STEP BY STEP RTAI TARGET FOR BEGINNERS

STEP 1 - Install Linux

Any recent distribution is supposed to be ok, however
any particular machine+kernel+compiler+distribution
combination has its own specific problems.

I would suggest using a distribution that was released before the
last rtai version. I am currently using Fedora Core 3 since i have
found less problems with older gcc (version less than 4) compilers.

On Fedora, be sure to install both the 'x' and 'qt' development packages.

On Ubuntu, you need to install several development packages, which
can be done, as root, with the following commands:

```
$ apt-get update
$ apt-get install patch
$ apt-get install build-essential
$ apt-get install kernel-package
$ apt-get install gcc
$ apt-get install libncurses5
$ apt-get install libncurses5-dev
$ apt-get install libqt3-mt-dev
$ apt-get install cvs
$ apt-get install subversion
$ apt-get install libxmu-dev
$ apt-get install libxi-dev
```

In what follows, the shell prompt sign is indicated with the symbol "$",
and a root user is always assumed. You can either log in as root or just
open a terminal and use the su command and then and type the root password
to get root privileges:

```
$ su
```

Note that there is also a good guide on how to install RTAI on Ubuntu 7.10,
prepared by Arno Stienen and available at:
https://www.rtai.org/RTAILAB/RTAI-UbuntuGutsy-Matlab.txt

If you are running linux inside a vmware virtual machine make
sure the boot loader passes the option vdso=0 to the kernel
otherwise the machine tends to hang up.

Finally, keep in mind that older distributions based on 2.4 or 2.2 kernels
require different procedures for most of the points below.

STEP 2 - Download and unpack RTAI

Go to http://www.rtai.org/   and look for the newest version (3.2 or
greater).
We are still using 3.2 since it is the one that we have found to work
flawlessly
under all circumstances for everything we do, but this document has been
tested
several times until version 3.6

You should unpack your RTAI source in /usr/src, (standard /usr/src use),
so AS ROOT, download the rtai-x.x.tar.bz2 file into /usr/src
and then unpack it using the following commands in a terminal window :

```
$ cd /usr/src
$ tar -xjvf rtai-x.x.tar.bz2
```

Create a symbolic link to the new rtai directory,
since it will be useful later:

```
$ rm -f rtai
$ ln -fs rtai-x.x rtai
```

After that, have a look in the directory
/usr/src/rtai/base/arch/i386/patches/
to locate the patches for the supported kernels.

You can have a look at the README.INSTALL file in the
rtai root folder for advanced installation options.

IMPORTANT: If you are using an older version or RTAI (e.g 3.2 to 3.5),
and a version of MATLAB greater than 7.0.1, then you have to download
and unpack the LAST version of rtai somewhere, (e.g. in /usr/tmp),
and then overwrite the OLD rtai.tmf file in /usr/src/rtai-x.x/rtai-lab/matlab
with the NEW rtai.tmf file found in /usr/tmp/rtai-n.n/rtai-lab/matlab

STEP 3 - Download and unpack a new Kernel

Never use the kernel that comes with the distribution,
always download a new kernel from http://www.kernel.org/

Make sure that the kernel is actually supported by RTAI
that is make sure that a patch exist for that kernel.
The first three numbers in the kernel have to match the
first three numbers in the patch. If more than one patch
exist with the same first three numbers pick the last one.
If more than one kernel exist with the same first three numbers
it is safer to pick the base version (i.e. the one that does not
have the fourth number).

For example, the following patches exist for the 2.6.15 kernels
in the /usr/src/rtai/base/arch/i386/patches/ directory:
hal-linux-2.6.15-i386-1.1-03.patch and hal-linux-2.6.15-i386-1.2-00.patch
so the last patch should be used, with the base kernel 2.6.15, although
it should work fine with kernels until 2.6.15.7 too.

