The CanFestival CANopen stack manual.

CanFestival v3.0 Manual

1 -   Introduction

CanFestival is an OpenSource (LGPL and GPL) CANopen framework.

1.1)   The CanFestival project

This project, initiated by Edouard TISSERANT in 2001, has grown thanks
to Francis DUPIN and other contributors.

Today, CanFestival focuses on providing an ANSI-C platform independent
CANopen stack that can be implemented as master or slave nodes on PCs,
Real-time IPCs, and Microcontrollers.

CanFestival is a project supported by Lolitech.

1.2)   What is CANopen

CANopen is a CAN based high level protocol. It defines some protocols to :

1. Configure a CAN network.

2. Transmit data to a specific node or in broadcast.

3. Administrate the network. For example detecting a not responding
   node.

The documentation can be found on the CAN in Automation website :
   http://www.can-cia.de/canopen

The most important document about CANopen is the normative CiA Draft Standard 301, version 4.02. You can now download the specification from the CAN in Automation website at no cost.

To continue reading this document, let us assume that you have read some papers introducing CANopen.

2 -   CanFestival Features

2.1)   Tools

The CANopen library is coming with some tools :

1. Object Dictionary editor GUI. WxPython Model-View-Controler based GUI, that help a lot in generating object dictionary source code for each node.

2. A configure script, that let you chose compile time options such as target CPU/HOST, CAN and TIMER drivers.
   This script has not been generated with autoconf, it has been made keeping micro-controller target in mind.

2.2)   Standard conformance

a)   Multi-Platform

1. Library source code is C-ANSI.

2. Driver and examples coding conventions merely depend on target specific contributor/compiler.

3. Unix compatible interfaces and examples should compile and run on any Unix system (tested on GNU/Linux and GNU/FreeBSD).

b)   CANopen conformance   DS-301

1. Should conform to DS301. V.4.02 13 february 2002.

2. Master and Slave functionality implemented.

3. Sending SYNC implemented.

4. 1 SDO server per node. (update: more than one possible. To be more tested)

5. Unlimited SDO client.

6. SDO transmission mode : normal, expedited download and upload.

7. Unlimited PDO receive.

8. Unlimited PDO transmit.

9. Object Data type implemented : 8, 16, 32 bits values, and fixed length strings.

10. Slave state full implemented.

11. NMT to change slave's state implemented.

12. PDO transmission mode : on request, every reception of 0 to n SYNC, on event.

13. NMT Heartbeat implemented : A node can be either heartbeat producer or receiver.

14. NMT NodeGuard implemented : Not fully implemented.

15. TIME (time Stamp) : Not implemented.

16. EMCY (emergency objects) : Sending and reception of EMCY objectos (without manufacturer specific error field).

17. PDO Mapping bit per bit implemented.

DS-302

1. Concise DFC : implemented.

3 -   How to start

3.1)   Host requirements

What you need on your development workstation.

3.1.1)   Object Dictionary Editor GUI

1. Python, with

2. wxPyhon modules installed (at least version 2.6.3).

3. Gnosis xml tools. (Optional can also be installed locally to the project automatically with the help of a Makefile. Please see 9.1) Using Dictionary Editor GUI)

3.1.2)   Linux and Unix-likes

1. Linux, FreeBSD, Cygwin or any Unix environment with GNU toolchain.

2. The GNU C compiler (gcc) or any other ANSI-C compiler for your target platform.

3. Xpdf, and the official 301_v04000201.pdf file in order to get GUI context sensitive help. Download the ds301 at http://www.can-cia.org/downloads/ciaspecifications/?1390.

4. GNU Make

5. Bash and sed

3.1.3)   Windows (for native win32 target)

1. Visual Studio Express 2005 or worst.

2. Microsoft platform SDK (requires Genuine Advantage)

3. Cygwin (for configuration only)

3.2)   How to get CanFestival

Please always use CVS, this is the best way to get the most reactive support from the developer community :

   cvs -d:pserver:anonymous@lolitech.dyndns.org:/canfestival login
(type return, without entering a password)
   Then, enter :
cvs -z3 -d:pserver:anonymous@lolitech.dyndns.org:/canfestival co -P CanFestival-3

4 - Understanding Canfestival

4.1) CanFestival Project tree layout

Simplified directory structure.

```
./src ANSI-C source of CANopen stack

/include Exportables Header files

./drivers Interfaces to specific platforms/HW

./drivers/unix Linux and Cygwin OS interface

./drivers/win32 Native Win32 OS interface

./drivers/timers_xeno Xenomai timers/threads (Linux only)

./drivers/timers_kernel Linux kernel timer/threads

./drivers/timers_unix Posix timers/threads (Linux, Cygwin)

./drivers/can_virtual_kernel Fake CAN network (kernel space)

./drivers/can_peak_linux PeakSystem CAN library interface

./drivers/can_peak_win32 PeakSystem PCAN-Light interface

./drivers/can_uvccm_win32 Acacetus's RS232 "CAN-uVCCM" interface

./drivers/can_virtual Fake CAN network (Linux, Cygwin)

./drivers/hcs12 HCS12 full target interface

./examples Examples

./examples/TestMasterSlave 2 nodes, NMT SYNC SDO PDO, win32+unix

./examples/TestMasterMicroMod 1 node, control Peak I/O Module,
unix

./examples/gene_SYNC_HCS12 Just send periodic SYNC on HCS12

./examples/win32test Ask some DS301 infos to a node (win32)

./objdictgen Object Dictionary editor GUI

./objdictgen/config Pre-defined OD profiles

./objdictgen/examples Some examples/test OD

./doc Project and CANopen doc
```

4.2) Implement CanFestival in your application
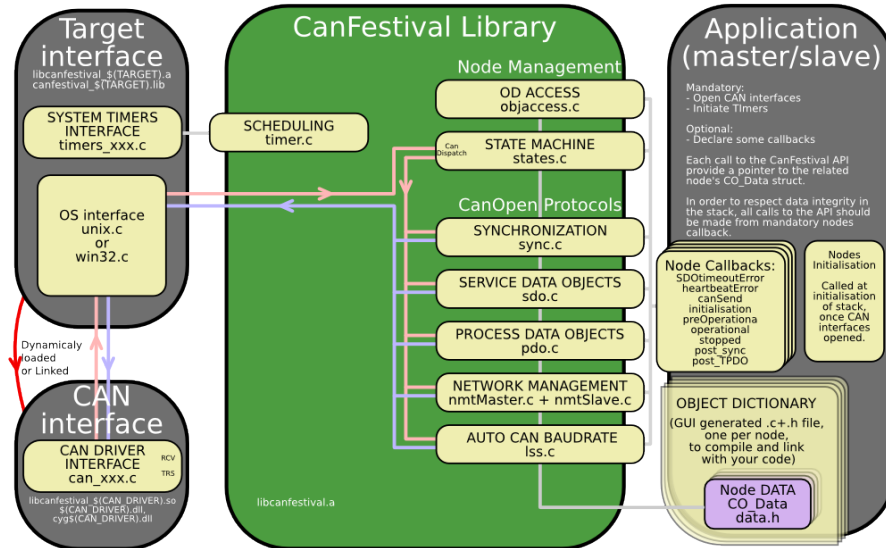
4.3) CanFestival CAN interfaces

Because most CAN controllers and drivers implement FIFOs, CanFestival consider sending message as a non bloking operation.

