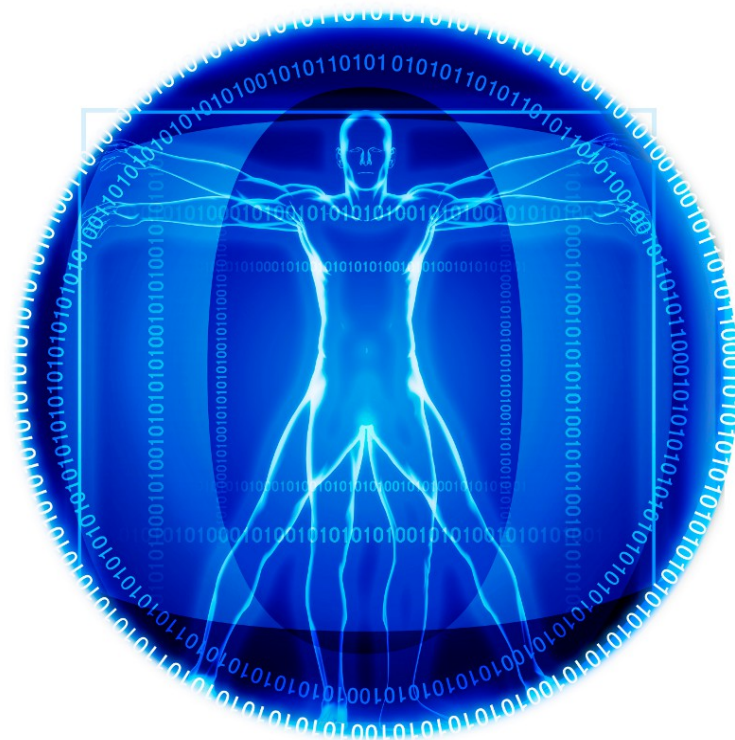


Cybernetics Oriented Programming

(CYBOP)

Beginners Tutorial



04.04.2013

Tim Illner<tim.illner@it2010.ba-leipzig.de>

Index Of Contents

1	Introduction.....	4
2	Installing CYBOP.....	4
2.1	Download for Linux distributions.....	4
3	Compiling with CYBOI.....	6
3.1	preparing to run make command.....	6
3.2	make command.....	7
3.3	run examples.....	8
3.3.1	HelloWorld.....	8
3.3.2	addition.....	8
3.3.3	addition_dynamic_model.....	9
3.3.4	addition_dynamic_model_with_root.....	9
3.3.5	addition_static_model.....	9
3.3.6	addition_using_indices.....	9
3.3.7	addition_using_serialisation.....	9
3.3.8	character_encoding.....	10
3.3.9	counter.....	10
3.3.10	counter_static_model.....	10
3.3.11	counter_storage.....	10
3.3.12	double.....	11
3.3.13	exit.....	11
3.3.14	exit_cybol_file.....	11
3.3.15	gui.....	12
3.3.16	if-else.....	12
3.3.17	index_usage.....	13
3.3.18	programme_execution.....	13
3.3.19	shell_command_execution.....	13
3.3.20	shell_output.....	13
3.3.21	shell_output_branch.....	14

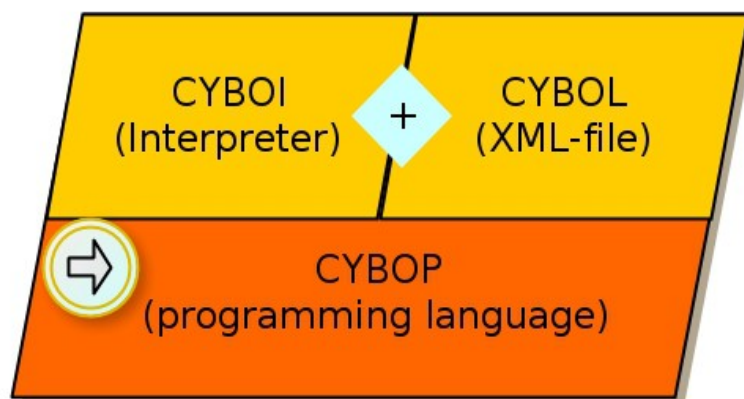
3.3.22 shell_output_file.....	14
3.3.23 shell_output_sequence.....	14
3.3.24 time_output_1.....	15
3.3.25 time_output_2.....	15
3.3.26 time_output_3.....	15
3.3.27 tui.....	16
3.3.28 ui_control.....	16
4upshot.....	17