Again, source code traditionally go in /usr/src, so, as root,
download the kernel (linux-x.x.xx.tar.bz2) into /usr/src
and do the following:

```
$ cd /usr/src
$ tar -xjvf linux-x.x.xx.tar.bz2
```

You need to set a symbolic link to the new linux root folder
since it's needed later:

```
$ rm -f linux
$ ln -fs linux-x.x.xx linux
```

STEP 4 - Patch your Kernel

Again, as root (assuming an x86 architecture):

```
$ cd /usr/src/linux
$ patch -p1 < /usr/src/rtai/base/arch/i386/patches/hal-linux-x.x.xx_rx.patch
```

STEP 5 - Configure your Kernel

Again, as root:

```
$ cd /usr/src/linux
```

It is a good idea to copy the distribution .config file into the linux folder,
so one could start configuring something that already works:

```
$ cp -f /boot/config-x.x.xx-x.xxx-generic .config
```

However, that is not strictly necessary,
so you can safely skip the previous step.
Then, to configure the kernel do the following:

```
$ make xconfig (or make menuconfig or make config)
```

In the code maturity level options, set the "prompt for development and/or
incomplete code/drivers" option to 'yes'.
In the "loadable module support" section,
make sure that the "Enable loadable module support" is set to 'yes',
and set the "Module Versioning support" is to 'no'.
In the "Processor Type and Features" make sure that
the "interrupt pipeline" option, (aka IPIPE) is set to 'yes'.
In older kernels disable the "use register arguments" if it is there.
Rtai versions older than 3.6 need to have the "Preemptible Kernel"
disabled, that is set to "No Forced Preemption (Server)".

Finally, make sure that the "/proc file system support", under the
"Pseudo filesystems" subsection of the "File systems" section is set to yes.

Turning off the Power management options it's also a good idea.

Remember to save the kernel configuration on exit.

STEP 6 - Compile and install your Kernel

Again, as root :

```
$ cd /usr/src/linux
```

then compile the kernel:

```
$ make clean && make && make modules_install && make install
```

if everything goes fine (i.e. no errors encountered),
the new kernel image, together with its system map file, should

appear in the boot directory.

If you are using Fedora, the boot loader configuration file /etc/grub.conf will be automatically changed to allow booting with the new kernel and, if required, the new initial ram disk.
Note that you need to change the number in the "default" line (which starts from 0) to boot with the new kernel by default.

On Ubuntu, you have to create the initial ram disk with the command:

```
$ update-initramfs -ck x.x.xx
```

where x.x.xx is the directory under /lib/modules that contains the modules for the new kernel.

Then you have to edit the /boot/grub/menu.lst, comment the "hiddenmenu" option, set the timeout to sonmething like 10 seconds, and physically insert the lines relative to the new kernel, resulting in something like this:

```
title       Ubuntu 7.10, kernel 2.6.20
root        (hd0,0)
kernel      /boot/vmlinuz-2.6.20 root=UUID=xx-whatever ro quiet splash
initrd      /boot/initrd.img-2.6.20
```

As an alternative, the make-kpkg package lets you compile and install the kernel automatically, so you can give it a try too.

Now reboot:

```
$ /sbin/reboot
```

(make sure to reboot into your new rtai-patched-kernel).

If everything works fine, it is a good idea to copy the configuration file for future reference, so as root:

```
$ cd /usr/src/linux
$ cp -f .config config-my-rtlx-yy.txt
```

It is also a good idea to change the name of the kernel image, ramdisk image, and system.map so that they won't be overwritten the next time you compile a new kernel, so as root:

```
$ cd /boot

$ cp -f vmlinuz-x.x.xx my-rtlx-yy
$ cp -f initrd-x.x.xx.img my-inrd-yy.img
$ cp -f System.map-x.x.xx System.map-my-rtlx-yy
```

Under Fedora, assuming grub is your boot loader,
you can copy the automatically added entry relative to the new kernel,
(which is usually the first one) to a new entry that won't be overwritten
when you compile and install a new kernel.
For example this is a valid entry for the grub bootloader in
/etc/grub.conf :

```
title my-rtlx-xx (2.x.xx-rtai)
    root (hd0,0)
    kernel /my-rtlx-xx ro root=/dev/VolGroup00/LogVol00 rhgb quiet
    initrd /my-inrd-xx.img
```

    reboot again chosing the entry you just added to make sure everything
    works as it should.

```
STEP 7 - Download and install the MESA library
----------------------------------------------
```

    Note that you might be able to skip this step and the next one
    (EFLTK) if you are NOT planning to install RTAILab later on.
    However, if this is your first time installation i suggest
    you try to follow these steps and install RTAILab.