In order to prevent reentrent calls to the stack, messages reception is implemented differently on µC and OS.:

1. µC must provide interuption masking for timer and can receive TI
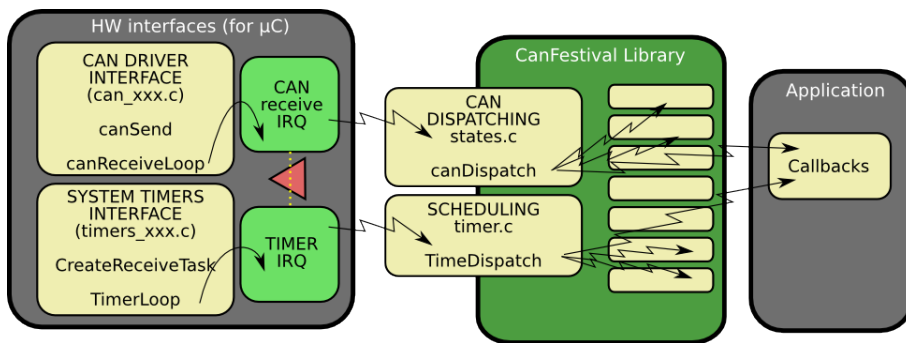
2. OS must provide a receive thread, a timer thread and a mutex. CAN reception is a bloking operation.

CanFestival integration with Unix and win32 OS
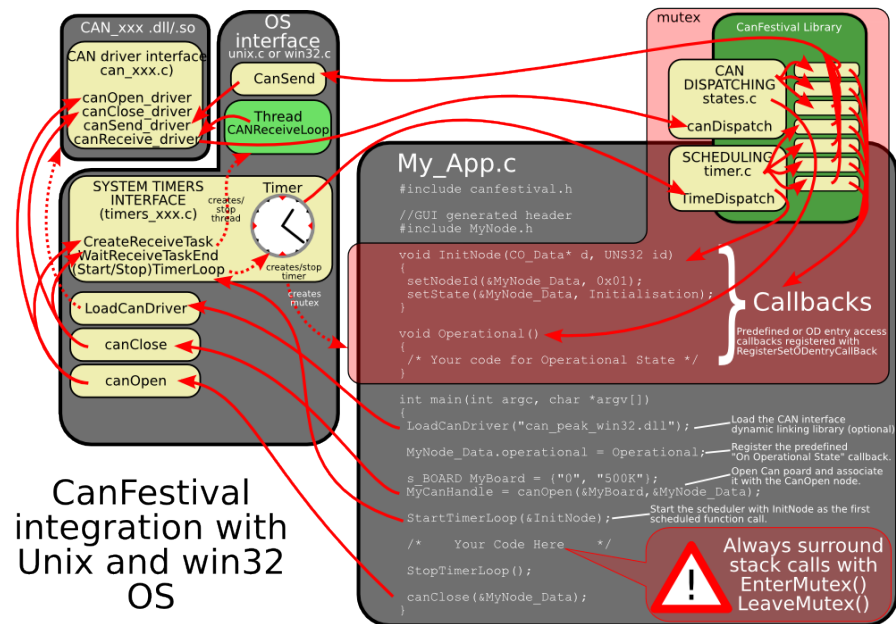
4.4)   CanFestival events scheduling

A CANopen node must be able to take delayed actions.

As exemples, periodic sync emission, heartbeat production or SDO timeout need to set some alarms that will be called later and do the job.

µC generaly do not have enough free timers to handle all the CANopen needs directly. Moreover, CanFestival internal data may be corrupt by reentrant calls.

CanFestival implement a micro-scheduler (timer.c). It uses only one timer to mimic many timers. It manage an alarm table, and call alarms at desired time.
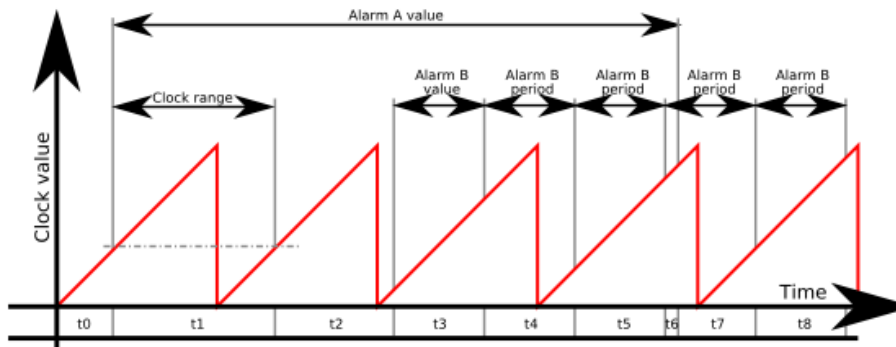
Scheduler can handle short clock value ranges limitation found on some µC. As an example, value range for a 16bit clock counter with 4µs tick is crossed within 0.26 seconds... Long alarms must be segmented.

Chronogram illustrate a long alarm (A) and a short periodic alarm (B), with a A value > clock range > B value. Values t0...t8 are successive setTimer call parameter values. t1 illustrates an intermediate call to TimeDispatch, caused by a delay longer than clock range. Because of long alarm segmentation, at the end of t1, TimeDispatch call will not trig any alarm callback.
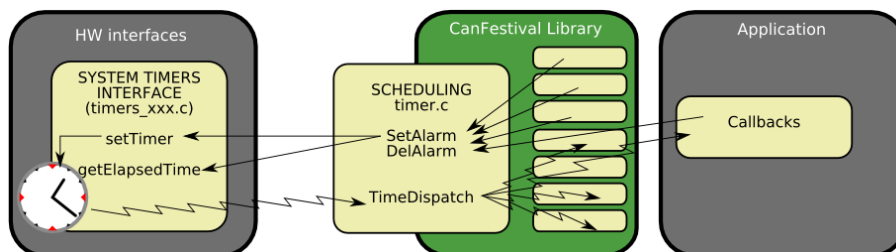
5 -   Linux Target

Linux target is default configure target.

5.1)   Linux Compilation and installation

Call ./configure –help to see all available compile time options.

