120138 1518 0 NONE  0
[root@localhost Chapter5]#
```

第六章：TIMER

本章重點：
(1)、TIMER 練習
(2)、TIMER 用法說明
(3)、GETCURRENTTIME、GETNODETIME、SEC_TO_TICK、MILLI_TO_TICK、BASE_OBJTYPE、POINTER_TO_MEMBER 等 API 的說明

下载练习：

Chapter6.tar.bz2

Timer 在仿真中是一个重要对象，尤其是仿真时，需要控制一些特定函数何时被触发、启动，可以说是一个重要对象，这个对象中，提供了像 **init()**、**start()**、**cancel()**、**expire()**等函数来控制，以下用一个简单的范例来说明：

■ 练习 6-1

这个范例中，一样用 user_module01 的拓扑档(user_module01.xtpl)。

接着在 user_module01.h 文件是宣告了一个 timer 对象为 myTimer，timerObj 这个是仿真器已经定义好的对象，只要 include timer.h 即可使用。
在 public 区块内宣告一个 timeout 的函式，这个函式是希望在 timer 被触发时，可以被启动执行的函式。

user_module01.h 档修改部分如下红字所示：

```
#ifndef __user_module_01_h__
#define __user_module_01_h__

#include <event.h>
#include <object.h>
#include <timer.h>

class UserModule01 : public NslObject {
```

```
private:
        int             Number;
        char            *String;
        timerObj        myTimer;

public:

        UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name);
        ~UserModule01();

        int             init();
        int             command(int argc, const char *argv[]);
        int             recv(ePacket_ *pkt);
        int             send(ePacket_ *pkt);
        void            timeout();
};

#endif    /* __user_module_01_h__ */
```

接着在 user_module01.cc 檔中，定义这个 timeout 的函式被呼叫到时，可以打印一些信息如目前的虚拟时间，以及每个 node 上的虚拟时间。**GetCurrentTime** 是用来取得仿真器所使用的虚拟时间，单位是 1 个 tick 为 1 picoosecond(10^{-12})。而在仿真器中，每个 node 上还有自己的时间，使用 **GetNodeCurrentTime**()这个 API 来取得。

另外在 init()函式，要加入 myTimer 对象的设定。
首先先宣告两个变量，first_timeout_in_tick 是第一次执行的时间点；另一个 subsequent_timeout_in_tick 是第一次执行后每次 timeout 的时间间隔。

user_module01.cc 檔中修改部分如下红字所示：

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>

MODULE_GENERATOR(UserModule01);

UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name)
        : NslObject(type, id, pl, name) {
```

```
        vBind_int("myNumber", &Number);
        vBind_char_str("myString", &String);
        REG_VAR("shared-number", &Number);
}

UserModule01::~UserModule01(){}

int UserModule01::init() {
        u_int64_t   first_timeout_in_tick;
        u_int64_t   subsequent_timeout_in_tick;
        BASE_OBJTYPE(mem_func);

        SEC_TO_TICK(first_timeout_in_tick, 3);
        MILLI_TO_TICK(subsequent_timeout_in_tick, 500);

        mem_func = POINTER_TO_MEMBER(UserModule01, timeout);
        myTimer.setCallOutObj(this, mem_func);
        myTimer.start(first_timeout_in_tick, subsequent_timeout_in_tick);

        return(NslObject::init());
}

int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        return(NslObject::send(pkt));
}

void UserModule01::timeout() {

        printf("\e[1;33;40mExercise 6-1: Timeout (%llu) (%llu)\e[m\n",
                        GetCurrentTime(), GetNodeCurrentTime(get_nid()));
```

```
        return;
}
```

BASE_OBJTYPE(mem_func);
这行则是宣告一个基本对象的指针，等一下用来指向某一个模块的某个函式。

SEC_TO_TICK(first_timeout_in_tick, 3);
这个 SEC_TO_TICK 是可以把秒数转为 tick 的单位，例如此例中的 3 秒，转为 3000000000000 tick，带入到 first_timeout_in_tick 这个变数中。

MILLI_TO_TICK(subsequent_timeout_in_tick, 500);
这个 MILLI_TO_TICK 的 API 与 SEC_TO_TICK 类似，但是是将 MILLISECOND 的单位，转为 tick 单位，例如此例中的 500 millisecond (即 0.5 秒)，转为 500000000000tick，将此值带入到 subsequent_timeout_in_tick 这个变数中。

mem_func = POINTER_TO_MEMBER(UserModule01, timeout);
接着让 mem_func 这个基本对象使用 POINTER_TO_MEMBER 这个 API，来指向 UserModule01 模块中的 timeout 的位置。

myTimer.setCallOutObj(this, mem_func);
接着设定 myTimer 时间到时，要呼叫这个 mem_func 所指到的函式。

myTimer.start(first_timeout_in_tick, subsequent_timeout_in_tick);
最后设定 myTimer 启动的时间以及执行后再重复执行的时间循环。第一个参数是第一次启动的时间点；第二个参数是重复执行的时间单位。

因此可以知道，这个 timer 在 3 秒会被启动，接着每 0.5 秒重复被执行，如下图所示。

执行结果如下：

```
net.ipv6.conf.eth0.disable_ipv6 = 0
net.ipv6.conf.eth0.autoconf = 0
Current Time:    0.00 sec   Event#: <Insert:23, Dequeue:5, Rest:18>
current ticks= 100100000, run "1 sh init_daemon.sh"
current ticks= 100200000, run "2 sh init_daemon.sh"
Current Time:    1.00 sec   Event#: <Insert:1048, Dequeue:1048, Rest:18>
Current Time:    2.00 sec   Event#: <Insert:1053, Dequeue:1053, Rest:18>
Current Time:    3.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-1: Timeout (3000000000000) (3000000004383)
Exercise 6-1: Timeout (3000000000000) (3000000003677)
Exercise 6-1: Timeout (3500000000000) (3500000004383)
Exercise 6-1: Timeout (3500000000000) (3500000003677)
Current Time:    4.00 sec   Event#: <Insert:1043, Dequeue:1043, Rest:18>
Exercise 6-1: Timeout (4000000000000) (4000000004383)
Exercise 6-1: Timeout (4000000000000) (4000000003677)
Exercise 6-1: Timeout (4500000000000) (4500000004383)
Exercise 6-1: Timeout (4500000000000) (4500000003677)
Current Time:    5.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-1: Timeout (5000000000000) (5000000004383)
Exercise 6-1: Timeout (5000000000000) (5000000003677)
Exercise 6-1: Timeout (5500000000000) (5500000004383)
Exercise 6-1: Timeout (5500000000000) (5500000003677)
Current Time:    6.00 sec   Event#: <Insert:1045, Dequeue:1045, Rest:18>
Exercise 6-1: Timeout (6000000000000) (6000000004383)
Exercise 6-1: Timeout (6000000000000) (6000000003677)
Exercise 6-1: Timeout (6500000000000) (6500000004383)
Exercise 6-1: Timeout (6500000000000) (6500000003677)
Current Time:    7.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-1: Timeout (7000000000000) (7000000004383)
Exercise 6-1: Timeout (7000000000000) (7000000003677)
Exercise 6-1: Timeout (7500000000000) (7500000004383)
```

■ 练习 6-2：延伸练习 timer

请使用者练习加入一个 timer 呼叫函式 timeout，第 5 秒被启动，之后每隔 2 秒执行，参考答案如下所示：

user_module01.cc 檔中修改部分如下红字所示：

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>

MODULE_GENERATOR(UserModule01);

UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct iflist* ifl, const char
*name): NslObject(type, id, ifl, name) {
        vBind_int("myNumber", &Number);
        vBind_char_str("myString", &String);
        REG_VAR("shared-number", &Number);
```

```
}

UserModule01::~UserModule01(){}

int UserModule01::init() {
        u_int64_t    first_timeout_in_tick;
        u_int64_t    subsequent_timeout_in_tick;
        BASE_OBJTYPE(mem_func);

        SEC_TO_TICK(first_timeout_in_tick, 5);
        SEC_TO_TICK(subsequent_timeout_in_tick, 2);

        mem_func = POINTER_TO_MEMBER(UserModule01, timeout);
        myTimer.setCallOutObj(this, mem_func);
        myTimer.start(first_timeout_in_tick, subsequent_timeout_in_tick);

        return(NslObject::init());
}

int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        return(NslObject::send(pkt));
}

void UserModule01::timeout() {

        printf("\e[1;33;40mExercise 6-2: Timeout (%llu) (%llu)\e[m\n",
                        GetCurrentTime(), GetNodeCurrentTime(get_nid()));
        return;
}
```

```
current ticks= 100200000, run "2 sh init_daemon.sh"
Current Time:     1.00 sec    Event#: <Insert:1048, Dequeue:1048, Rest:18>
Current Time:     2.00 sec    Event#: <Insert:1052, Dequeue:1052, Rest:18>
Current Time:     3.00 sec    Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:     4.00 sec    Event#: <Insert:1043, Dequeue:1043, Rest:18>
Current Time:     5.00 sec    Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-2: Timeout (5000000000000) (5000000004383)
Exercise 6-2: Timeout (5000000000000) (5000000003677)
Current Time:     6.00 sec    Event#: <Insert:1045, Dequeue:1045, Rest:18>
Current Time:     7.00 sec    Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-2: Timeout (7000000000000) (7000000004383)
Exercise 6-2: Timeout (7000000000000) (7000000003677)
Current Time:     8.00 sec    Event#: <Insert:1043, Dequeue:1043, Rest:18>
Current Time:     9.00 sec    Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-2: Timeout (9000000000000) (9000000004383)
Exercise 6-2: Timeout (9000000000000) (9000000003677)
Current Time:    10.00 sec    Event#: <Insert:1045, Dequeue:1045, Rest:18>
Current Time:    11.00 sec    Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-2: Timeout (11000000000000) (11000000004383)
Exercise 6-2: Timeout (11000000000000) (11000000003677)
Current Time:    12.00 sec    Event#: <Insert:1043, Dequeue:1043, Rest:18>
Current Time:    13.00 sec    Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-2: Timeout (13000000000000) (13000000004383)
Exercise 6-2: Timeout (13000000000000) (13000000003677)
Current Time:    14.00 sec    Event#: <Insert:1043, Dequeue:1043, Rest:18>
Current Time:    15.00 sec    Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-2: Timeout (15000000000000) (15000000004383)
Exercise 6-2: Timeout (15000000000000) (15000000003677)
```

■ 练习 6-3：延伸练习 timer

接着练习在 mytimer 呼叫函式 timeout，第 3 秒被启动，之后每隔 0.5 秒执行，第 5 秒暂停，第 7 秒恢复，第 9 秒被取消
这需要用到 timer 对象中的 **cancel()**、**pause()**、以及 **resume()**

首先还需要在 user_module01.h 档中多宣告一个 timer 跟一个函式，让 timer 被暂停后，由另一个 timer 帮忙恢复。如下面蓝字所示。