1 Introduction

Cybernetics Oriented Programming (CYBOP) follows a new idea of software development. Its structure bases upon native concepts for intuitive understanding. CYBOP consists of two core elements: Cybernetics Oriented Language (CYBOL) and Cybernetics Oriented Interpreter (CYBOI).

CYBOL is a platform independent application programming language. As based on XML CYBOL is structured in tags and trees.

CYBOI instead is the appropriate interpreter to run in CYBOL written programs.



2 Installing CYBOP

There are two ways provided to download the source code of CYBOP. You can get the release directly from the GNU web page or use subversion (SVN). The most current release is cybop-0.13.0.tar.gz.

2.1 Download for Linux distributions

On this web page you will find all releases of CYBOP as tar.gz packages. <http://download.savannah.gnu.org/releases/cybop/>
Simply pick and download a tar.gz file and unpack it into a new created project directory (eg. "cybop").

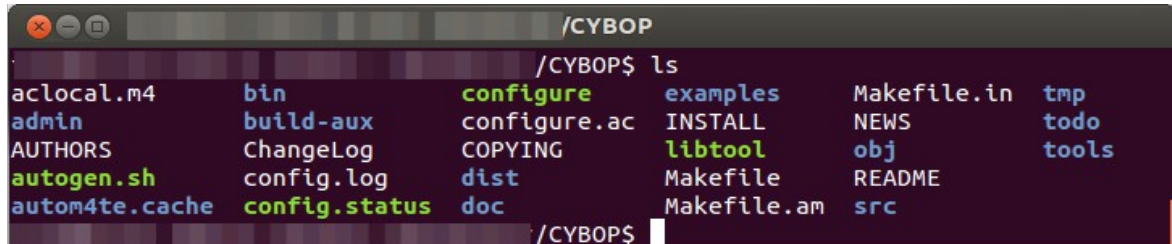
Make sure the required packages are installed. Open the Terminal and use the command "cd" to navigate to the location of the packed file. Now enter the following command line `tar -xvf cybop-0.13.0.tar.gz /[home]/[user]/[install_directory]` and the package will be unpacked to the target install directory.

To use SVN and the created repository which is highly recommended for later development. Therefore you switch to the desired directory using the Terminal and paste the following command line:

```
svn co svn://svn.savannah.nongnu.org/cybop/modulename
```

Instead of “*modulename*” you type “trunk”, typically.

Finally you should find the following files and folders in the directory



```

/CYBOP
/CYBOP$ ls
aclocal.m4      bin           configure     examples      Makefile.in  tmp
admin          build-aux    configure.ac  INSTALL       NEWS         todo
AUTHORS        ChangeLog   COPYING      libtool       obj          tools
autogen.sh     config.log  dist         Makefile     README
autom4te.cache config.status doc          Makefile.am  src
/CYBOP$
```

3 Compiling with CYBOI

Before you can start to compile it's recommended to check wheather all necessary packeages are already installed. Most of them are listed below:

autotools

libtool

xorg

xorg-dev

xlibs-dev

freeglut3

3.1 preparing to run make command

After that the autogen.sh script file in the CYBOP directory has to be exectuted. To make it executable, use the command `chmod +x autogen.sh` and run the script file after.