After invoking ./configure with your platform specific switches, just type make.

```
./configure [options]
make
make install
```

5.1.1)   Standard Linux node

Configure switch:
```
--timers=unix
```
To do a CANopen node running on PC-Linux, you need :

1. A working linux distribution

2. One or more Peak system PC CAN interface and the last Peak Linux driver installed.

5.1.2)   Real-Time Linux node

Configure switch:
```
--timers=xeno
```
To do a CANopen node running on PC-Linux, you need :

1. A working Linux distribution patched with XENOMAI 2.1 or greater.

2. One or more Peak system PC CAN interface and the last Peak Real Time Linux driver installed.

5.1.3)   Linux kernel node

To do a CANopen node running on PC-Linux in kernel space, you need:

a)   A working Linux distribution with pre-built 2.6.x.x kernel sources

b)   A CAN card driver compatible with CanFestival

5.1.4)   CAN devices

Curently supported CAN devices and corresponding configure switch:

a)   Peak systems   Configure switch:
```
--can=peak_linux
```
PeakSystems CAN interface is automatically chosen as default CAN interface if libpcan is present in the system.

Please download driver at http://www.peak-system.com/linux and follow instructions in order to install driver on your system.

b)   Socket-Can (http://socketcan.berlios.de)   Configure switch:
```
--can=socket
```

c) LinCan Configure switch:

```
--can=lincan
```

d) Virtual CAN interfaces (for test/debug) Configure switch:

```
--can=virtual (in user space)
--can=virtual_kernel (in kernel space)
```

Virtual CAN interface use Unix pipes to emulate a virtual CAN network. Each message issued from a node is repeat to all other nodes. Currently only works with nodes running in the same process, and does not support work with Xenomai.

5.2) Testing your CanFestival installation

Sample provided in /example/TestMasterSlave is installed into your system during installation.

```
TestMasterSlave
```

Default CAN driver library is libcanfestival_can_virtual.so., which will simply pass CAN messages through Unix pipes between Master and Slave.

You may also want to specify different can interface and define some CAN ports. Another example using Peak's dual PCMCIA (configure and install with –can=peak) :

```
TestMasterSlave -l libcanfestival_can_peak.so -s 40 -m 41
```

`example/kerneltest` It's based on TestMasterSlave example and has the same functionality. Uses virtual can driver as default too. After successfull installation you can insert the module by typing: modprobe canf_ktest Module control is done by simple console 'canf_ktest_console' which is used to start/stop sending data.

6 - Windows Targets

CanFestival can be compiled and run on Windows platform. It is possible to use both Cygwin and win32 native runtime environment.

6.1) Object Dictionary Editor GUI installation.

Please refer to

6.2) CYGWIN

6.2.1) Requirements

Cygwin have to be installed with those packages :

1. gcc

2. unzip

3. wget

4. make

Currently, the only supported CAN devices are PeakSystems ones, with PcanLight driver and library.

Please download driver at http://www.peak-system.com/themen/download_gb.html and follow instructions in order to install driver on your system.

Install Cygwin as required, and the driver for your Peak CAN device.

Open a Cygwin terminal, and follow those instructions:

6.2.2) Cygwin configuration and compilation

a) A single node with PcanLight and Peak CAN-USB adapter

Download the PCAN-Light Zip file for your HW (URL from download page):

```
wget http://www.peak-system.com/files/usb.zip
```

Extract its content into your cygwin home (it will create a "Disk" directory):

```
unzip usb.zip
```

Configure CanFestival3 providing path to the desired PcanLight implementation:

```
cd CanFestival-3
export PCAN_INCLUDE=~/Disk/PCAN-Light/Api/
export PCAN_HEADER=Pcan_usb.h
export PCAN_LIB=~/Disk/PCAN-Light/Lib/Visual\ C++/Pcan_usb.lib
./configure --can=peak_win32
make
```

In order to test, you have to use another CanFestival node, connect with a CAN cable.

```
cp ~/Disk/PCAN-Light/Pcan_usb.dll .
./examples/TestMasterSlave/TestMasterSlave \
-l drivers/can_peak_win32/cygcan_peak_win32.dll \
```

```
-S 500K -M none
```
Then, on the other node :
```
./TestMasterSlave -l my_driver.so -S none -M 500K
```
Now messages are beeing exchanged between master and slave node.

b) Two nodes with PcanLight and Peak dual PCMCIA-CAN adapter
Download the PCAN-Light Zip file for your HW (URL from download page):
```
wget http://www.peak-system.com/files/pccard.zip
```
Extract its content into your cygwin home (it will create a "Disk" directory):
```
unzip pccard.zip
```
The configure CanFestival3 providing path to the desired PcanLight implementation:
```
export PCAN_INCLUDE=~/Disk/PCAN-Light/Api/
export PCAN_HEADER=Pcan_pcc.h
export PCAN_LIB=~/Disk/PCAN-Light/Lib/Visual\ C++/Pcan_pcc.lib
export PCAN2_HEADER=Pcan_2pcc.
export PCAN2_LIB=~/Disk/PCAN-Light/Lib/Visual\ C++/Pcan_2pcc.lib
```
In order to test, just connect together both CAN ports of the PCMCIA card. Don't forget 120ohms terminator.
```
cp ~/Disk/PCAN-Light/Pcan_pcc.dll .
cp ~/Disk/PCAN-Light/Pcan_2pcc.dll .
./examples/TestMasterSlave/TestMasterSlave \
-l drivers/can_peak_win32/cygcan_peak_win32.dll
```
Messages are then exchanged between master and slave node, both inside TestMasterSlave's process.

6.3) Visual Studio C++

6.3.1) Requirements

Minimal Cygwin installation is required at configuration time in order to create specific header files (config.h and cancfg.h). Once this files created, cygwin is not necessary any more.

Project and solution files have been created and tested with Visual Studio Express 2005. Be sure to have installed Microsoft Platform SDK, as recommended at the end of Visual Studio installation.

6.3.2)   Configuration with cygwin

Follow instructions given at Cygwin configuration and compilation, but do neither call make nor do tests, just do configuration steps. This will create headers files accordingly to your configuration parameters, and the desired CAN hardware.

6.3.3)   Compilation with Visual Studio

You can either load independents "*.vcproj" project files along your own projects in your own solution or load the provided "CanFestival-3.vc8.sln" solution files directly.

Build CanFestival-3 project first.

a)   PcanLight and the can_peak_win32 project.   Chosen Pcan_xxx.lib and eventually Pcan_2xxx.lib files must be added to can_peak_win32 project before build of the DLL.