```
#ifndef __user_module_01_h__
#define __user_module_01_h__

#include <event.h>
#include <object.h>
#include <timer.h>

class UserModule01 : public NslObject {

 private:
```

```
        int             Number;
        char            *String;
        timerObj        myTimer, resumeTimer;

    public:

        UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name);
        ~UserModule01();

        int         init();
        int         command(int argc, const char *argv[]);
        int         recv(ePacket_ *pkt);
        int         send(ePacket_ *pkt);
        void        timeout();
        void        resumetimer();
};

#endif    /* __user_module_01_h__ */
```

接着修改 user_module01.cc，如下红字所示：

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>

MODULE_GENERATOR(UserModule01);

UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char
*name)
    : NslObject(type, id, pl, name) {}

UserModule01::~UserModule01(){}


int UserModule01::init() {
    u_int64_t   first_timeout_in_tick;
    u_int64_t   subsequent_timeout_in_tick;
    BASE_OBJTYPE(mem_func);
```

```
        SEC_TO_TICK(first_timeout_in_tick, 3);
        MILLI_TO_TICK(subsequent_timeout_in_tick, 500);

        mem_func = POINTER_TO_MEMBER(UserModule01, timeout);
        myTimer.setCallOutObj(this, mem_func);
        myTimer.start(first_timeout_in_tick, subsequent_timeout_in_tick);

        u_int64_t   resume_timeout_in_tick;
        BASE_OBJTYPE(mem_func2);
        mem_func2 = POINTER_TO_MEMBER(UserModule01, resumetimer);
        SEC_TO_TICK(resume_timeout_in_tick, 7);
        resumeTimer.setCallOutObj(this, mem_func2);
        resumeTimer.start(resume_timeout_in_tick, 0);

        return(NslObject::init());
}

int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        return(NslObject::send(pkt));
}

void UserModule01::timeout() {

        printf("\e[1;33;40mExercise 6-3: Timeout (%llu) (%llu) Get time timer expired
time: %llu \e[m\n", GetCurrentTime(), GetNodeCurrentTime(get_nid()), myTimer.expire());

        if(GetCurrentTime() == 9000000000000)
        {
                myTimer.cancel();
```

```
        }
        else if(GetCurrentTime() == 5000000000000)
        {
                myTimer.pause();
        }

        return;
}

void UserModule01::resumetimer() {

        myTimer.resume();
        return;
}
```

修改 timeout()函式，在得到模拟时间为 5 秒时，将 mytimer 暂停，而在模拟时间 9 秒时，将 mytimer 取消。

打印中加入 myTimer()的 expire()函式，可以看到下次 myTimer 被启动的时间。

接着定义 resumetimer()函式则是用来恢复 mytimer

在 init 下多加上蓝色的区段，定义在第 7 秒由 resumeTimer 启动 resumetimer()函式来恢复 myTimer()

值得注意的是 resumeTimer 后面的 interval 是带 0，表示这个 Timer 只执行 1 次即可。

可以从下面执行结果中，看到第 5 秒时打印完时间后，接着就被暂停了，等到第 7 秒时，由 resumeTimer 恢复 myTimer 后，又开始呼叫 timeout 函式打印，直到第 9 秒 timeout 函式时，打印后直接取消 myTimer。

```
current ticks= 100200000, run "2 sh init_daemon.sh"
Current Time:    1.00 sec   Event#: <Insert:1048, Dequeue:1048, Rest:18>
Current Time:    2.00 sec   Event#: <Insert:1052, Dequeue:1052, Rest:18>
Current Time:    3.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Exercise 6-3: Timeout (3000000000000) (3000000004383) Get time timer expired time: 3500000000000
Exercise 6-3: Timeout (3000000000000) (3000000003677) Get time timer expired time: 3500000000000
Exercise 6-3: Timeout (3500000000000) (3500000004383) Get time timer expired time: 4000000000000
Exercise 6-3: Timeout (3500000000000) (3500000003677) Get time timer expired time: 4000000000000
Current Time:    4.00 sec   Event#: <Insert:1043, Dequeue:1043, Rest:18>
Exercise 6-3: Timeout (4000000000000) (4000000004383) Get time timer expired time: 4500000000000
Exercise 6-3: Timeout (4000000000000) (4000000003677) Get time timer expired time: 4500000000000
Exercise 6-3: Timeout (4500000000000) (4500000004383) Get time timer expired time: 5000000000000
Exercise 6-3: Timeout (4500000000000) (4500000003677) Get time timer expired time: 5000000000000
Current Time:    5.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-3: Timeout (5000000000000) (5000000004383) Get time timer expired time: 5500000000000
Exercise 6-3: Timeout (5000000000000) (5000000003677) Get time timer expired time: 5500000000000
Current Time:    6.00 sec   Event#: <Insert:1045, Dequeue:1045, Rest:18>
Current Time:    7.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-3: Timeout (7500000000000) (7500000004383) Get time timer expired time: 8000000000000
Exercise 6-3: Timeout (7500000000000) (7500000003677) Get time timer expired time: 8000000000000
Current Time:    8.00 sec   Event#: <Insert:1043, Dequeue:1043, Rest:18>
Exercise 6-3: Timeout (8000000000000) (8000000004383) Get time timer expired time: 8500000000000
Exercise 6-3: Timeout (8000000000000) (8000000003677) Get time timer expired time: 8500000000000
Exercise 6-3: Timeout (8500000000000) (8500000004383) Get time timer expired time: 9000000000000
Exercise 6-3: Timeout (8500000000000) (8500000003677) Get time timer expired time: 9000000000000
Current Time:    9.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-3: Timeout (9000000000000) (9000000004383) Get time timer expired time: 9500000000000
Exercise 6-3: Timeout (9000000000000) (9000000003677) Get time timer expired time: 9500000000000
Current Time:   10.00 sec   Event#: <Insert:1045, Dequeue:1045, Rest:18>
Current Time:   11.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
Current Time:   12.00 sec   Event#: <Insert:1043, Dequeue:1043, Rest:18>
Current Time:   13.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
Current Time:   14.00 sec   Event#: <Insert:1043, Dequeue:1043, Rest:18>
Current Time:   15.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
Current Time:   16.00 sec   Event#: <Insert:1043, Dequeue:1043, Rest:18>
Current Time:   17.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
Current Time:   18.00 sec   Event#: <Insert:1043, Dequeue:1043, Rest:18>
Current Time:   19.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
```

第七章：EVENT

本章重點：
(1)、EVENT 練習
(2)、EVENT 用法說明

下载练习：


Chapter7.tar.bz2

仿真器中，也可以使用 EVENT 的数据结构来触发事件，在仿真器中其实像 PACKET 其实都是用 EVENT 去包装起来的。每个封包都是一个 EVENT。
而 timer 其实也是 event 结构的延伸，timer 的本质上也是一个 event，但多了许多为 timer 设计使用的函式。

本章要学会怎么自己创造一个 EVENT，来触发一个函式执行，之后再将此 EVENT 空间释放

下面用一个例子说明 event 的用法，我们创造一个 event 并宣告一个 print_event()的函式，设定在第一秒时让仿真器中的 event 去触发此函式来打印。

■ 练习 7-1
同样使用第一章的范例拓扑(user_module01.xtpl)。

修改 user_module01.h，如下红字所示：

```
#ifndef __user_module_01_h__
#define __user_module_01_h__

#include <event.h>
#include <object.h>

class UserModule01 : public NslObject {

  private:
```

```
        int             Number;
        char            *String;

 public:

        UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name);
        ~UserModule01();

        int         init();
        int         command(int argc, const char *argv[]);
        int         recv(ePacket_ *pkt);
        int         send(ePacket_ *pkt);
        void        print_event(Event_ *ep);
};

#endif   /* __user_module_01_h__ */
```

修改 user_module01.cc，如下红字所示：

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>

MODULE_GENERATOR(UserModule01);

UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name)
        : NslObject(type, id, pl, name) {
        vBind_int("myNumber", &Number);
        vBind_char_str("myString", &String);
        REG_VAR("shared-number", &Number);
}

UserModule01::~UserModule01(){}

int UserModule01::init() {

        Event_ *ep;
        ep = createEvent();
```

```
        u_int64_t expire;
        BASE_OBJTYPE(mem_func3);

        SEC_TO_TICK(expire, 1);
        mem_func3 = POINTER_TO_MEMBER(UserModule01, print_event);

        setObjEvent(ep,
                        expire,
                        0,
                        this,
                        mem_func3,
                        (void *)0
                    );

        return(NslObject::init());

}

int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        return(NslObject::send(pkt));
}

void UserModule01::print_event(Event_ *ep) {
        printf("\e[1;33;40mExercise 7-1: Timeout (%llu) (%llu)\e[m\n",
                        GetCurrentTime(), GetNodeCurrentTime(get_nid()));
        freeEvent(ep);
        return;
}
```

createEvent()用来创造一个新的 event。

而 freeEvent()函式则是用来释放这个 event 的空间。

setObjEvent 则是设定此 event 的细节，第一个参数是 event 的指标，用来指向某个 event，第二个参数是执行的时间，第三个参数是 interval，如果是 0 表示只执行一次，第四个参数是指执行的对象，而第五个函式是指定处理此 event 的函式或对象，第六个参数是一些额外的 data，像封包的数据就会带入于此处

下图就是执行结果，我们可以看到，在第一秒时让仿真器中的 event 去触发此函式来打印。因为第三个参数带入 0，因此只执行一次。

