

12th European EMME/2 Conference
Basel, Switzerland - May 22-23, 2003

SEnC - A Sequential Enif Client

Heinz Spiess, EMME/2 Support Center, Aegerten, Switzerland

EMME/2:

- Macro command language

- sequential dialogs---> sequential macros
- is part of the EMME/2 program
- cannot be replaced by the user

Enif:

- "Server" mode (enif -S <portno>)
 - TCP/IP server
 - remote access over LAN or Internet
 - open non-proprietary protocol
 - no predefined sequential order (event driven)
 - direct access to the Enif "engine"
 - Enif responsible for server part

Everything is possible! :-)

But someone has to program it first! :-)

Enif's basic concept of configurability:

■ Parameter:

- Name, description

- Type

Click, Bool, Integer, Float, String, Expression, Selector, Stylus, ...

- Flags

- Group specification

send/receive/update

- One or several indexed values

In Enif all configuration information is implemented by means of parameters

Enif's basic concept of configurability ...

Parameter group specification:

- send group / receive group / update group

Parameter grouping:

ParA : Joe/Mary

ParB : Joe/Joe

ParC : Mary/

ParD : /Sue/Mary

ParA = 16	-----Joe---->	ParB = 16
ParB = 55	-----Joe---->	
ParC = 0	-----Mary--->	ParA = 0 , Update(ParD)

Enif's basic concept of configurability ...

Configurable objects:

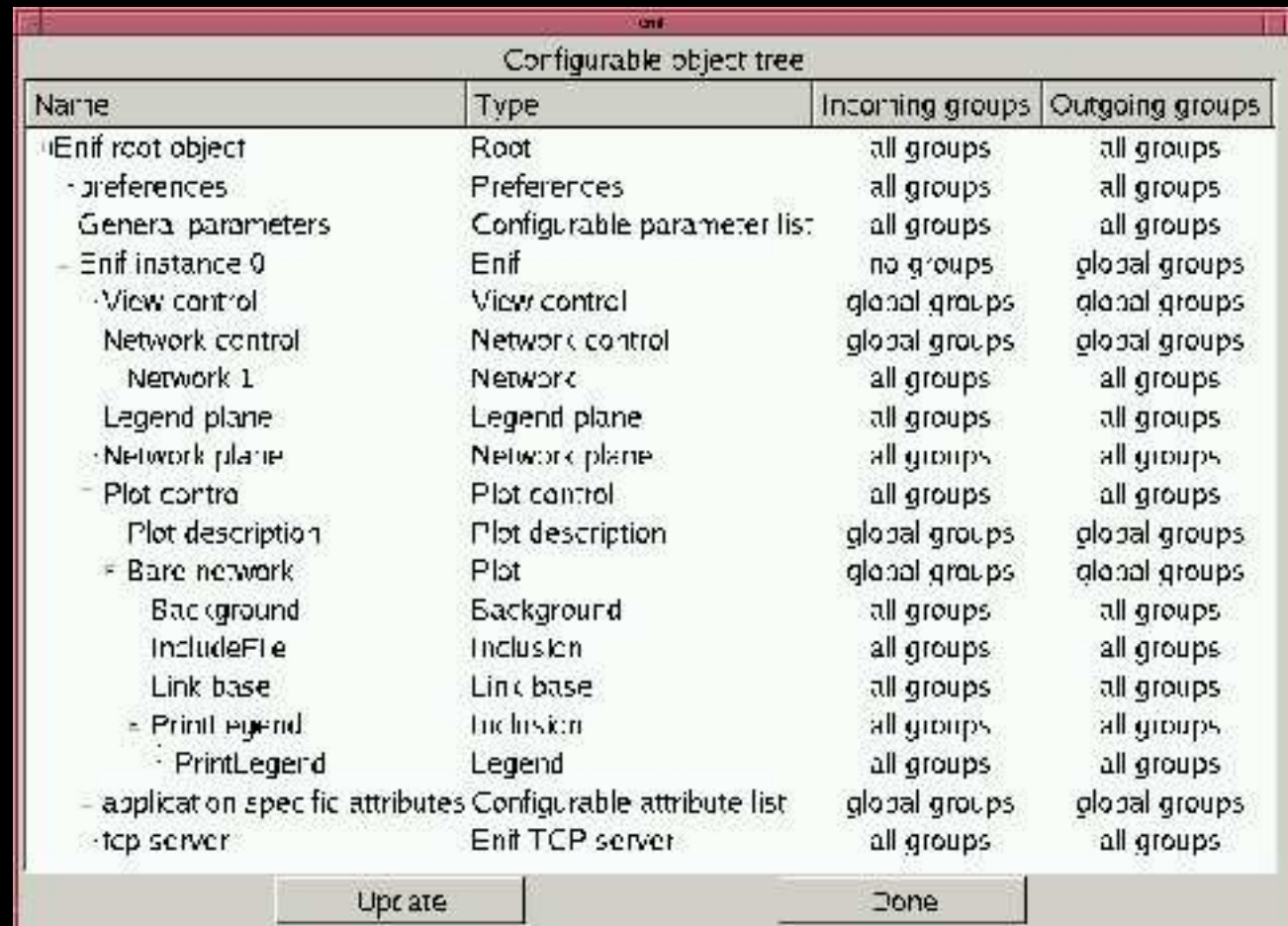
Each functional block of Enif is implemented as a configurable object.