The output should look like the following

```
libtoolize: Consider adding `AC_CONFIG_MACRO_DIR([m4])' to configure.ac and
libtoolize: rerunning libtoolize, to keep the correct libtool macros in-tree.
libtoolize: Consider adding `-I m4' to ACLOCAL_AMFLAGS in Makefile.am.
running CONFIG_SHELL=/bin/bash /bin/bash ./configure --no-create --no-recursion
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
[...]
libtool: link: gcc -I/usr/include -DGNU_LINUX_OPERATING_SYSTEM -g -O2 -o cyboi cyboi.o
-lxcb -lpthread -IX11 -IGLU -IGL
make[2]: Verlasse Verzeichnis '/home/CYBOP/src/controller'
make[2]: Betrete Verzeichnis '/home/CYBOP/src'
make[2]: Für das Ziel »all-am« ist nichts zu tun.
make[2]: Verlasse Verzeichnis '/home/CYBOP/src'
make[1]: Verlasse Verzeichnis '/home/CYBOP/src'
make[1]: Betrete Verzeichnis '/home/CYBOP'
make[1]: Für das Ziel »all-am« ist nichts zu tun.
make[1]: Verlasse Verzeichnis '/home/CYBOP'
```

Next the configure script file has to be run. To do that simply repeat the steps of the description for running autogen.sh. The output should look like the following:

```
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
checking how to create a ustar tar archive... gnutar
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking Host cpu... x86_64
[...]
checking for setlocale... yes
checking for socket... yes
checking for strtol... yes
configure: creating ./config.status
config.status: creating Makefile
config.status: creating src/Makefile
config.status: creating src/controller/Makefile
config.status: executing depfiles commands
config.status: executing libtool commands
```

Now you can optionally run the `make clean` command to clean (delete) object files and libraries of an older compilation.

3.2 make command

The make file creates an executable program from the sources in the CYBOP folder. After that CYBOP projects can be interpreted by CIBOY. Simply run the make file by the the command `make`. Be sure the folders "src" and "src/controller" have been created.

Now you can start compiling CYBOP programs.

3.3 run examples

To get a first insight of compiling with CYBOI let's run some files from the example folder. Therefore the previously described steps 3.1 and 3.2 have to be successfully completed.

In CYBOL written files can be compiled via terminal the following way: `[path/]cyboi [path/]filename.cybol`. To run them all correctly you have to switch to the examples directory itself.

The interpreter (cyboi) is situated in `cybop-directory/src/controller/cyboi`. The directory `cybop-directory/examples` accommodates the example folders. Each folder is named by the specific example module.

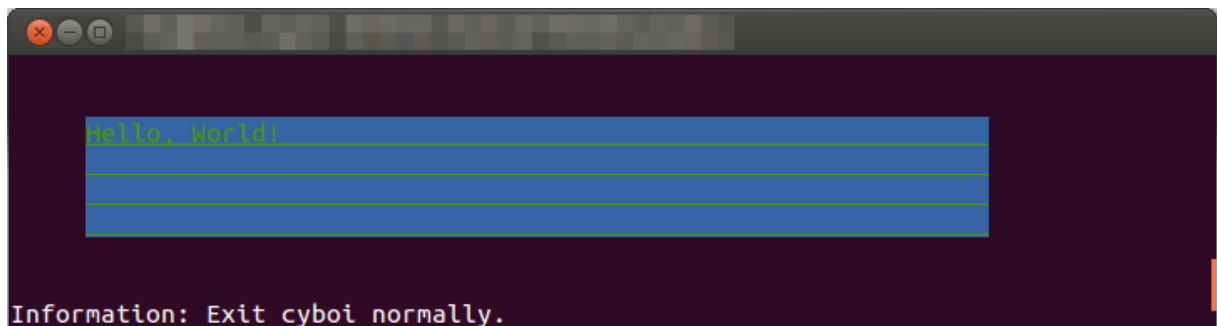
Below you will find a survey of all examples and their output. Feel free to copy the files and edit them to get a better feeling for CYBOL. As it is the most common example, we will start with "Hello, World!".