```
current ticks= 100100000, run "1 sh init_daemon.sh"
current ticks= 100200000, run "2 sh init_daemon.sh"
Exercise 7-1: Timeout (1000000000000) (1000000004383)
Current Time:    1.00 sec   Event#: <Insert:1056, Dequeue:1055, Rest:19>
Exercise 7-1: Timeout (1000000000000) (1000000003677)
Current Time:    2.00 sec   Event#: <Insert:1046, Dequeue:1047, Rest:18>
Current Time:    3.00 sec   Event#: <Insert:1043, Dequeue:1043, Rest:18>
Current Time:    4.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:    5.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:    6.00 sec   Event#: <Insert:1044, Dequeue:1044, Rest:18>
Current Time:    7.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:    8.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:    9.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:   10.00 sec   Event#: <Insert:1044, Dequeue:1044, Rest:18>
Current Time:   11.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:   12.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:   13.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:   14.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:   15.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:   16.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:   17.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
```

- 练习 7-2：
  若希望这个 EVENT，周期性的执行，可以使用下面修改 user_module01.cc 后的程序代码，如红字所示：

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>

MODULE_GENERATOR(UserModule01);

UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct iflist* ifl, const char
*name): NslObject(type, id, ifl, name) {
```

```
        vBind_int("myNumber", &Number);
        vBind_char_str("myString", &String);
        REG_VAR("shared-number", &Number);
}

UserModule01::~UserModule01(){}

int UserModule01::init() {

        Event_ *ep;
        ep = createEvent();

        u_int64_t expire;
        BASE_OBJTYPE(mem_func3);

        SEC_TO_TICK(expire, 1);
        mem_func3 = POINTER_TO_MEMBER(UserModule01, print_event);

        setObjEvent(ep,
                        expire,
                        expire,
                        this,
                        mem_func3,
                        (void *)0
                    );

        return(NslObject::init());

}

int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}
```

```
int UserModule01::send(ePacket_ *pkt) {
        return(NslObject::send(pkt));
}

void UserModule01::print_event(Event_ *ep) {
        printf("\e[1;33;40mExercise 7-2: Timeout (%llu) (%llu)\e[m\n",
                        GetCurrentTime(), GetNodeCurrentTime(get_nid()));
        //freeEvent(ep);
        setEventReuse(ep);
        return;
}
```

执行结果如下：

```
current ticks= 100200000, run "2 sh init_daemon.sh"
Exercise 7-2: Timeout (1000000000000) (1000000004383)
Current Time:     1.00 sec    Event#: <Insert:1057, Dequeue:1055, Rest:20>
Exercise 7-2: Timeout (1000000000000) (1000000003677)
Exercise 7-2: Timeout (2000000000000) (2000000003677)
Exercise 7-2: Timeout (2000000000000) (2000000004383)
Current Time:     2.00 sec    Event#: <Insert:1054, Dequeue:1054, Rest:20>
Exercise 7-2: Timeout (3000000000000) (3000000003677)
Exercise 7-2: Timeout (3000000000000) (3000000004383)
Current Time:     3.00 sec    Event#: <Insert:1046, Dequeue:1046, Rest:20>
Current Time:     4.00 sec    Event#: <Insert:1037, Dequeue:1037, Rest:20>
Exercise 7-2: Timeout (4000000000000) (4000000003677)
Exercise 7-2: Timeout (4000000000000) (4000000004383)
Exercise 7-2: Timeout (5000000000000) (5000000004383)
Current Time:     5.00 sec    Event#: <Insert:1049, Dequeue:1049, Rest:20>
Exercise 7-2: Timeout (5000000000000) (5000000003677)
Exercise 7-2: Timeout (6000000000000) (6000000003677)
Exercise 7-2: Timeout (6000000000000) (6000000004383)
Current Time:     6.00 sec    Event#: <Insert:1047, Dequeue:1047, Rest:20>
Exercise 7-2: Timeout (7000000000000) (7000000003677)
Exercise 7-2: Timeout (7000000000000) (7000000004383)
Current Time:     7.00 sec    Event#: <Insert:1045, Dequeue:1045, Rest:20>
Current Time:     8.00 sec    Event#: <Insert:1037, Dequeue:1037, Rest:20>
Exercise 7-2: Timeout (8000000000000) (8000000003677)
Exercise 7-2: Timeout (8000000000000) (8000000004383)
Exercise 7-2: Timeout (9000000000000) (9000000004383)
Current Time:     9.00 sec    Event#: <Insert:1049, Dequeue:1049, Rest:20>
Exercise 7-2: Timeout (9000000000000) (9000000003677)
```