Each configurable object owns a set of parameters.

System objects are those functionalities which are not configurable by the user.

System parameters start with "\$".

Configurable objects form a tree:



The screenshot shows a window titled 'Configurable object tree' with a table listing various objects and their group relationships. The table has four columns: Name, Type, Incoming groups, and Outgoing groups. The objects listed include a root object, preferences, general parameters, an Enif instance, view and network controls, legend and network planes, plot controls, a bare network, background, include files, link bases, print legends, application-specific attributes, and a TCP server.

Name	Type	Incoming groups	Outgoing groups
Enif root object	Root	all groups	all groups
- preferences	Preferences	all groups	all groups
General parameters	Configurable parameter list	all groups	all groups
- Enif instance 0	Enif	no groups	global groups
- View control	View control	global groups	global groups
Network control	Network control	global groups	global groups
Network 1	Network	all groups	all groups
Legend plane	Legend plane	all groups	all groups
- Network plane	Network plane	all groups	all groups
- Plot control	Plot control	all groups	all groups
Plot description	Plot description	global groups	global groups
- Bare network	Plot	global groups	global groups
Background	Background	all groups	all groups
IncludeFile	Inclusion	all groups	all groups
Link base	Link base	all groups	all groups
- PrintLegend	Inclusion	all groups	all groups
- PrintLegend	Legend	all groups	all groups
- application specific attributes	Configurable attribute list	global groups	global groups
- tcp server	Enif TCP server	all groups	all groups

Update Done

Parameter group signal can propagate along the object tree.
Each object has group filters to control inter-object grouping.

Typical tasks used in automated procedures:

- change the network scenario
- load a new plot configuration
- change the current view
- print the current plot
- export the current plot to an image file

Which system parameters control these functionalities?

Some important system parameters:

Network control:

`$LoadScenario` (Integer, write-only, indexed)

Load scenario onto network stack

`$ScenarioNumber` (Integer, read-only)

Read out current scenario number

`$ScenarioTitle` (String, read-only)

Read out current scenario title

Plot control:

`$LoadPlotConfiguration` (String, write-only)

Load a new plot configuration from the specified file

View control:

`$CurrentView` (Box, read-write)

Read or redefine coordinate box of current view

Important system parameters ... :

Network plane:

\$PrintCurrentViewNoSetup (Click, write-only)

Send current view to the default printer

\$ExportScreenViewToFile (String, write-only)

Export current screen view to an bitmap image file

\$ExportPrintViewToFile (String, write-only)

Export current screen view to an bitmap image file

\$ExportEnlargementFactor (Integer, write-only)

Increase the resolution of the bitmap image file

Communicating with an Enif server:

Server commands:

lp [<parameter>]

new <partype> <parname> [<elementtype>]

delete <parname>

check <expressionpar>

eval <expressionpar> <selectorpar>

echo <anytext>

exit

Parameter specifications:

<parameter> : send/receive/update

<parameter> = <value>

<parameter>[<index>] = <value>

Acknowledgments:

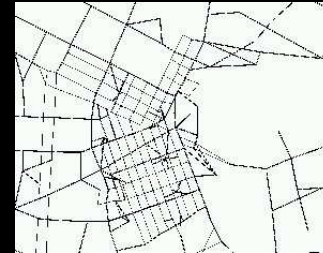
"OK" -> command processed with no error

"KO" -> error encountered

Communicating with an Enif server:

Simplest possible Enif client?-----> **Telnet!**

```
% enif -S 9009
% telnet localhost 9009
Trying 127.0.0.1...
Connected to localhost (127.0.0.1).
Escape character is '^]'.
OK
new Box view
OK
view : $CurrentView/
OK
view= -1,-1,1,1
OK
exit
Connection closed by foreign host.
```



---> Use telnet to explore Enif's internals!

The SEnC Program:

- An Enif client for processing sequential input files
- "Enif scripting"
- Calling sequence:

```
senc <options> <file|varspec> ...
```

- Options:

-d	Generate debug output
-e	Continue after error
-i	Interactive mode
-l	Launch Enif automatically
-p <port>	Define port used to contact Enif server
-q	Quit after all file have been processed
-s <server>	Define host name / IP-address of Enif server
-v	Verbose output

The SEnC Program:

- Calling sequence:

`senc <options> <file|varspec> ...`

- Input files:

contain commands that are processed sequentially

- Variable specification:

`<variable>=<value>` No spaces!