6.3.4)   Testing

Copy eventually needed dlls (ie : Pcan_Nxxx.lib) into Release or Debug directory, and run the test program:

```
TestMasterSlave.exe -l can_peak_win32.dll
```

6.4)   MSYS

6.4.1)   Requirements

Download from : http://sourceforge.net/project/showfiles.php?group_id=2435

1. MSYS-1.0.10.exe

2. MinGW-5.1.3.exe

3. mingwPORT (which contains wget-1.9.1)

Please download driver at http://www.peak-system.com/themen/download_gb.html and follow instructions in order to install driver on your system. Install MSYS and MingW as required, and the driver for your Peak CAN device. Open a MSYS terminal, and follow those instructions:

- extract wget-1.9.1-mingwPORT.tar.bz2

– copy wget.exe in c:\msys\1.0\bin\

– start MSYS and check the file /etc/fstab contain the line c:/MinGW /mingw

6.4.2)   MSYS configuration and compilation

Instructions for compilation are nearly the same as CYGWIN part.

a)   A single node with PcanLight and Peak CAN-USB adapter

Download the PCAN-Light Zip file for your HW (URL from download page):

```
wget http://www.peak-system.com/files/usb.zip
```

Extract its content into your MSYS's home (it will create a "Disk" directory):

```
unzip usb.zip
```

Configure CanFestival3 providing path to the desired PcanLight implementation:

```
cd CanFestival-3
export PCAN_INCLUDE=~/Disk/PCAN-Light/Api/
export PCAN_HEADER=Pcan_usb.h
export PCAN_LIB=~/Disk/PCAN-Light/Lib/Visual\ C++/Pcan_usb.lib
./configure --can=peak_win32
make
```

In order to test, you have to use another CanFestival node, connect with a CAN cable.

```
cp ~/Disk/PCAN-Light/Pcan_usb.dll .
./examples/TestMasterSlave/TestMasterSlave \
-l drivers/can_peak_win32/cygcan_peak_win32.dll \
-S 500K -M none
```

Then, on the other node :

```
./TestMasterSlave -l my_driver.so -S none -M 500K -m 0
```

Now messages are beeing exchanged between master and slave node.

b)   Two nodes with PcanLight and Peak dual PCMCIA-CAN adapter

Download the PCAN-Light Zip file for your HW (URL from download page):

```
wget http://www.peak-system.com/files/pccard.zip
```

Extract its content into your MSYS's home (it will create a "Disk" directory):

```
unzip pccard.zip
```

The configure CanFestival3 providing path to the desired PcanLight implementation:

```
export PCAN_INCLUDE=~/Disk/PCAN-Light/Api/
export PCAN_HEADER=Pcan_pcc.h
export PCAN_LIB=~/Disk/PCAN-Light/Lib/Visual\ C++/Pcan_pcc.lib
export PCAN2_HEADER=Pcan_2pcc.h
export PCAN2_LIB=~/Disk/PCAN-Light/Lib/Visual\ C++/Pcan_2pcc.lib
```

In order to test, just connect together both CAN ports of the PCMCIA card. Don't forget 120ohms terminator.

```
cp~/Disk/PCAN-Light/Pcan_pcc.dll   .
cp ~/Disk/PCAN-Light/Pcan_2pcc.dll   .
./examples/TestMasterSlave/TestMasterSlave \
-l drivers/can_peak_win32/cygcan_peak_win32.dll -m 0 -s 1
```

Messages are then exchanged between master and slave node, both inside TestMasterSlave's process.

7 -   Motorola HCS12

The examples have been tested on a MC9S12DG255 mounted on a Elektronikladen HCS12 T-board.

Beware that there are few differences in the MSCAN module of the 68HC12 and HCS12 microcontroller. For a HC12, you must adapt the driver that we provide for the HCS12.

For the difference MSCAN HC12/HCS12, see the Motorola application note AN2011/D.

Configure switch:

```
--target=hcs12
```

To do a CANopen node running on a microncontroller Motorola MC9S12DP256, you need :

1. The compiler GNU gcc for HC11, HC12, HCS12 : m6811-elf.
   Download the **release 3.1** at :   http://m68hc11.serveftp.org/m68hc11_pkg_rpm.php

2. A board with this chip. We are using the T-board from Electroniklad-
den.

3. At least about 40 kBytes of program memory.

4. A tool to flash the memory. (We are using the hight cost Lauterbach
debugger).

7.1)   Running a HCS12 node

7.1.1)   Compiling Canfestival:

```
./configure -target=hcs12
```

7.1.2)   Compiling and building an example

Enter in the folder of an HCS12 example,
```
make all
```

7.1.3)   Flashing the memory :

Use your prefered loader ! If you are using a debugger Lauterbach, you can
load the bash file : trace32_flash_programmer.cmm. It loads directly the
elf file.

7.1.4)   Connecting to a serial RS232 console :

Connect the portS(TxD0) of the HCS12 to a console configured at 19200
bauds 8N1, via a Max232 chip to adapt the electricals levels. On Linux, you
can use minicom. Connecting to a console is usefull to read the messages,
but not required.

7.1.5)   Connecting to the CAN network :

Connect the port CAN0 (pin PM0, PM1) to the network via a CAN con-
troller. On our board, the CAN controler is a PCA82C250 chip.

7.1.6)   starting the node :

Press the reset of your HCS12 board.

8 - Example and test program:

The "examples" directory contains some test program you can use as example for your own developments.

8.1) TestMasterSlave

```
****************************************************************
 *   TestMasterSlave                                          *
 *                                                            *
 *   A simple example for PC. It does implement 2 CANopen     *
 *   nodes in the same process.  A master and a slave.  Both    *
 *   communicate together, exchanging periodically NMT, SYNC, *
 *   SDO and PDO. Master configure heartbeat producer time    *
 *   at 1000 ms for slave node-id 0x02 by concise DCF.        *
 *                                                            *
 *   Usage:                                                   *
 *   ./TestMasterSlave  [OPTIONS]                             *
 *                                                            *
 *   OPTIONS:                                                 *
 *     -l :  Can library ["libcanfestival_can_virtual.so"]     *
 *                                                            *
 *    Slave:                                                  *
 *     -s :  bus name ["0"]                                     *
 *     -S : 1M,500K,250K,125K,100K,50K,20K,10K,none(disable)  *
 *                                                            *
 *    Master:                                                 *
 *     -m :  bus name ["1"]                                      *
 *     -M : 1M,500K,250K,125K,100K,50K,20K,10K,none(disable)  *
 *                                                            *
****************************************************************
```

Notes for Concise DCF :

In this example, Master configure heartbeat producer time at 1000 ms for slave node-id 0x02 by concise DCF according DS-302 profile.

Index 0x1F22 , sub-index 0x00 of the master OD, correspond to the number of entries. This equal to the maximum possible nodeId (127). Each sub-index points to the Node-ID of the device, to which the configuration belongs.