3.3.1 HelloWorld

This is the command to run the "Hello, World!" example from the main CYBOP directory:

```
src/controller/cyboi examples/helloworld/run.cybol.
```

The picture shows the output:



3.3.2 addition

This example adds the two vectors and prints the result to the screen:

```
2,3,4
```


3.3.3 addition_dynamic_model

In this example three different integer variables are dynamically created, added and the sum is being printed to the console output:

4

3.3.4 addition_dynamic_model_with_root

“With root” describes the root node that is created and after all values are added to this node, called “addition_application”. All other functionalities are equal to the former example, except the integer values:

7

3.3.5 addition_static_model

For this example an external model file (“domain.cybol”) is called on execution. It contains several integer values that are added and finally printed:

4

3.3.6 addition_using_indices

This example is very comparable to “addition_dynamic_model”. The difference is the way of accessing the knowledge tree nodes. Here they are not only accessed by dot-separated names (.sum) but also by indices (. [0]). The output looks following:

5

3.3.7 addition_using_serialisation

In this specific example some integer values are dynamically created and initialised. Additionally this time another text node is created and initialised. The text from the node is being deserialised into an integer value which is being added to the sum of the other integer values. Then the sum us being printed on screen:

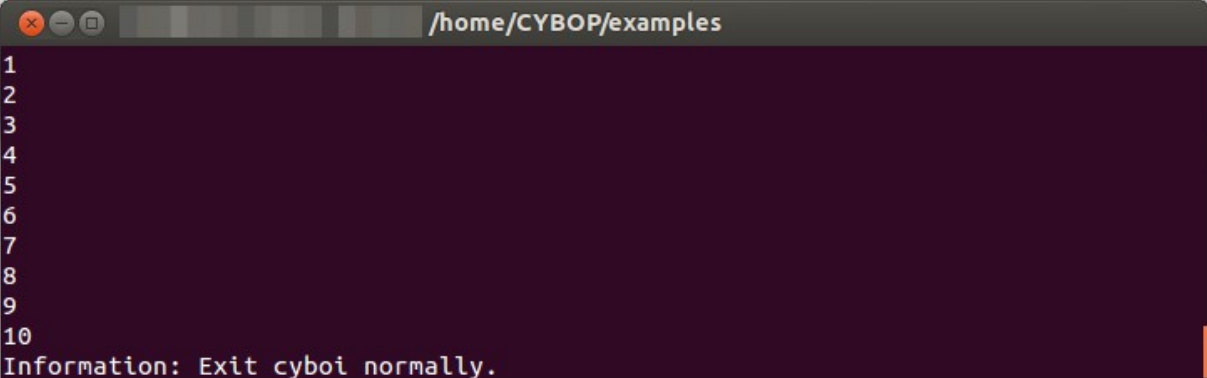
7

3.3.8 character_encoding

As there are characters that cannot be displayed correctly on console they are written into a file in the local directory so they can be checked. The file will be utf-8-encoded and named "iso-8859-15.txt".

3.3.9 counter

The example "counter" counts from 1 to 10 and prints each number on the screen:



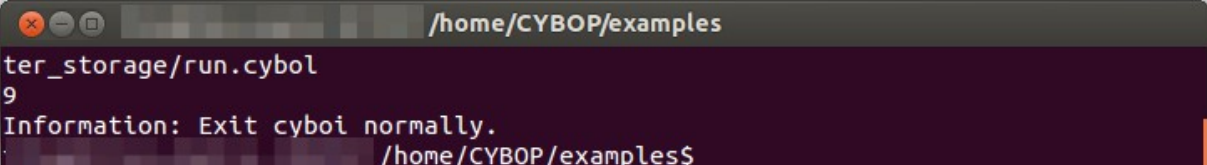
```
1
2
3
4
5
6
7
8
9
10
Information: Exit cyboi normally.
```

3.3.10 counter_static_model

On the console you will have the same output as in the previous example. But here the program is run in a loop.

3.3.11 counter_storage

This problem gets the number from the external file count.txt. This number is printed on the screen, counted up and rewritten to the file. Here is the output when you started with 9 (in the file):