Example: `SCENARIO=2000`

Automatic substitution for variable names in braces occurring in input commands, e.g.

`$LoadScenario[1] = {SCENARIO}`

The SEnC Program:

- Client commands:

Commands which are not passed to the Enif server, but processed locally. Always start with "!"

!set <variable>=<value>

Set variable to specified value

!unset <variable>

Unset (delete) a variable

!print <anytext>

Perform variable substitution and output

!wait <seconds>

Wait for the given number of seconds

!prompt <message>

Issue prompt message and wait for user reply

The SEnC Program:

- Client commands (continued):

```
!function <function> [<argname> ...]  
... commands (using {<argname>} substitution)...  
!!
```

Define a function with arguments

```
!call <function> [<argvalue> ...]  
Call a function with argument values
```

```
!if <value1> <compop> <value2>  
<conditional command>  
Compare the values and execute the next  
command only if the specified condition  
is true.  
Conditions: == != < > <= >= <>
```


The SEnC Program - An Example

Header file "plotgen.sen":

a) Creation and grouping of parameters

new Integer Scenario

Scenario : \$LoadScenario/

new String PlotConf

PlotConf : \$LoadPlotConfiguration/

new Box View

View : \$CurrentView/

new Click FullView

FullView : \$FullView/

Header file "plotgen.sen" (continued):

b) Creation and grouping of parameters (cont.)

```
new Integer ExportEnlargementFactor
```

```
ExportEnlargementFactor : $ExportEnlargementFactor/
```

```
new String ExportScreenView
```

```
ExportScreenView : $ExportScreenViewToFile/
```

```
new Click PrintView
```

```
PrintView : $PrintCurrentViewNoSetup/
```

c) define function "scenario" to change scenarios

```
!function scenario SCENARIO
```

```
!if 0 < {SCENARIO}
```

```
Scenario[1] = {SCENARIO}
```

```
echo Scenario %<$ScenarioNumber>%: %<$ScenarioTitle>%
```

```
!!
```

Header file "plotgen.sen" (continued):

d) define function "view" to select predefined views

```
!function view VIEW
!print Changing view to {VIEW} ...
!if full == {VIEW}
FullView = 1
!if airport == {VIEW}
View = -10;-3;-7;3;Airport;
!if cbd == {VIEW}
View = -2;-2;2;2;CBD;
!if kildonan == {VIEW}
View = -1.2;1.5;4.2;4.8;Kildonan corridor;
!!
```

e) define function "export" to export view to image file

```
!function export FILENAME
ExportScreenView = {FILENAME}
!print Exporting view to file {FILENAME} ...
!!
```

Header file "plotgen.sen" (continued):

f) define function "plot" to load plot configuration

```
!function plot CONFIGURATION
!print Loading {CONFIGURATION} ...
PlotConf = {CONFIGURATION}
!!
```

g) define function "plot" to load plot configuration

```
!function action {ACTION}
!if prompt == {ACTION}
!prompt Continue?
!if wait == {ACTION}
!wait 5
!if print == {ACTION}
PrintView = 1
!if print == {ACTION}
!print Printing view ...
!!
```

Task file "volumeplots" generates four different plots for the specified scenario:

```
!call scenario {SCENARIO}
```

```
!call plot autovol.e2p
```

```
!call action
```

```
!call view cbd
```

```
!call action
```

```
!call view kildonan
```

```
!call action
```

```
!call plot transitvol.e2p
```

```
!call view cbd
```

```
!call action
```

Running the example - calling SEnC:

`senc -l plotgen.sen volumeplots`

No action taken, just produces flicker on the screen as one plots follows the next.
Default scenario is used.

`senc -l plotgen.sen SCENARIO=2000 ACTION=wait volumeplots`

Four plots are generated for scenario 2000.
After each plot there is a 5 second pause.

`senc -l plotgen.sen SCENARIO=3000 ACTION=print volumeplots`

Four plots are generated for scenario 3000
and sent to the default printer.

Availability of SEnC:

Download from website of the EMME/2 Support Center

<http://www.spiess.ch/emme2>

Binary executables for Windows, Linux and Sun

Source code C++ / Qt for those interested to program their own clients.

SEnC is a voluntary contribution, it is not part of the official EMME/2 / Enif software distribution.

No warranty!

Conclusions:

Enif provides a very powerful client/server mechanism for implementing scripting and remote control.

Enif clients are completely independent of the Enif program. They can be programmed by anyone, using any programming language and providing any type of interface that might be needed.

SEnC is just one simple example of such a client, whose aim is to automate simple repetitive tasks.

Enif clients can also be programmed for more exciting tasks, such as e.g. implementing a direct interface with a web server, in order to allow accessing Enif generated plots and data over the web!