To add more parameters configurations to the slave, the value at sub-index 0x02 must be a binary stream (little-endian) following this structure :

{ (UNS32) nb of entries
(UNS16) index parameter 1
(UNS8) sub-index parameter 1
(UNS32) size data parameter 1
(DOMAIN) data parameter 1
(UNS16) index parameter 2
(UNS8) sub-index parameter 2
(UNS32) size data parameter 2
(DOMAIN) data parameter 2
....
(UNS16) index parameter n
(UNS8) sub-index parameter n
(UNS32) size data parameter n
(DOMAIN) data parameter n
}
    So the binary value stream to configure heartbeat producer time must be :
010000001710000020000000e803
    The slave node is configured just before the Master entering in Pre_operational
state.

8.2)   gene_SYNC_HCS12 :

This is a simple CANopen node that only send cyclic SYNC message. It
demonstrate implementation on HCS12 based board.

8.3)   kerneltest :

Example based on TestMasterSlave slightly modified to suit kernel space req-
uisites. It will do the same as TestMasterSlave but in kernel space sending
kernel messages (displayed by dmesg for example). It is designed as exter-
nal kernel module implemented as character device. There is a shell script
called 'insert.sh', which will insert the module and create a new device file
/dev/canf_ktest (used to sending commands to module). To actual send-
ing commands you can use simple console named 'canf_ktest_console'. The
module is dependent on a another separate module 'canfestival.ko' imple-
menting CanOpen stack which exports requisite functions. Canfestival.ko
module is then dependent on CAN card driver module, by default CAN
virtual driver will be loaded. After installing modules (make install), all
dependencies are solved automatically by kernel. To run the example type:

`sh run.sh` It will insert required modules, start console, and after quitting console it'll remove modules from kernel.

8.4)   TestMasterMicroMod

```
****************************************************************
   *  TestMasterMicroMod                                      *
   *                                                          *
   *  A simple example for PC.                                *
   *  A CANopen master that control a MicroMod module:          *
   *  - setup module TPDO 1 transmit type                     *
   *  - setup module RPDO 1 transmit type                     *
   *  - setup module hearbeatbeat period                      *
   *  - disable others TPDOs                                  *
   *  - set state to operational                             *
   *  - send periodic SYNC                                   *
   *  - send periodic RPDO 1 to Micromod (digital output)    *
   *  - listen Micromod's TPDO 1 (digital input)             *
   *  - Mapping RPDO 1 bit per bit (digital input)           *
   *                                                          *
   *   Usage:                                                 *
   *   ./TestMasterMicroMod  [OPTIONS]                        *
   *                                                          *
   *   OPTIONS:                                               *
   *      -l :  Can library ["libcanfestival_can_virtual.so"]   *
   *                                                          *
   *    Slave:                                                *
   *      -i :  Slave Node id format [0x01 , 0x7F]              *
   *                                                          *
   *    Master:                                               *
   *      -m :  bus name ["1"]                                  *
   *      -M :  1M,500K,250K,125K,100K,50K,20K,10K            *
   *                                                          *
   ****************************************************************
```

9 -   Developing a new node

Using provided examples as a base for your new node is generally a good idea. You can also use the provided *.od files as a base for your node object dictionary.

Creating a new CANopen node implies to define the Object Dictionary of this node. For that, developer has to provide a C file. This C file contains the definition of all dictionary entries, and some kind of index table that helps the stack to access some entries directly.

9.1)   Using Dictionary Editor GUI

The Object Dictionary Editor is a WxPython based GUI that is used to create the C file needed to create a new CANopen node.

9.1.1)   Installation and usage on Linux

You first have to download and install Gnosis XML modules. This is automated by a Makefile rule.

```
cd objdictgen
make
```
Now start the editor.
```
python objdictedit.py [od files...]
```

9.1.2)   Installation and usage on Windows

Install Python (at least version 2.4) and wxPython (at least version 2.6.3.2).

Cygwin users can install Gnosis XML utils the same as Linux use. Just call make.
```
cd objdictgen
make
```
Others will have to download and intall Gnosis XML by hand :
```
Gnosis Utils:
http://freshmeat.net/projects/gnosisxml/
http://www.gnosis.cx/download/Gnosis_Utils.More/Gnosis_Utils-1.2.1.win32-py24.exe
Get latest version.
```
Download CanFestival archive and uncompress it. Use windows file explorer to go into CanFestival3\objdicgten, and double-click on objdictedit.py.
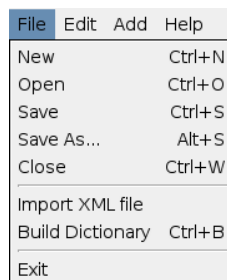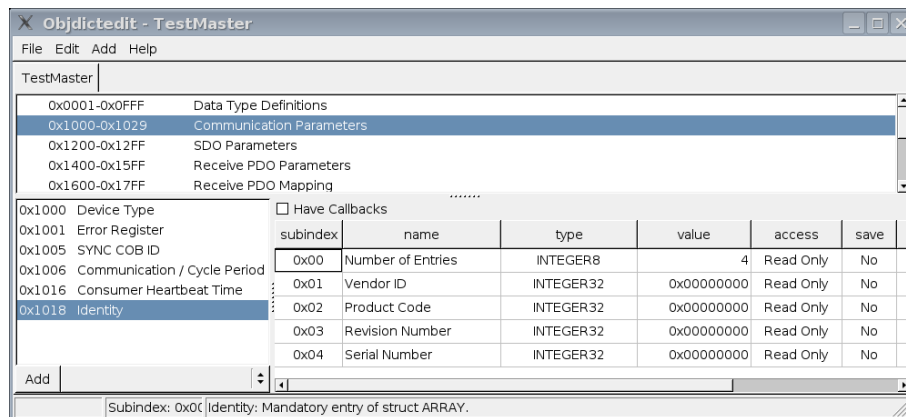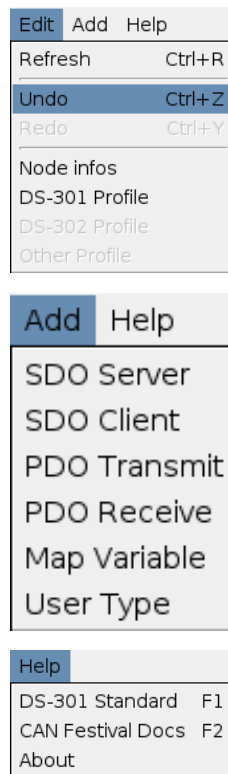
9.1.3)   About

The Object Dictionary editor GUI is a python application that use the Model-View-Controller design pattern. It depends on WxPython to display view on any supported platform.

9.1.4)   Main view

Top list let you choose dictionary section, bottom left list is the selected index in that dictionary, and bottom right list are edited sub-indexes.

Edit	Add	Help	
Refresh			Ctrl+R
Undo			**Ctrl+Z**
Redo			Ctrl+Y
Node infos			
DS-301 Profile			
DS-302 Profile			
Other Profile			

Add	Help
SDO Server	
SDO Client	
PDO Transmit	
PDO Receive	
Map Variable	
User Type	

Help	
DS-301 Standard	F1
CAN Festival Docs	F2
About	

9.1.5)   New node

Edit your node name, ID and type. Choose your inherited specific profile.