    At this point, if your system already has some pre-installed
    version of the mesa library, the thing that makes most sense
    is to try to skip this step, and only later on, if step 8
    fails due to a Mesa library that's incompatible with efltk,
    return here and install MesaLib-7.0.2.tar.gz, in any case,
    whatever you do, do not uninstall the preinstalled MesaLib.

    Download MesaLib-7.0.2.tar.gz (from www.mesa3d.org)
    in a temporary directory (/tmp), then, as root :

```
    $ cd /usr/local/src
    $ tar -jxf MesaLib-7.0.2.tar.bz2
    $ cd Mesa-7.0.2
    $ make realclean
```

    For Mesalib 7.0.2, create the file src/glw/glw.pc.in
    and insert the following lines inside it
    (skip this step for 7.0.3, or for 6.2
     continuing at the make linux-x86):

```
    prefix=@INSTALL_DIR@
    exec_prefix=${prefix}
    libdir=${exec_prefix}/@LIB_DIR@
    includedir=${prefix}/include

    Name: glw
    Description: Mesa OpenGL widget library
    Requires: gl
    Version: @VERSION@
    Libs: -L${libdir} -lGLw
    Cflags: -I${includedir}
```

    Then compile everyting:

```
    $ make linux-x86-static
```

    and install:

```
    $ make install
```

    I do not suggest to install and use the "checkinstall" tool,
    because it does not work flawlessly with efltk,
    and adds unnecessary installation steps.

    After the installation reboot and make sure everything works.

```
STEP 8 - Download and install the EFLTK package
-----------------------------------------------
```

```
    As root, download EFLTK in /usr/local/src

    $ cd /usr/local/src
    $ svn co https://ede.svn.sourceforge.net/svnroot/ede/trunk/efltk
    $ cd efltk
    $ autoconf

    $ ./configure --disable-mysql --disable-unixODBC
    $ ./emake
    $ ./emake install

    Add the line /usr/local/lib/ to the file /etc/ld.so.conf
    the entries in the file should look more or less like this:

    include ld.so.conf.d/*.conf
    /usr/local/lib

    then run ldconfig to update the library database:

    $ /sbin/ldconfig

    Again, reboot and make sure everything works fine.


STEP 9 - Configure and install RTAI
-----------------------------------

    Again, as root :

    $ cd /usr/src/rtai
    $ make config (or make menuconfig of make xconfig)

    In the Machine Menu adjust the number of processors.
    Say yes to rtai lab, and leave the efltk folder to /usr/local
    Do not include comedi support at this time even if you plan to use it later.
    Do not change any other default choice unless you know what you are doing.

    $ make
    $ make install

    reboot :

    $ /usr/bin/reboot


STEP 10 - Test the RTAI installation
------------------------------------

    Under Ubuntu you have to change the first line of the file
    '/usr/realtime/bin/rtai-load' from '#!/bin/sh' into '#!/bin/bash',
    or else the system could hang.

    In RTAI versions prior to 3.4 there could be errors opening FIFOs
    due to the fact that entries like /dev/rtf/0 are not persistent,
    so they get deleted after a reboot. If that's the case,
     you should include code to recreate those entries in
    /etc/rc.d/rc.local to make sure that they are recreated everytime,
    see http://www.captain.at/rtai-error-opening-dev-rtf.php for example.

    Then, as root, (make sure you are running the patched kernel) do :
```

```
    $ cd /usr/realtime/testsuite/user/latency
    $ ./run (use ctrl-c to stop it)

    $ cd /usr/realtime/testsuite/user/preempt
    $ ./run (use ctrl-c to stop it)

    $ cd /usr/realtime/testsuite/user/switches
    $ ./run (use ctrl-c to stop it)

    $ cd /usr/realtime/testsuite/kern/latency
    $ ./run (use ctrl-c to stop it)

    $ cd /usr/realtime/testsuite/kern/preempt
    $ ./run (use ctrl-c to stop it)

    $ cd /usr/realtime/testsuite/kern/switches
    $ ./run (use ctrl-c to stop it)
```

make sure that there are no errors.