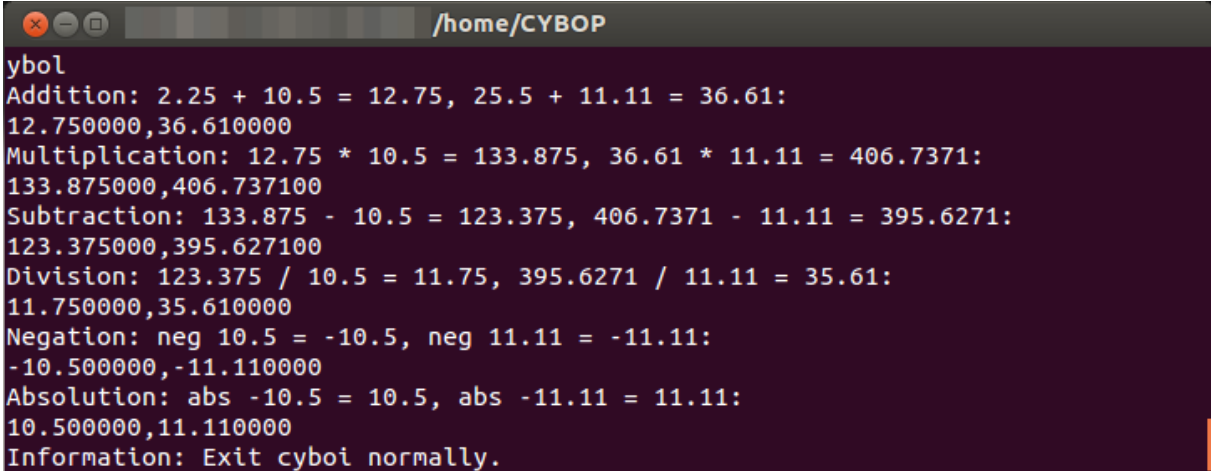


```
ter_storage/run.cybo1
9
Information: Exit cyboi normally.
/home/CYBOP/examples$
```

The file itself now will be containing the integer number 10.

3.3.12double

In the output from this example you can see a few calculation examples for double values:



```

/home/CYBOP
ybol
Addition: 2.25 + 10.5 = 12.75, 25.5 + 11.11 = 36.61:
12.750000,36.610000
Multiplication: 12.75 * 10.5 = 133.875, 36.61 * 11.11 = 406.7371:
133.875000,406.737100
Subtraction: 133.875 - 10.5 = 123.375, 406.7371 - 11.11 = 395.6271:
123.375000,395.627100
Division: 123.375 / 10.5 = 11.75, 395.6271 / 11.11 = 35.61:
11.750000,35.610000
Negation: neg 10.5 = -10.5, neg 11.11 = -11.11:
-10.500000,-11.110000
Absolution: abs -10.5 = 10.5, abs -11.11 = 11.11:
10.500000,11.110000
Information: Exit cyboi normally.
```

3.3.13exit

The exit example simply starts the exit operation. That means the program is shutting down right after it started.