Create a new Node

Name :
MyNode

Node ID :
3

Type :
master

Profile
DS-401

Network
○ None
○ Node Guarding
⦿ Heartbeat

Options :
☐ DS-302 Profile
☑ Generate SYNC
☐ Emergency suppor
☐ Save Configuration
☐ Store EDS
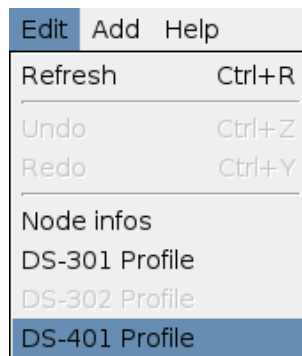
✗ Annuler   ⬟ Valider

9.1.6)   Node info

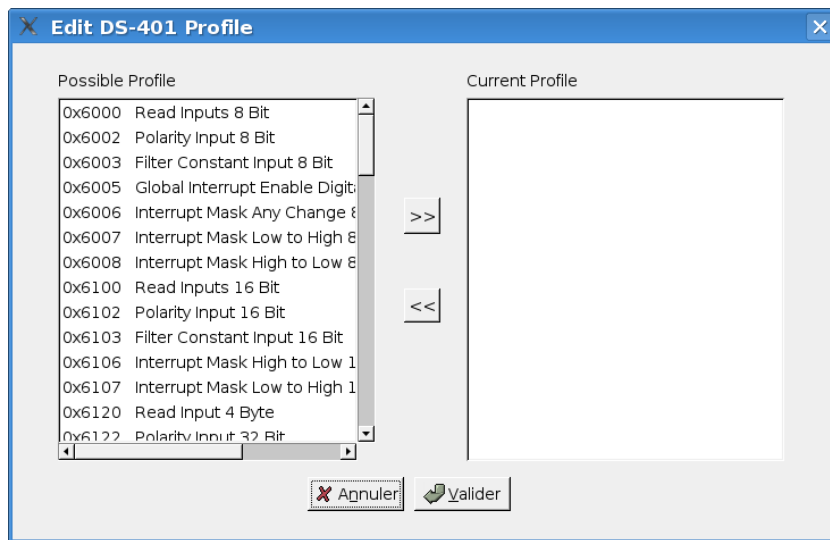Edit your node name, ID and type.
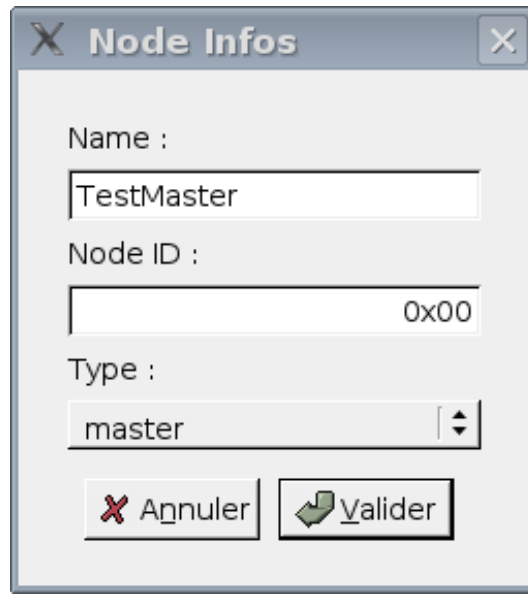
9.1.7)   Profile editor

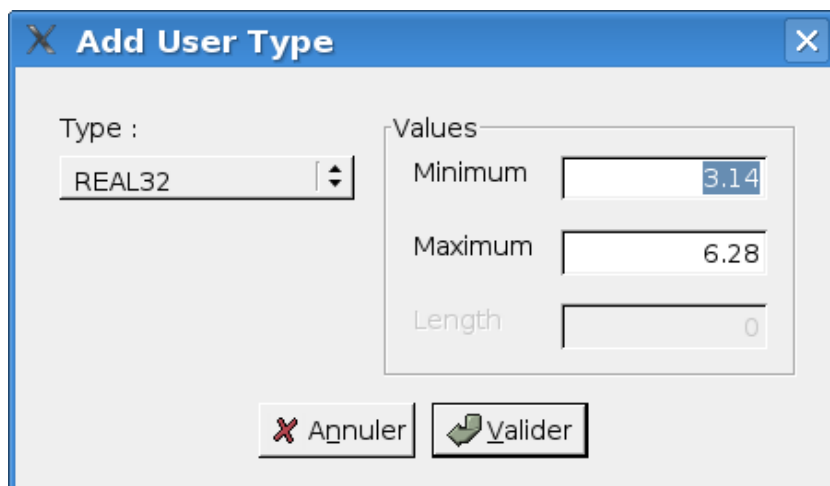Chose the used profile to edit.

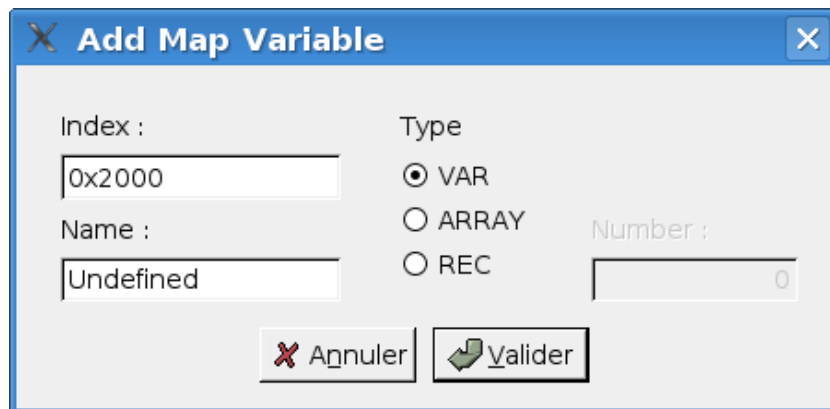Pick up optional chosen profile entries.