STEP 11 - Download, unpack and install Comedi and Comedilib
```
-----------------------------------------------------------
```

If you are new to Linux and RTAI i strongly suggest
that you SKIP this step since you do not need
comedi or comedilib to use or learn RTAI.

However, if you plan to use comedi, this is the right
time to download and install comedi and comedilib.
The reference web site is http://www.comedi.org/

You should unpack the newest source in /usr/src,
so AS ROOT, dowload both comedi-x.x.xx.tar.gz and
comedilib-x.x.xx.tar.gz files into /usr/src
and then unpack them using the following commands (still as root):

```
    $ cd /usr/src
    $ tar -xzvf comedi-x.x.xx.tar.gz
    $ tar -xzvf comedilib-x.x.xx.tar.gz
```

Create a symbolic link to comedi since it will be useful later:

```
    $ rm -f comedi
    $ ln -fs comedi-x.x-xx comedi
```

If your comedi version is 0.7.70 (or older) then you should manually
edit the /usr/src/linux/.config file and add the line:
CONFIG_RTHAL=Y

At this point you are ready to configure and install
both comedilib and comedi:

```
    $ cd /usr/src/comedilib-x.x.xx
    $ ./configure --sysconfdir=/etc
    $ make
    $ make install
    $ make dev

    $ cd /usr/src/comedi
    $ ./configure --with-linuxdir=/usr/src/linux --with-rtaidir=/usr/realtime
    $ make
    $ make install

    $ cp include/linux/comedi.h /usr/include/
    $ cp include/linux/comedilib.h /usr/include/

    $ mkdir /usr/local/include/linux
    $ ln -s /usr/include/comedi.h /usr/local/include/linux/comedi.h
    $ ln -s /usr/include/comedilib.h /usr/local/include/linux/comedilib.h
```

Reboot and make sure everything works fine.

At this point, you have to add comedi to the rtai configuration, so, as root:

```
    $ cd /usr/src/rtai
    $ make config (or make menuconfig of make xconfig)
```

In the Add-ons Menu select Comedi support over LXRT, and specify the comedi installation directory, (typically /usr/local, in any case it should contain lib/libcomedi.a include/linux/comedilib.h and include/linux/comedi.h Then recompile and install rtai:

```
    $ make
    $ make install
```

Reboot and re-test rtai (Step 10) to make sure everything is fine.

STEP 12 - Copy the rtai-lab folder into the current MATLAB installation

It is assumed that you have a working MATLAB installation that includes simulink and Real Time Workshop (i currently use version 7.8).
A few extra step are required if your MATLAB installation is not on the same system where you installed rtai. Typically people have MATLAB installed under a Windows system. A Windows version of MATLAB installed under Wine, counts as an installation on a different windows system.

The thing to do at this point is to copy the entire content of the folder (typically) /usr/src/rtai-3.6/rtai-lab/matlab into Matlab_Root\rtw\c\
rtai
(typically the Matlab_Root is something like C:\Matlab701\ or C:\Program Files\Matlab\R2007a\ for Windows and /usr/local/matlab for Unix).

STEP 13 - Run setup.m and create mex files

After you copied the rtai-lab folder you should open MATLAB and run the Setup.m file that is included in the rtai-lab folder.
This basically adds the new folder to the MATLAB path.

After you have run setup.m you should be able to access the simulink library RTAI Devices from the Simulink browser.

In older rtai versions you would now need to create mex (MATLAB Executable) files for every s-function but this is now done automatically.