除了要加入第三个参数外，另外还要搭配 setEventReuse 这个 API。因此若需要周期性的执行，不妨选择使用 timer 对象较为理想。

跟封包有关的 event 结构，在下一个章节进行介绍。

第八章：PACKET

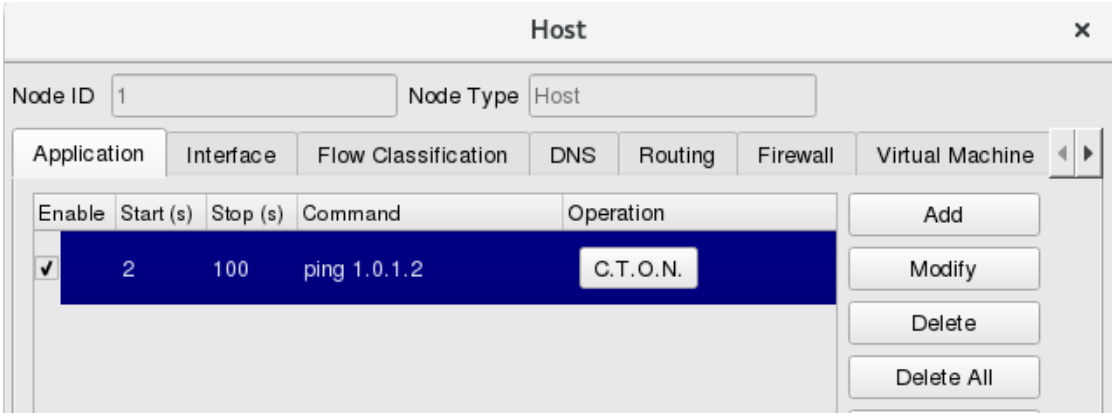
本章重點：
(1)、PACKET 練習
(2)、PACKET 用法說明

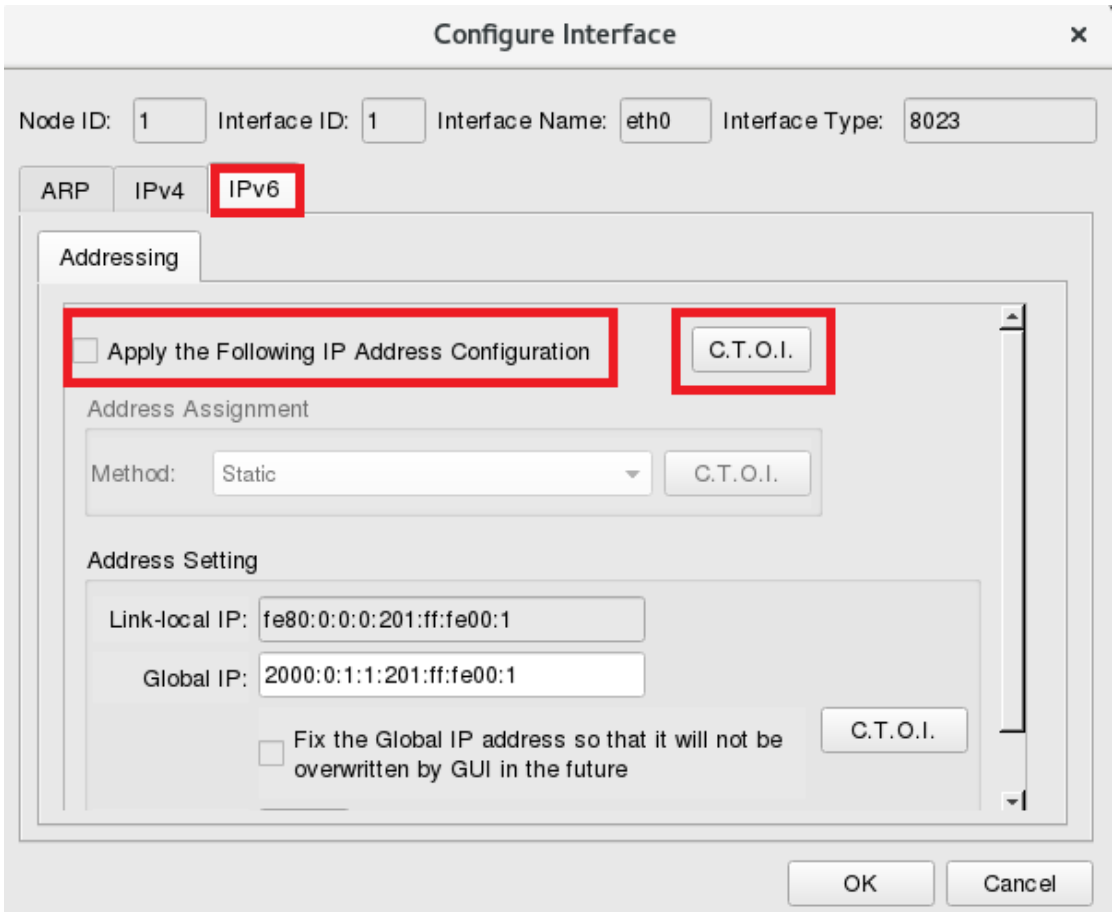
下载练习：

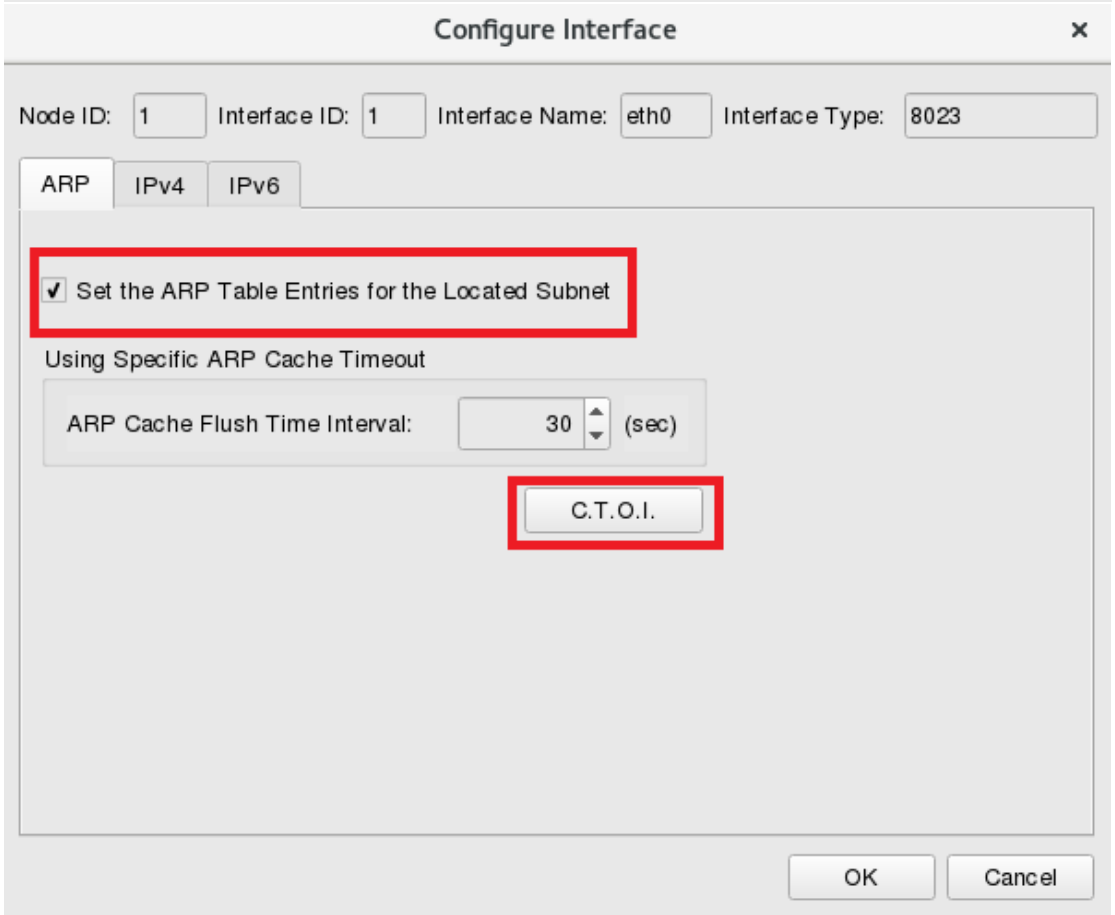
Chapter8.tar.bz2

在网络仿真中，最重要的是封包的处理，在仿真器中，主要是有一个处理的对象，一开始会由 Interface 模块，将封包包装在 Packet 结构中，再循序经由每一个模块中的 send()进行处理(模块接收与传送的架构，可以查看第四章中说明)。实际上 PACKET 对象使用 EVENT 的结构，因此也具有 EVENT 的特性。以下例来使用及说明这个 packet 对象结构。

■ 练习 8-1：
同样使用第一章的范例拓扑(user_module01.xtpl)，并加入通讯流如下：
首先使用 Host1 ping Host2，在 Host1 中加入 ping 1.0.1.2。在 Host1 与 Host2 上都关闭 ARP 与 IPv6 界面。




Configure Interface
Node ID: 1
Interface ID: 1
Interface Name: eth0
Interface Type: 8023
ARP
IPv4
IPv6
Addressing
Apply the Following IP Address Configuration
C.T.O.I.
Address Assignment
Method: Static
C.T.O.I.
Address Setting
Link-local IP: fe80:0:0:0:201:ff:fe00:1
Global IP: 2000:0:1:1:201:ff:fe00:1
Fix the Global IP address so that it will not be overwritten by GUI in the future
C.T.O.I.
OK
Cancel


Configure Interface
Node ID: 1
Interface ID: 1
Interface Name: eth0
Interface Type: 8023
ARP
IPv4
IPv6
Set the ARP Table Entries for the Located Subnet
Using Specific ARP Cache Timeout
ARP Cache Flush Time Interval: 30 (sec)
C.T.O.I.
OK
Cancel

接着在 user_module01.h 档中宣告一个函式 pkt_delay，如下红字所示：

```
#ifndef __user_module_01_h__
#define __user_module_01_h__

#include <event.h>
#include <object.h>

class UserModule01 : public NslObject {

 private:
        int             Number;
        char            *String;

 public:

        UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name);
        ~UserModule01();

        int             init();
        int             command(int argc, const char *argv[]);
        int             recv(ePacket_ *pkt);
        int             send(ePacket_ *pkt);
        void            pkt_delay(ePacket_ *pkt);
};

#endif    /* __user_module_01_h__ */
```

接着修改 user_module01.cc，修改部分如下红字所示：

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>
#include <packet.h>
#include <ethernet.h>
#include <ip.h>

MODULE_GENERATOR(UserModule01);
```

```
UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name)
        : NslObject(type, id, pl, name) {
        vBind_int("myNumber", &Number);
        vBind_char_str("myString", &String);
        REG_VAR("shared-number", &Number);
}

UserModule01::~UserModule01(){}

int UserModule01::init() {
        return(NslObject::init());
}

int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        u_int64_t    current_time_in_tick;
        u_int64_t    delay_time_in_tick;
        BASE_OBJTYPE(mem_func);

        current_time_in_tick = GetCurrentTime();
        SEC_TO_TICK(delay_time_in_tick, 1);
        mem_func = POINTER_TO_MEMBER(UserModule01, pkt_delay);
        setObjEvent(pkt, current_time_in_tick + delay_time_in_tick,
                0, this, mem_func, (void *)pkt->DataInfo_);
        return(1);

}

void UserModule01::pkt_delay(ePacket_ *pkt) {

        Packet                  *packet;
```

```
        struct ether_header     *eh;
        char                    src_mac_str[18];
        char                    dst_mac_str[18];

        struct ip               *iph;
        char                    src_ip_str[16];
        char                    dst_ip_str[16];

        if(pkt != NULL && pkt->DataInfo_ != NULL) {
                packet = (Packet *)pkt->DataInfo_;
                eh = (struct ether_header *)packet->pkt_get();

                macaddr_to_str(eh->ether_shost, src_mac_str);
                macaddr_to_str(eh->ether_dhost, dst_mac_str);

                printf("\e[1;36;40mExercise 8-1: Src Mac: %s, Dst Mac: %s\e[m\n",
                        src_mac_str, dst_mac_str);

                iph = (struct ip *)packet->pkt_sget();
                if(iph != NULL) {
                        ipv4addr_to_str(iph->ip_src, src_ip_str);
                        ipv4addr_to_str(iph->ip_dst, dst_ip_str);
                        printf("\e[1;36;40mExercise 8-1: Src IP: %s, Dst IP: %s\e[m\n",
                        src_ip_str, dst_ip_str);
                }
        }

        NslObject::send(pkt);
}
```

在 send()函式中，接收到通讯流的封包后，先不要送出去，而将这个封包中的 event 设定在 1 秒后，触发了 pkt_delay 这个函式进行处理，才送给下一个模块。

这样可以做到封包延迟的效果。也可以发现其实在模块中的封包是由 event 对象所包装。

另外，封包结构中的 DataInfo_，存放的就是通讯流封包。

而使用 **pkt_get()**，首先会取得 ether header，如下所示：

```
eh = (struct ether_header *)packet->pkt_get();
```

接着用 **macaddr_to_str** API 将 eh 中的 source mac address，跟 dest mac address 转成 str 的 API。如下所示：

```
macaddr_to_str(eh->ether_shost, src_mac_str);
macaddr_to_str(eh->ether_dhost, dst_mac_str);
```

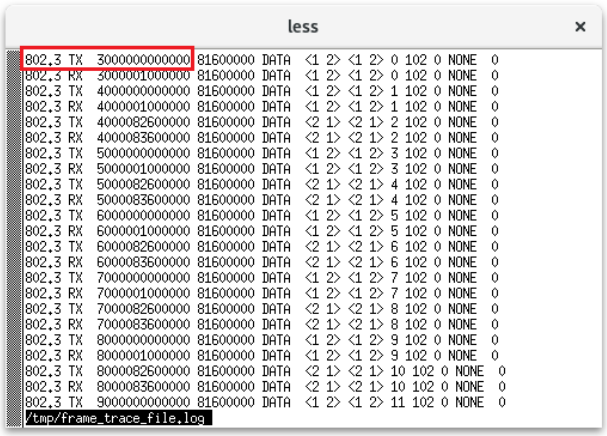
若要取得 IP HEADER，则是使用 **pkt_sget()**，如下所示：