9.1.8) User types

Use User Types to implement value boundaries, and string lentgth
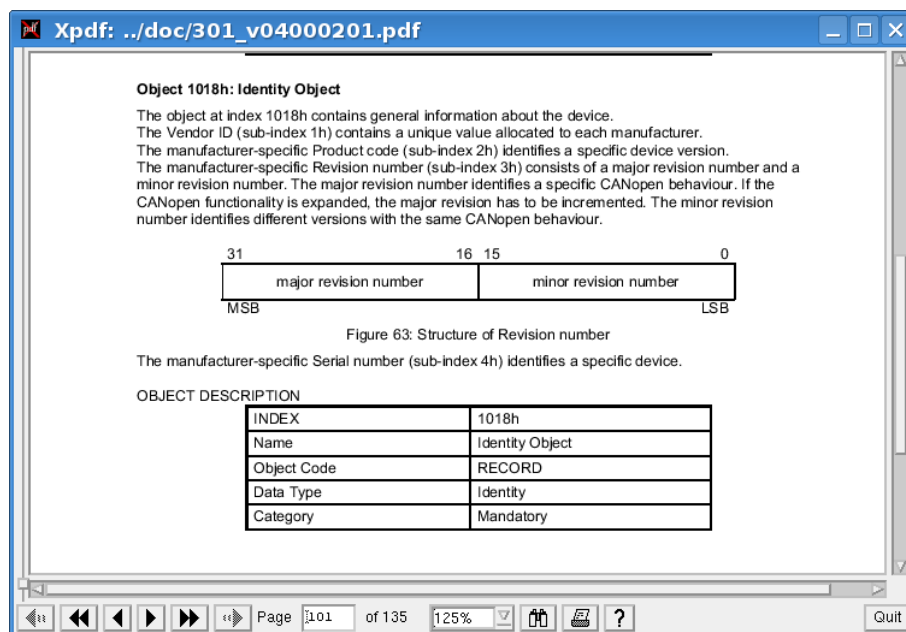

9.1.9) Mapped variable

Add your own specific dictionary entries and associated mapped variables.
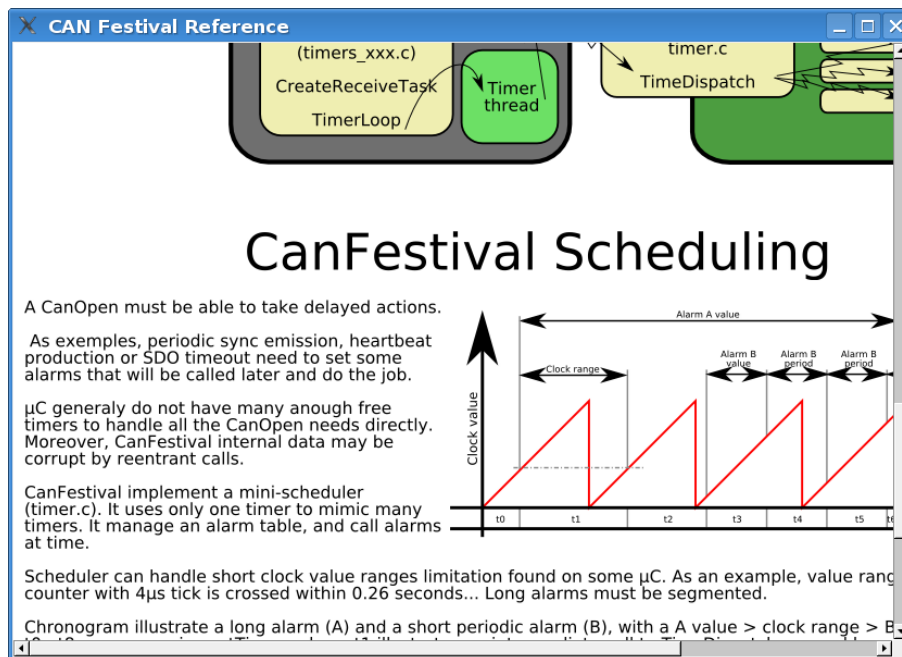
9.1.10)   Integrated help

Using F1 key, you can get context sensitive help.

In order to do that, official 301_v04000201.pdf file must be placed into doc/ directory, and xpdf must be present on your system.

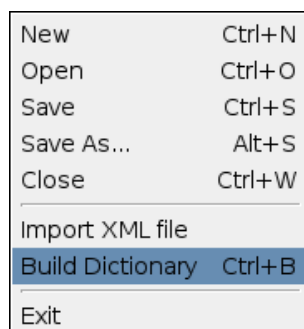F2 key open HTML CanFestival help.

9.2)   Generating the object Dictionary

Once object dictionary has been edited and saved, you have to generate object dictionary C code for your CanFestival node.

9.2.1)   With GUI

Menu entry "File/Build Dictionary".

Choose C file to create or overwrite. Header file will be also created with the same prefix as C file.

9.2.2)   With command line

```
Usage of objdictgen.py :
python objdictgen.py XMLFilePath CfilePath
```

10 -   FAQ

10.1)   General

10.1.1)   Does the code compiles on Windows ?

Yes, with both Cygwin and Visual Studio C++.

Because CANopen layer is coded with C, put a compilation option /TC or /TP if you plan to mix C++ files. See the MSDN documentation about that.

10.1.2)   How to fit the library to an other microcontrôler ?

First, be sure that you have at least 40K bytes of program memory, and about 2k of RAM.

You have to create target specific interface to HW resources. Take model on bundled interfaces provided in drivers/ and create your own interface. You also have to update Makefile.in files for target specific cflags and options. Choose –target= configure switch to compile your specific interface.

You are welcome to contribute-back your own interfaces! Other Canfestival users will use it and provide feedback, tests and enhancements.

10.1.3)   Is CanFestival3 conform to DS301 v.4.02 ?

Thanks to Philippe Foureys (IUT of Valence), a slave node have been tested with the National Instrument CANopen Conformance Test. It passed the test with success.

Some very small unconformity have been found in very unusual situations, for example in the SDO code response to wrong messages.

10.2)   LINUX

10.2.1)   How to use a Peaksystem CAN board ?

Just install peak driver and then compile and install Canfestival. Peak driver is detected at compile time.

10.2.2)   How to use an unsupported CAN board ?

You have to install the specific driver on your system, with necessary libs and headers.

Use can_peak.c/h or can_virtual.c/h as an example, and adapt it to your driver API.

Execute configure script and choose --can=mydriver

10.3)   Win32

Compatibility:

1. Code was compiled MS VisualStudio 2003.NET and VisualStudio 2005.NET for WindowsXP with ANSI and UNICODE configurations and for WindowsCE 5.0.

2. Some preliminary testing was done, but not enough to be used in mission critical projects.

Additional Features:

1. Non-integral integers support implementation UNS24, UNS40, UNS48 etc.

2. When enable debug output with DEBUG_WAR_CONSOLE_ON or DEBUG_ERR_CONSOLE_ON, you can navigate in CanFestival source code by double clicking at diagnostic lines in VisualStudio.NET 200X Debug Output Window.

Custom size integral types such as INTEGER24, UNS40, INTEGER56 etc. have been defined as 64 bits integers. You will need to replace sizeof(TYPE) operators to sizeof_TYPE definitions in generated code, i.e. replace sizeof(UNS40) with sizeof_UNS40.

10.4)   HCS12

10.4.1)   Which board are you using ?

A T-board from elektronikladen with a MC9S12DP256 or MC9S12DG256.