STEP 14 - Edit rtai.tmf

If you are using a MATLAB installation (for Unix) on the same Linux
system in which you installed RTAI, then you can skip the following
and jump to the next step, otherwise you have to modify the rtai.tmf file
in the rtai-lab folder, specifically, you must modify the following lines :

```
MATLAB_ROOT     = |>MATLAB_ROOT<|
---------------------------------
```

Here you have to replace |>MATLAB_ROOT<| with the path of a Linux
folder in which you will later copy some MATLAB related files.
Typically you will replace |>MATLAB_ROOT<| with /usr/local/matlab,
and only later you will actually create the /usr/local/matlab folder
in the Linux_with_RTAI filesystem and copy stuff within it (Step 16).

```
COMPUTER        = |>COMPUTER<|
------------------------------
```

Here you have to replace |>COMPUTER<| with GLNX86
(assuming Linux is innstalled on a x86 architecture).

You also have to modify the entries LINUX_HOME,
RTAIDIR and COMEDI_HOME, to make sure they refer
to the paths where Linux, rtai, and comedi are installed,
(typically they are /usr/src/linux, /usr/realtime
and /usr/src/comedi respectively).

Finally, into the "Rules" section delete the following lines:

```
|>START_EXPAND_RULES<|%.o : |>EXPAND_DIR_NAME<|/%.c
      gcc -c $(CFLAGS) $<

|>END_EXPAND_RULES<|
```

otherwise, later on, the generated make files (*.mk)
will contain the (windows) relative paths of the custom
s-functions used in the scheme, therefore causing the make
process to fail miserably.

STEP 15 - Generating code from a simple example

Copy the file Matlab_Root\rtw\c\rtai\examples\test.mdl
somewhere else, for example in Matlab_Root\work,
or in My Documents\MATLAB, then within MATLAB,
move to that folder, open the file with simulink and run it.
It should run without problems.

At this point you are ready to generate code from it.
Open the simulation menu, configuration parameters,
Real-Time Workshop, and make sure that the RTAI Target
is selected, (otherwise browse and select it).
Then click on Build, to generate code from the scheme.

At this point you should have the folder test_rtai
under your working directory. This folder contains
the code generated from the test simulink scheme.

If your simulink scheme uses some custom s-functions,
(which is not specifically the case of test.mdl but
it is indeed the normal case), then it is very important

at this point to remember to copy the C code of every custom s-functions used in the scheme inside the generated scheme_rtai folder.

Another important thing to remember about s-functions is to always USE LOWERCASE characters in the file name, this is because the file names of the s-functions will be converted to lowercase within the make file (*.mk), which does not make any difference under windows but causes missing file errors under unix if the original name contained uppercase characters.

STEP 16 - Copy the extern, rtw, and Simulink folders to Linux

If you are using a MATLAB installation (for Unix) on the same Linux system in which you installed RTAI, then you can skip this step.

In your Linux filesystem, create a folder that corresponds to the entry MATLAB_ROOT in your rtai.tmf file, (typically /usr/local/matlab), then copy within it the three folders Matlab_Root\extern, Matlab_Root\rtw and Matlab_Root\Simulink from the current MATLAB installation.

Make sure all the c files in the folder Matlab_Root\rtw\c\libsrc have been transfered, since they all will be needed later, and for mysterious reasons some of them have a tendency to get lost.

STEP 17 - Copy, compile and run the generated code

Copy the test_rtai folder somewhere in the linux filesystem, (e.g. usr/local/src/tests/test_rtai) and then, from within the test_rtai folder, as root :

```
make -f test.mk
```

at this point, if the compilation proceeds without errors you should get the executable test in the parent folder, so move to the parent folder, insert the required modules:

```
sync
insmod /usr/realtime/modules/rtai_hal.ko
insmod /usr/realtime/modules/rtai_lxrt.ko
insmod /usr/realtime/modules/rtai_fifos.ko
insmod /usr/realtime/modules/rtai_sem.ko
insmod /usr/realtime/modules/rtai_mbx.ko
insmod /usr/realtime/modules/rtai_msg.ko
insmod /usr/realtime/modules/rtai_netrpc.ko ThisNode="127.0.0.1"
sync
```

at this point run the test:

```
./test -v -f 5
```

the process should run for 5 seconds.

Then to test also xrtailab:

```
./test &
```

starts the process in wait mode,

```
/usr/realtime/bin/xrtailab &
```

starts the xrtailab window, from that window you can then
actually start the real time process, log the data,
see the plots from the scopes in the simulink schemes as the
process runs, and then stop the process.