```
iph = (struct ip *)packet->pkt_sget();
```

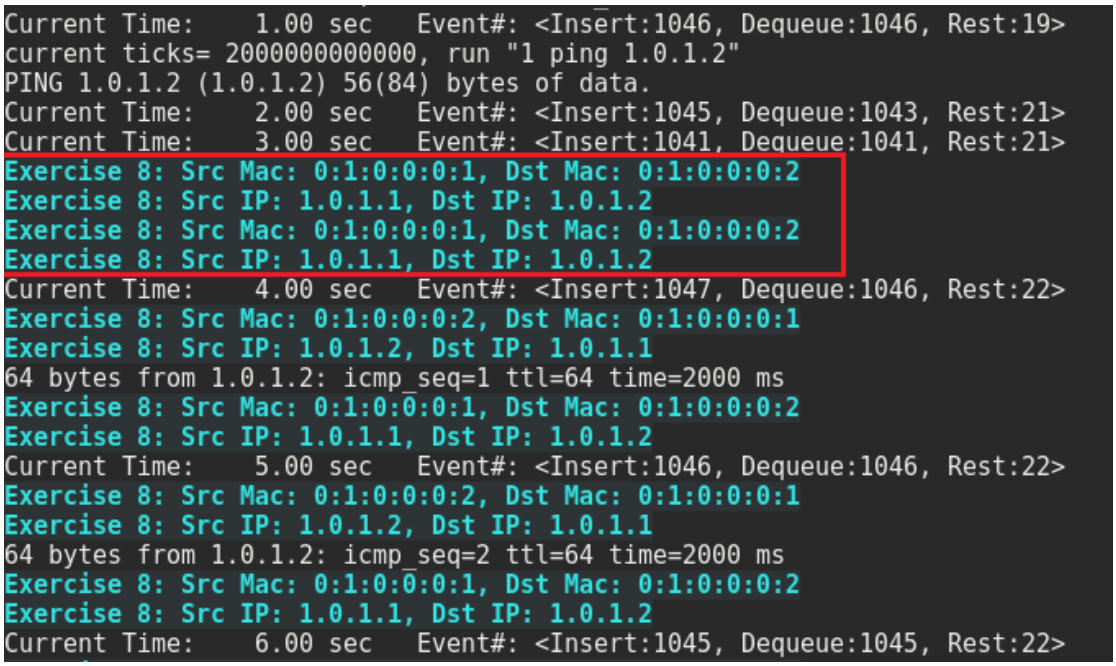
接着用 **ipv4addr_to_str** API 将 IP HEADER 中的 ip source address 以及 ip dest address 转成 str，如下所示：

```
ipv4addr_to_str(iph->ip_src, src_ip_str);
ipv4addr_to_str(iph->ip_dst, dst_ip_str);
```

完成修改后，将原始码 make 以及 make install 后执行模拟，模拟结果如下所示，可以看到原本第二秒就要送出的封包，在第三秒才有 log，达成了封包延迟的效果。



接着看一下 estinetss 的窗口，刚才使用 ipv4addr_to_string()以及 macaddr_to_str()所打印出来的信息如下图所示，开发者可以印此信息来 debug 或是查看相关信息。



■ 练习 8-2：pkt_addinfo()、pkt_getinfo()与 pkt_delinfo()

有时候在模块之间处理数据时，会希望封包有一些携带额外的信息，但封包通常有特定的格式。因此在仿真器中的封包结构，每个封包都有一个额外的 buffer，可以存放一些额外的信息，称为 PT_INFO。可以使用 **pkt_addinfo()**、**pkt_getinfo()**、**pkt_delinfo()**等 API，让封包可以携带额外的信息。

user_module01.h 修改部分，如下红字所示：

```
#ifndef __user_module_01_h__
#define __user_module_01_h__

#include <event.h>
#include <object.h>

class UserModule01 : public NslObject {

 private:
      int          Number;
      char         *String;

 public:
```

```
        UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name);
        ~UserModule01();

        int             init();
        int             command(int argc, const char *argv[]);
        int             recv(ePacket_ *pkt);
        int             send(ePacket_ *pkt);
        void            pkt_delay(ePacket_ *pkt);
};

#endif    /* __user_module_01_h__ */
```

修改 user_module01.cc，延用上一练习红色修改的部分，并加上蓝色的修改如下所示：

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>
#include <packet.h>
#include <ethernet.h>
#include <ip.h>

MODULE_GENERATOR(UserModule01);

UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name)
        : NslObject(type, id, pl, name) {
}

UserModule01::~UserModule01(){}

int UserModule01::init() {
        return(NslObject::init());
}

int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
```

```
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        u_int64_t    current_time_in_tick;
        u_int64_t    delay_time_in_tick;
        BASE_OBJTYPE(mem_func);

        current_time_in_tick = GetCurrentTime();
        SEC_TO_TICK(delay_time_in_tick, 1);
        mem_func = POINTER_TO_MEMBER(UserModule01, pkt_delay);
        setObjEvent(pkt, current_time_in_tick + delay_time_in_tick,
                0, this, mem_func, (void *)pkt->DataInfo_);
        return(1);

}

void UserModule01::pkt_delay(ePacket_ *pkt) {

        Packet                  *packet;
        struct ether_header     *eh;
        char                    src_mac_str[18];
        char                    dst_mac_str[18];

        struct ip               *iph;
        char                    src_ip_str[16];
        char                    dst_ip_str[16];

        if(pkt != NULL && pkt->DataInfo_ != NULL) {
                packet = (Packet *)pkt->DataInfo_;
                eh = (struct ether_header *)packet->pkt_get();

                macaddr_to_str(eh->ether_shost, src_mac_str);
                macaddr_to_str(eh->ether_dhost, dst_mac_str);

                printf("\e[1;36;40mExercise 8-2: Src Mac: %s, Dst Mac: %s\e[m\n",
                        src_mac_str, dst_mac_str);
```

```
            iph = (struct ip *)packet->pkt_sget();
            if(iph != NULL) {
                ipv4addr_to_str(iph->ip_src, src_ip_str);
                ipv4addr_to_str(iph->ip_dst, dst_ip_str);
                printf("\e[1;36;40mExercise 8-2: Src IP: %s, Dst IP: %s\e[m\n",
                src_ip_str, dst_ip_str);
            }
        }

        int NodeColor=random()%3;
        packet->pkt_addinfo("NodeColor", (char *)&NodeColor, sizeof(NodeColor));
        NslObject::send(pkt);
}
```

在这个范例中，User_module01 在传送封包到下一个模块前，在每一个封包上加上一个 info，是 node 的颜色，有 3 种 random 值。

- pkt_addinfo 用法：

  第一个参数是此 info 的识别值(使用者定义，最多 15 个字符)，第二个参数则是一个 c++变数，第三个参数则是此 c++变数的大小。

在 phy.cc 修改部分如下蓝字所示：

```
int phy::send(ePacket_ *pkt) {

        struct con_list             *cl;
        Packet                      *p;
        struct phyInfo              *phyinfo;

        assert(pkt&&(p=(Packet *)pkt->DataInfo_));
        if ( LinkFailFlag > 0 ) {
            freePacket(pkt);
            return(1);
        }

        int *nodecolor = (int *)p->pkt_getinfo("NodeColor");
        if(*nodecolor == 1)
```

```
            printf("\e[1;33;42mExercise 8-2: Node Color : %d\e[m\n", *nodecolor);
        else if(*nodecolor == 2)
            printf("\033[1;35;45mExercise 8-2: Node Color : %d\033[m\n", *nodecolor);
        else
            printf("\e[1;31;41mExercise 8-2: Node Color : %d\e[m\n", *nodecolor);
        p->pkt_delinfo("NodeColor");
```

在 phy 模块中，则用 pkt_getinfo()，取得此 NodeColor 信息，并针对每一个值印出不同的 ascii 颜色。

处理后，使用 pkt_delinfo 将这个 NodeColor 信息删除。

- pkt_getinfo()与 pkt_delinfo()用法：
  pkt_getinfo 带入 info 的标识符会回传此标识符的数值；而 pkt_delinfo 带入 info 的标识符会删除此笔 info 的信息。

执行结果：

```
Current Time:    1.00 sec   Event#: <Insert:1046, Dequeue:1046, Rest:19>
current ticks= 2000000000000, run "1 ping 1.0.1.2"
PING 1.0.1.2 (1.0.1.2) 56(84) bytes of data.
Current Time:    2.00 sec   Event#: <Insert:1045, Dequeue:1043, Rest:21>
Current Time:    3.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:21>
Exercise 8-2: Src Mac: 0:1:0:0:0:1, Dst Mac: 0:1:0:0:0:2
Exercise 8-2: Src IP: 1.0.1.1, Dst IP: 1.0.1.2
Exercise 8-2: Node Color : 0
Exercise 8-2: Src Mac: 0:1:0:0:0:1, Dst Mac: 0:1:0:0:0:2
Exercise 8-2: Src IP: 1.0.1.1, Dst IP: 1.0.1.2
Exercise 8-2: Node Color : 0
Current Time:    4.00 sec   Event#: <Insert:1047, Dequeue:1046, Rest:22>
Exercise 8-2: Src Mac: 0:1:0:0:0:2, Dst Mac: 0:1:0:0:0:1
Exercise 8-2: Src IP: 1.0.1.2, Dst IP: 1.0.1.1
Exercise 8-2: Node Color : 2
64 bytes from 1.0.1.2: icmp_seq=1 ttl=64 time=2000 ms
Exercise 8-2: Src Mac: 0:1:0:0:0:1, Dst Mac: 0:1:0:0:0:2
Exercise 8-2: Src IP: 1.0.1.1, Dst IP: 1.0.1.2
Exercise 8-2: Node Color : 1
Current Time:    5.00 sec   Event#: <Insert:1046, Dequeue:1046, Rest:22>
Exercise 8-2: Src Mac: 0:1:0:0:0:2, Dst Mac: 0:1:0:0:0:1
Exercise 8-2: Src IP: 1.0.1.2, Dst IP: 1.0.1.1
Exercise 8-2: Node Color : 0
64 bytes from 1.0.1.2: icmp_seq=2 ttl=64 time=2000 ms
Exercise 8-2: Src Mac: 0:1:0:0:0:1, Dst Mac: 0:1:0:0:0:2
Exercise 8-2: Src IP: 1.0.1.1, Dst IP: 1.0.1.2
Exercise 8-2: Node Color : 0
Current Time:    6.00 sec   Event#: <Insert:1045, Dequeue:1045, Rest:22>
Exercise 8-2: Src Mac: 0:1:0:0:0:2, Dst Mac: 0:1:0:0:0:1
Exercise 8-2: Src IP: 1.0.1.2, Dst IP: 1.0.1.1
Exercise 8-2: Node Color : 2
64 bytes from 1.0.1.2: icmp_seq=3 ttl=64 time=2000 ms
Exercise 8-2: Src Mac: 0:1:0:0:0:1, Dst Mac: 0:1:0:0:0:2
Exercise 8-2: Src IP: 1.0.1.1, Dst IP: 1.0.1.2
Exercise 8-2: Node Color : 2
Current Time:    7.00 sec   Event#: <Insert:1047, Dequeue:1047, Rest:22>
```