**10.4.2)   Does the code compile with an other compiler than GNU
            gcc ?**

It is known to work with Metrowerks CodeWarrior. Here are some tips from
Philippe Foureys. :

a)   Interrupt functions

   i)   Code for GCC: `// prototype`
```
void __attribute__((interrupt))timer3Hdl(void):
// function
void __attribute__((interrupt))timer3Hdl(void){...}
```

   ii)   Code for CodeWarrior `// protoype`
```
void interrupt timer3Hdl(void);
// function
pragma CODE_SEG__NEAR_SEG_NON_BANKED
void interrupt timer3Hdl(void)
{...}
pragma CODE_SEG_DEFAULT
```

b)   Interrupt lock, unlock

   i)   Code for GCC `void unlock (void)`
```
{
__asm__ __volatile__("cli");
}
void lock (void)
{
unsigned short mask;
__asm__ __volatile__("tpa\n\tsei":"=d"(mask));
}
```

   ii)   Code for CodeWarrior `void unlock (void)`
```
{
__asm("cli");
}
void lock (void)
{
unsigned short mask;
```

```
__asm
{
tpa:tsei:"=d"(mask);
}
}
```

c)   Initialize function

i)   Code for GCC   void initCanHCS12 (void)

```
{
//Init the HCS12 microcontroler for CANopen
initHCS12();
// Init the HCS12  CAN driver
const canBusInit bi0 = {
0,    /* no low power              */
0,    /* no time stamp             */
1,    /* enable MSCAN              */
0,    /* clock source :  oscillator (In fact, it is not used)   */
0,    /* no loop back              */
0,    /* no listen only            */
0,    /* no low pass filter for wk up */
CAN_Baudrates[CAN_BAUDRATE_250K],
{
0x00,   /* Filter on 16 bits.
See Motorola Block Guide V02.14 fig 4-3 */
0x00, 0xFF, /* filter 0 hight accept all msg       */
0x00, 0xFF, /* filter 0 low accept all msg         */
0x00, 0xFF, /* filter 1 hight filter all of  msg */
0x00, 0xFF, /* filter 1 low filter all of  msg     */
0x00, 0xFF, /* filter 2 hight filter most of  msg */
0x00, 0xFF, /* filter 2 low filter most of  msg   */
0x00, 0xFF, /* filter 3 hight filter most of  msg */
0x00, 0xFF, /* filter 3 low filter most of  msg   */
}
};
```

ii)   Code for CodeWarrior   void initCanHCS12 (void)

```
{
//Init the HCS12 microcontroler for CANopen
initHCS12();
```

```
// Init the HCS12  CAN driver
const canBusInit bi0 = {
0,    /* no low power                 */
0,    /* no time stamp                */
1,    /* enable MSCAN                 */
0,    /* clock source :  oscillator (In fact, it is not used)   */
0,    /* no loop back                 */
0,    /* no listen only               */
0,    /* no low pass filter for wk up */
{
1, /* clksrc */
3, /* brp    */
0, /* sjw    */
0, /* samp   */
1, /* tseg2  */
12,/* tseg1  */
},
{
0x00,    /* Filter on 16 bits.
See Motorola Block Guide V02.14 fig 4-3 */
0x00, 0xFF, /* filter 0 hight accept all msg      */
0x00, 0xFF, /* filter 0 low accept all msg        */
0x00, 0xFF, /* filter 1 hight filter all of  msg */
0x00, 0xFF, /* filter 1 low filter all of  msg    */
0x00, 0xFF, /* filter 2 hight filter most of  msg */
0x00, 0xFF, /* filter 2 low filter most of  msg   */
0x00, 0xFF, /* filter 3 hight filter most of  msg */
0x00, 0xFF, /* filter 3 low filter most of  msg   */
}
};
```

10.4.3)   Does the code works in banked memory ?

No. Today it seems that the port of gcc is bogged for using the banked memory. So, unfortunately, we are limited to 48 Kbytes of memory code.

10.4.4)   What GCC version are you using ?

We are using the stable RPM release 2.2 :

1. GNU Gcc 3.0.4. Build 20030501

2. Newlib 1.10.0 Build 20030421

3. GNU Binutils 2.12.1 Build 20030427

11 - Documentation resources

a) CIA : Can in Automation
   Many documentation on CANopen.
http://www.can-cia.de

b) Resources and training in CANopen
   http://www.esacademy.com

c) Elektronikladen HCS12 T-board
   http://www.elektronikladen.de/en_hcs12tb.html

d) Gnu gcc compiler for HC12
   http://m68hc11.serveftp.org/m68hc11_port.php

e) Motorola documentation on HC12
   http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MC9S12DP256

f) Lauterbach debugger for HC12
   http://www.lauterbach.com

g) Python language
   http://www.python.org

12 -   About the project

12.1)   Contributors

Unité mixte de recherche INRETS-LCPC
sur les Interractions Véhicule-Infrastructure-Conducteur
14, route de la minière
78000 Versailles
FRANCE
Tel : +33 1 40 43 29 01
http://www.inrets.fr/ur/livic
Contributors : Francis DUPIN
Camille BOSSARD
Laurent ROMIEUX

LOLITECH
204, rue du Haut du Pin
88470 Saint-Michel sur Meurthe
FRANCE
Tel : +33 3 29 52 95 67
http://www.lolitech.fr
Contributors : Edouard TISSERANT (Original author)
Laurent BESSARD

Many thanks to the other contributors for their great work:
 Raphael ZULLIGER
 David DUMINY (sté A6R)
 Zakaria BELAMRI

12.2)   Getting support

Send your feedback and bug reports to canfestival-devel@lists.sourceforge.net.

For commercial support, training and specific integration and developments, please ask LOLITECH (see contributors).

12.3)   Contributing

You are free to contribute your specific interfaces back to the project. This way, you can hope to get support from CanFestival users community.

Please send your patch to canfestival-devel@lists.sourceforge.net.

Feel free to create some new predefined DS-4xx profiles (*.prf) in objdict-gen/config, as much as possible respectful to the official specifications.

12.4)   License

All the project is licensed with LGPL. This mean you can link CanFestival with any code without being obliged to publish it.

```
#This file is part of CanFestival, a library implementing CANopen
Stack.
#
#Copyright (C): Edouard TISSERANT, Francis DUPIN and Laurent BESSARD
#
#See COPYING file for copyrights details.
#
#This library is free software; you can redistribute it and/or
#modify it under the terms of the GNU Lesser General Public
#License as published by the Free Software Foundation; either
#version 2.1 of the License, or (at your option) any later version.
#
#This library is distributed in the hope that it will be useful,
#but WITHOUT ANY WARRANTY; without even the implied warranty of
#MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
#Lesser General Public License for more details.
#
#You should have received a copy of the GNU Lesser General Public
#License along with this library; if not, write to the Free Software
#Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307
USA
```