第九章：其它 API 练习

本章重點：
(1)、介紹 GetNodeLoc、GetTotalNumOfNodes、GetConFilePathAndName、getConnectNode 等 API

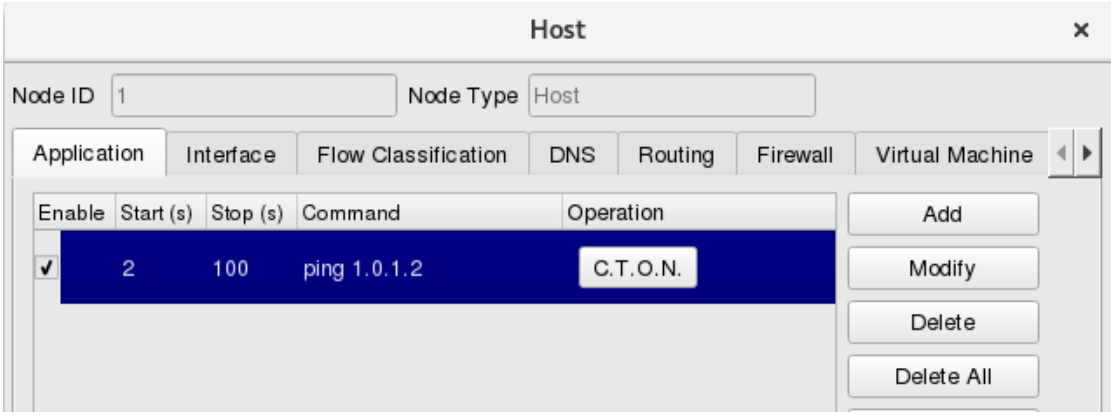
下载练习：

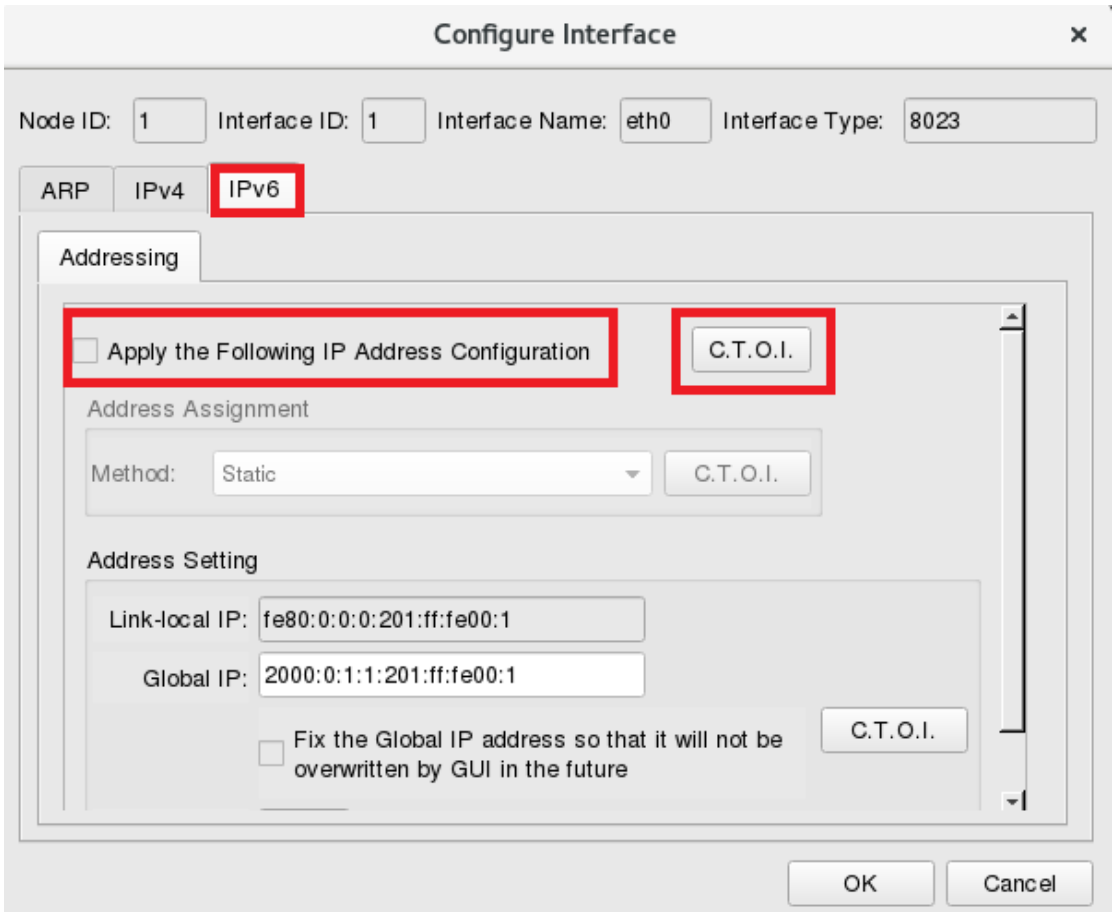
Chapter9.tar.bz2

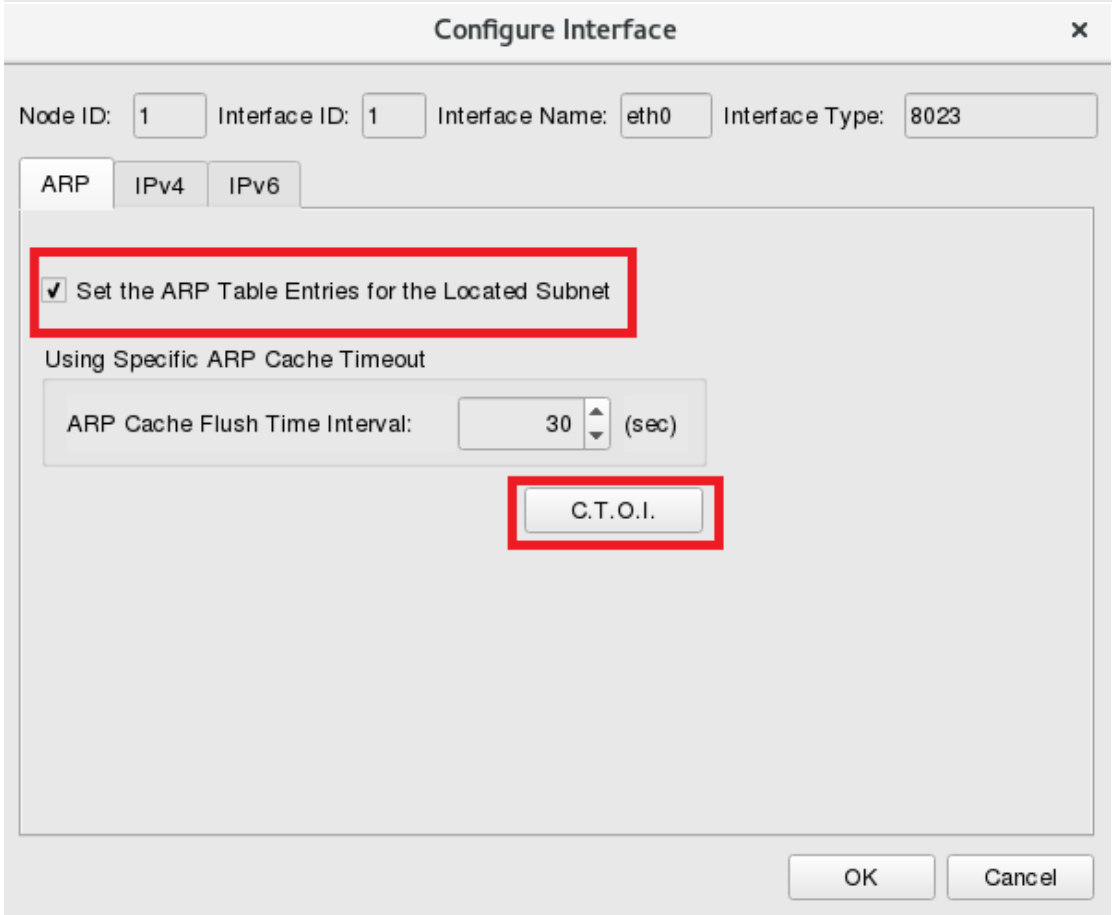
本章中介绍并使用一些常用的 API。

■ 练习 9-1

同样使用第一章的范例拓扑(user_module01.xtpl)，并加入通讯流如下：
首先使用 Host1 ping Host2，在 Host1 中加入 ping 1.0.1.2。在 Host1 与 Host2 上都关闭 ARP 与 IPv6 界面。




Configure Interface
Node ID: 1
Interface ID: 1
Interface Name: eth0
Interface Type: 8023
ARP
IPv4
IPv6
Addressing
Apply the Following IP Address Configuration
C.T.O.I.
Address Assignment
Method: Static
C.T.O.I.
Address Setting
Link-local IP: fe80:0:0:0:201:ff:fe00:1
Global IP: 2000:0:1:1:201:ff:fe00:1
Fix the Global IP address so that it will not be overwritten by GUI in the future
C.T.O.I.
OK
Cancel


Configure Interface
Node ID: 1
Interface ID: 1
Interface Name: eth0
Interface Type: 8023
ARP
IPv4
IPv6
Set the ARP Table Entries for the Located Subnet
Using Specific ARP Cache Timeout
ARP Cache Flush Time Interval: 30 (sec)
C.T.O.I.
OK
Cancel

修改 user_module01.h 如下红字所示：

```
#ifndef __user_module_01_h__
#define __user_module_01_h__

#include <event.h>
#include <object.h>

class UserModule01 : public NslObject {

 private:
        int             Number;
        char            *String;

 public:

        UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name);
        ~UserModule01();

        int             init();
        int             command(int argc, const char *argv[]);
        int             recv(ePacket_ *pkt);
        int             send(ePacket_ *pkt);
        void            pkt_delay(ePacket_ *pkt);
};

#endif    /* __user_module_01_h__ */
```

使用第八章的 8-1 的范例，如下面红字所示。并加上这章节的蓝字部分。

user_module01.cc

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>
#include <packet.h>
#include <ethernet.h>
#include <ip.h>

MODULE_GENERATOR(UserModule01);
```

```
UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name)
        : NslObject(type, id, pl, name) {
        vBind_int("myNumber", &Number);
        vBind_char_str("myString", &String);
        REG_VAR("shared-number", &Number);
}

UserModule01::~UserModule01(){}

int UserModule01::init() {
        return(NslObject::init());
}

int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        u_int64_t    current_time_in_tick;
        u_int64_t    delay_time_in_tick;
        BASE_OBJTYPE(mem_func);

        current_time_in_tick = GetCurrentTime();
        SEC_TO_TICK(delay_time_in_tick, 1);
        mem_func = POINTER_TO_MEMBER(UserModule01, pkt_delay);
        setObjEvent(pkt, current_time_in_tick + delay_time_in_tick,
                0, this, mem_func, (void *)pkt->DataInfo_);
        return(1);
}

void UserModule01::pkt_delay(ePacket_ *pkt) {

        Packet                  *packet;
        struct ether_header     *eh;
```

```
        char                src_mac_str[18];
        char                dst_mac_str[18];

        struct ip           *iph;
        char                src_ip_str[16];
        char                dst_ip_str[16];

        if(pkt != NULL && pkt->DataInfo_ != NULL) {
                packet = (Packet *)pkt->DataInfo_;
                eh = (struct ether_header *)packet->pkt_get();

                macaddr_to_str(eh->ether_shost, src_mac_str);
                macaddr_to_str(eh->ether_dhost, dst_mac_str);

                printf("\e[1;36;40mExercise 9-1: Src Mac: %s, Dst Mac: %s\e[m\n",
                        src_mac_str, dst_mac_str);

                iph = (struct ip *)packet->pkt_sget();
                if(iph != NULL) {
                        ipv4addr_to_str(iph->ip_src, src_ip_str);
                        ipv4addr_to_str(iph->ip_dst, dst_ip_str);
                        printf("\e[1;36;40mExercise 9-1: Src IP: %s, Dst IP: %s\e[m\n",
                        src_ip_str, dst_ip_str);

                }
        }

        double x, y, z;
        GetNodeLoc(get_nid(), x, y, z);
        printf("\e[1;32;40mExercise 9-1: GetTotalNumOfNodes()=%d\e[m\n",
GetTotalNumOfNodes());
        printf("\e[1;32;40mExercise 9-1: GetConfigFilePathAndName()=%s\e[m\n",
GetConfigFilePathAndName());
        printf("\e[1;32;40mExercise 9-1: GetNodeLoc()=%f, %f, %f\e[m\n", x, y, z);
        printf("\e[1;32;40mExercise 9-1: getConnectNode()=%d\e[m\n", getConnectNode(get_nid(),
get_ifid()));
        printf("\e[1;32;40mExercise 9-1: GetPacketLength=%d\e[m\n", packet->pkt_getlen());
        NslObject::send(pkt);
```

```
}
```

红字部分说明请参考第八章。

蓝字部分则使用了数个 API，说明如下：

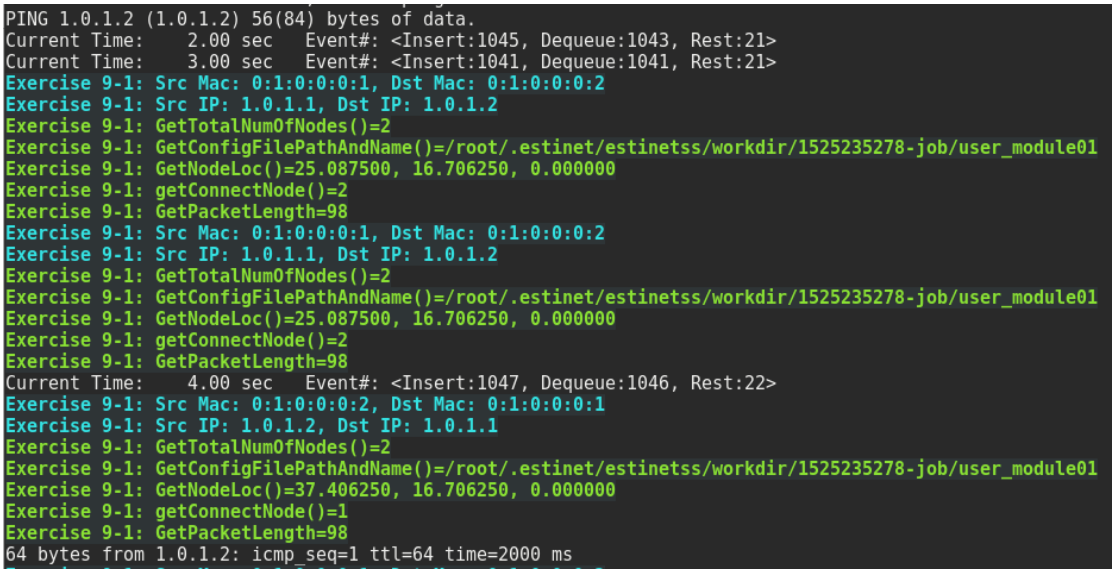
GetNodeLoc(nid, x, y, z)这个 API 可以得到 node 的位置。第一个参数是带入自己的 node id。而回传的信息将会存在参数 x,y,z。

而 GetTotalNumOfNodes 这个 API 则回传模拟环境中所有节点的数量。

GetConfigFilePathAndName 这个 API 回传目前配置文件的档案路径跟这个拓扑名称。

getConnectNode 这个 API，则是可以得到目前与这个 node id 相连的 node id，第一个参数要带入 node id，第二个参数则是实体联机编号，使用 get_ifid()得知即可。

执行结果如下图所示：



附录

附录 A、MDF HeaderSection 的细节

The first table collects the relevant variables and their meanings. The second table lists the set of possible values for each variable.

Field Name	Meaning
ModuleName	The name of this module
GroupName	The name of the group this module belongs to. And GUI will list GroupName in Node Editor to classify mdf file by GroupName.
Introduction	A short description or comment about this module
Parameter	A start-time parameter variable. The GUI program reads this part to know what parameters will be used at start-time. With this information, it will export these start-time parameters in the generated .if_and_medium_conffile.

TABLE THE MEANINGS OF THE VARIABLES USED IN THE HEADERSECTION

Field Name	Possible Values
ModuleName	Any user-specified string
GroupName	80211p, classification, interface, mac8023, mnode, phy, sdn_wifi_infra, traffic_control, vehicular_network, ap, hub, mac80211, mifx, openflow, pktgen, teleportal, user_defined, wphy (A user can create a new module group)
Introduction	Any user specified comment description string
Parameter	The format of a parameter statement is explained as follows: **Parameter Name Value Attribute** The possible attributes are listed below: “**local,**” “**gui_autogen,**” “**gui_autoassign,**” and “**local_only**”. “**local**” means that this parameter is used only in this module and if its value is updated, it will not be copied to other modules. Unless press "C.T.O.N." button, it can be copied to all modules on all nodes with the same node type. ”**gui_autogen**” means that the value of this parameter will be

	automatically generated by the GUI program. However, a user can **not** replace/change the auto-generated value with his (her) value. If a user press "C.T.O.N." button, the auto-generated value will be **not** copied to any module. Normally, a possible value of an **gui_autogen** parameter is a formula consisting of the three predefined variables: **$CASE$, NID, and PID.** **$CASE$** represents the main file name of a simulation case's topology file. It will be replaced by the main file name when this variable is accessed. For example, if a simulation case's topology file is saved with the filename "test.xtpl", **$CASE$** will be replaced by "test." **NID** represents the ID of the node to which this module is attached. Analogously, **PID** represents the ID of the interface to which this module is attached. "**gui_autoassign**" is similar to "**gui_autogen**" However, a user can **not** replace/change its value. No matter how a user replaces/changes the auto-generated value with his (her) desired one, the final value is still determined by a pre-defined formula. ex. ip address, mac address, interface id...etc.. "**local_only**" means that the value of this parameter can be replace/change the value with his (her) value. If a user press "C.T.O.N." button, the value will be **not** copied to any module. Because the parameter must be set respectively.

TABLE THE POSSIBLE VALUES FOR THE VARIABLES USED IN THE HEADERSECTION

附录 B、MDF InitVariable Section 的细节

Normally, a user should specify the caption and the size of the diabox. The key word "**Caption**" indicates the caption of the dialog box, and "**FrameSize width height**" indicates the size of the dialog box. For example,

```
Caption         "Parameters Setting"
FrameSize       340 80
```

These statements will generate a dialog box of 340x80 pixels with a caption of "Parameters Setting." After specifying the caption and the size of the dialog box, a user can arrange the layout inside the dialog box. A dialog box would contain a number of

GUI objects, such as an OK button, a Cancel button, a textline, etc. Each GUI object corresponds to a description block in “InitVariableSection” and always starts with “Begin” and ends with “End.” The following shows an example:

```
Begin BUTTON        b_ok
    Caption         "OK"
    Scale           270 12 60 30
    ActiveOn        MODE_EDIT
    Enabled         TRUE
    Action          ok
    Comment         "OK Button"
End
```

The description blocks for different objects share several common and basic attributes. For example, the caption and scale commands are used commonly. A “BUTTON”-like object is an example of an object consisting of only basic attributes. Let’s take the simple “BUTTON” object as an example. More specific attributes will be discussed later.

For a “BUTTON” object, the keyword “BUTTON” follows the keyword “Begin” and it is followed by the object name “b_ok”. The following table lists its attributes:

Attribute name	Possible values	Comment
Caption	User-specified	The caption of this object
Scale	User-specified	The four numbers represent (x, y, width, height).
ActiveOn	MODE_EDIT, MODE_SIMULATION, ALL_MODE	An option to specify in which mode this object should is active. The “MODE_EDIT” stand for the object is enabled at Edit Mode. The “MODE_SIMULATION” stand for the object is enabled at GUI G Mode. ALL_MODE indicates that the object will be activated whatever the Mode is.
Enabled	TRUE, FALSE	If an object is not enabled, it will be dimly displayed. That is, a user cannot operate this object.
Action	ok , cancel	An attribute used by button-like objects, such as the OK button and cancel

		buttons to indicate which action it should perform when a user presses it.
Comment	User-specified	Comment for this object

TABLE THE BASIC ATTRIBUTES USED TO DESCRIBE AN OBJECT

a. LABEL

"LABEL" is used to display some comment in a dialog box. The attributes of a LABEL object are the same as those of a "BUTTON" object. An example is following below:

```
Begin LABEL         l_ums
      Caption       "(ms)"
      Scale         220 24 35 35
      ActiveOn      MODE_EDIT
      Enabled       FALSE
End
```

b. GROUP

GROUP is used to organize related objects together. It can contain a number of objects that are related to an area. Like other objects, it has four basic attributes "**Caption**," "**Scale**," "**ActiveOn**," and "**Enabled**" to define the caption, the size of its area, the active mode, and the enabled/disabled conditions. An example is following below. The group has four objects include a Radiobox, two textline, a lable.

```
Begin Group         g_radio
      Caption       "Mode"
      Scale         10 15 260 135
      ActiveOn      MODE_EDIT
      Enabled       TRUE

      Begin RADIOBOX      myString
            Option        "op1"
            Enable        myNumber
            Enable        lable1
            OptValue      "string1"
            VSpace        5
            EndOption

            Option        "op2"
```

```
            Disable         myNumber
            Disable         lable1
            OptValue        "string2"
            VSpace          40
            EndOption

            Type            STRING
            Comment         "radiobox test"
        End

        Begin TEXTLINE      myNumber
            Caption         "input Number"
            Scale           35 35 180 35
            ActiveOn        MODE_EDIT
            Enabled         FALSE
            Type            INT
            Comment         "for test"
        End

        Begin LABEL         lable1
            Caption         "(INT)"
            Scale           220 35 35 35
            ActiveOn        MODE_EDIT
            Enabled         FALSE
        End
    End
```

c. RADIOBOX/CHECKBOX

In RADIOBOX/CHECKBOX, there are some new attributes. (Note: Outside of a **radiobox** must be a group object) Let's take the following example to explain:

```
    Begin Group         g_radio
        Caption         "Mode"
        Scale           10 15 260 135
        ActiveOn        MODE_EDIT
        Enabled         TRUE

        Begin RADIOBOX  myString
```

```
            Option          "op1"
            Enable          myNumber
            Enable          lable1
            OptValue        "string1"
            VSpace          5
            EndOption

            Option          "op2"
            Disable         myNumber
            Disable         lable1
            OptValue        "string2"
            VSpace          40
            EndOption

            Type            STRING
            Comment         "radiobox test"
        End

    Begin TEXTLINE      myNumber
            Caption     "input Number"
            Scale       35 35 180 35
            ActiveOn    MODE_EDIT
            Enabled     FALSE
            Type        INT
            Comment     "for test"
    End

    Begin LABEL         lable1
            Caption     "(INT)"
            Scale       220 35 35 35
            ActiveOn    MODE_EDIT
            Enabled     FALSE
    End
End
```

It is a RADIOBOX block whose name is "***myString***." The two option blocks follow, each of which starts with the "**Option**" keyword and ends with the "**EndOption**" keyword. The string following the "Option" keyword specifies the string that should be shown in

the dialog box for this option. The "**OptValue**" specifies the value that will be assigned to the radiobox option variable "***myString***" if this option is selected. The "Enable" and "Disable" statements inside an "Option" block specify that, when a user selects this option, the variable objects following these statements should be enabled or disabled (When an object is enabled, its input field is enabled in the parameter dialog box, otherwise, its input field is disabled). The term "**VSpace**" is used to specify the vertical height of the area used for outside group's y location(only using for Radiobox). The term "**Comment**" is used to specify comment for this object.

```
Begin CHECKBOX      check1
      Caption       "Set My Number"
      Scale         10 50 180 20
      ActiveOn      MODE_EDIT
      Enabled       TRUE

      Option        "TRUE"
      OptValue      "on"
      Enable        myNumber
      EndOption

      Option        "FALSE"
      OptValue      "off"
      Disable       myNumber
      EndOption
      Comment       ""
End
```

The following is a checkbox block whose name is "*check1*." Like other objects, it has four basic attributes "**Caption**," "**Scale**," "**ActiveOn**," and "**Enabled**" to define the caption, the size of its area, the active mode, and the enabled/disabled conditions. And then, the two option blocks follow, each of which starts with the "**Option**" keyword and ends with the "**EndOption**" keyword. The string following the "Option" keyword specifies the string that should be shown in the dialog box for this option. The "**OptValue**" specifies the value that will be assigned to the checkbox option variable "*check1*" if this option is selected. The "Enable" and "Disable" statements inside an "Option" block specify that, when a user selects this option, the variable objects following these statements should be enabled or disabled (When an object is enabled, its input field is enabled in the parameter dialog box, otherwise, its input field is

disabled). The term “**Comment**” is used to specify comment for this object.

d. TEXTLINE

TEXTLINE provides a text field for inputting or outputting data. Like other objects, it has four basic attributes “**Caption**,” “**Scale**,” “**ActiveOn**,” and “**Enabled**” to define the caption, the size of its area, the active mode, and the enabled/disabled conditions. A module developer can indicate the type of the data to be read from a textline. The data will be interpreted as a value of the type indicated by the “TYPE” key word. But now we only support “**STRING**” for “TYPE”, other data type not yet. The term “**Comment**” is used to specify comment for this object. An example is following below.

```
Begin TEXTLINE      myNumber
      Caption       "My Number "
      Scale         10 70 220 30
      ActiveOn      MODE_EDIT
      Enabled       FALSE
      Type          INT
      Comment       "An Integer"
End
```

附录 C、MDF 中的 Export Section 细节

“ExportSection” provides an area in a dialog box in which a user can get/set the current value of a variable at run-time. “**Caption**”, “**FrameSize**” are the two basic attributes for this section. If a module doesn’t have any variable that can be accessed during simulation, “Caption” should be set to “”, a null string, and “FrameSize” should be set to 0 0. Or the ExportSection does not be added.

```
ExportSection
      Caption       ""
      FrameSize     0 0
EndExportSection
```

In addition to the objects discussed above, there are two useful objects that are new in this section. They are the “**INTERACTIONVIEW**” and “**ACCESSBUTTON.**” The formats of these two objects are shown in the following examples:

```
Begin ACCESSBUTTON  ab_get_mystr
        Caption         "Get"
        Scale           215 50 70 25
        ActiveOn        MODE_SIMULATION
        Enabled         TRUE
        Action          GET
        ActionObj       "export-my-string"
        Reference       text_query_mystr
        Comment         "get"
End

Begin ACCESSBUTTON  ab_set_mynum
        Caption         "Set"
        Scale           290 15 70 25
        ActiveOn        MODE_SIMULATION
        Enabled         TRUE
        Action          SET
        ActionObj       "export-my-number"
        Reference       text_query_mynum
        Comment         "set"
End
```

For an "**ACCESSBUTTON**" object, it is used to get or set the value of a single-value run-time variable. There are three new attributes for "**ACCESSBUTTON**." They are "**Action,**" "**ActionObj,**" and "**Reference,**" respectively. The value of "Action" can be "**GET**" or "**SET**" to indicate when a user presses this button which operation should be performed. "**ActionObj**" indicates the name of the object that the GET/SET operation should operate on in the simulation engine. Finally, "**Reference**" points to the name of the GUI object (e.g., a TEXTLINE object) in which the retrieved value should be displayed. For example, the max queue length of a Interface module may be gotten and displayed at a TEXTLINE GUI object named "t_mq."

```
Begin INTERACTIONVIEW      iv_get_all
        Caption            "Get All Var"
        Scale              10 100 200 30
        ActiveOn           MODE_SIMULATION
        Enabled            TRUE
        Action             GET
        ActionObj          "export-all-data"
        Fields             "My String" "My Number"
        Comment            "All Data"
End
```

For an "**INTERACTIONVIEW**" object, it is used to display the content of a multi-column table at run-time. Normally, it is used to get a switch table, an ARP table, or an AP's association table. Besides "**Action**" and "**ActionObj**," there is a new attribute called "**Fields**" to specify the names of the fields (columns) of the table. Several quoted strings, each of which represents the name of a field, follow the "**Fields**" attribute.

附录 D、仿真器的分布式架构：

EstiNet uses a distributed architecture to support remote simulations and concurrent simulations. The estinetjd is used to do this task. It should be executed and remain alive all the time to manage multiple simulation machines. On every simulation machine, the estinetss needs to be executed and remain alive to let the estinetjd know whether currently this machine is busy running a simulation case or not. The following figure depicts the distributed architecture of EstiNet.

For example, the estinetjd in the simulation service center can accept simulation jobs from the whole world. When a user submits a simulation job to the estinetjd, the estinetjd selects an available simulation machine to service the job. If there is no available simulation machine, the job will be put into the job queue of the estinetjd. Every simulation machine always has a running estinetss to communicate with the GUI program and the estinetjd. The estinetss will notify the estinetjd whether the simulation machine managed by itself is available or not. When the estinetss receives a simulation job from the estinetjd, it forks (executes) a simulation engine process to simulate the specified network and protocols. When the simulation engine process is running, the estinetss will communicate with the estinetjd and the GUI program. For example, periodically the simulation engine process will send the current virtual time

of the simulation network to the estinetss. Then the estinetss will relay the information to the GUI program. This enables the GUI user to know the progress of the simulation. During a simulation, the user can also on-line set or get a protocol module's value (e.g. to query a switch's switch table). Message exchanges happening between the simulation engine process and the GUI program are all done via the estinetss.