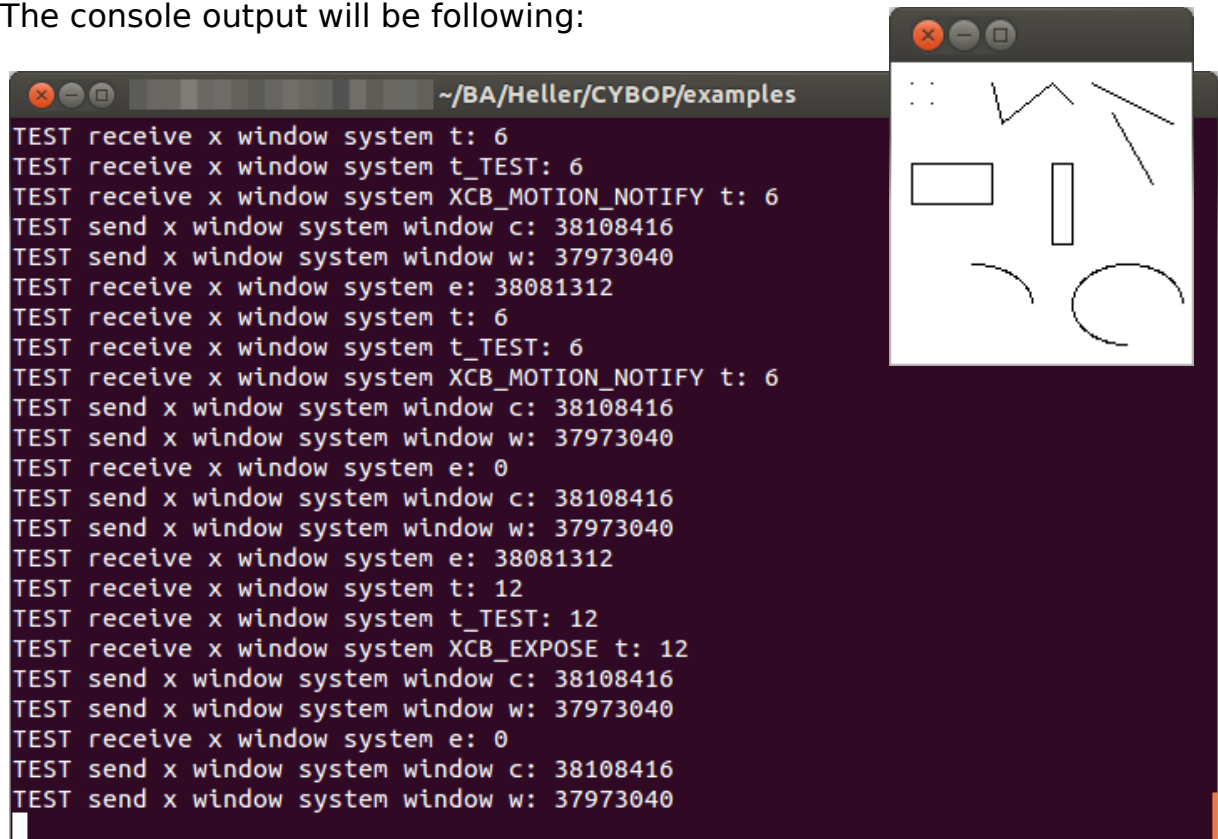
3.3.14exit_cybol_file

This example works equal to the previous one. It simply differs in the external run of the exit operation from an external file "exit.cybol".

3.3.15gui

As you can see below the gui example will draw a window filled with geometric functions.

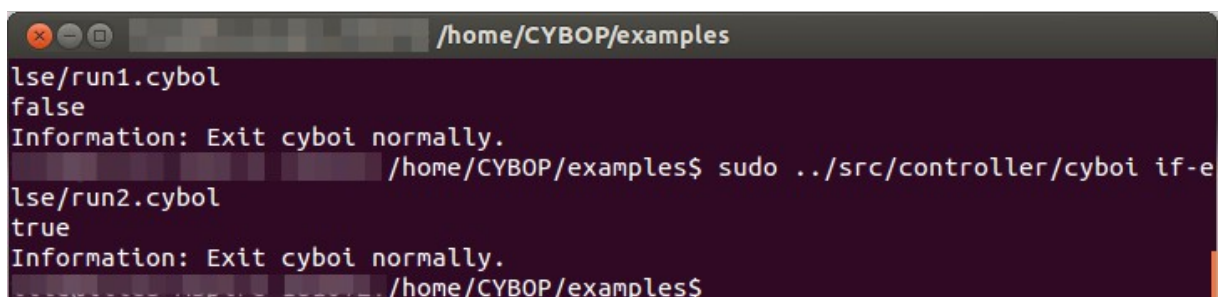
The console output will be following:



```
~/BA/Heller/CYBOP/examples
TEST receive x window system t: 6
TEST receive x window system t_TEST: 6
TEST receive x window system XCB_MOTION_NOTIFY t: 6
TEST send x window system window c: 38108416
TEST send x window system window w: 37973040
TEST receive x window system e: 38081312
TEST receive x window system t: 6
TEST receive x window system t_TEST: 6
TEST receive x window system XCB_MOTION_NOTIFY t: 6
TEST send x window system window c: 38108416
TEST send x window system window w: 37973040
TEST receive x window system e: 0
TEST send x window system window c: 38108416
TEST send x window system window w: 37973040
TEST receive x window system e: 38081312
TEST receive x window system t: 12
TEST receive x window system t_TEST: 12
TEST receive x window system XCB_EXPOSE t: 12
TEST send x window system window c: 38108416
TEST send x window system window w: 37973040
TEST receive x window system e: 0
TEST send x window system window c: 38108416
TEST send x window system window w: 37973040
```

3.3.16if-else

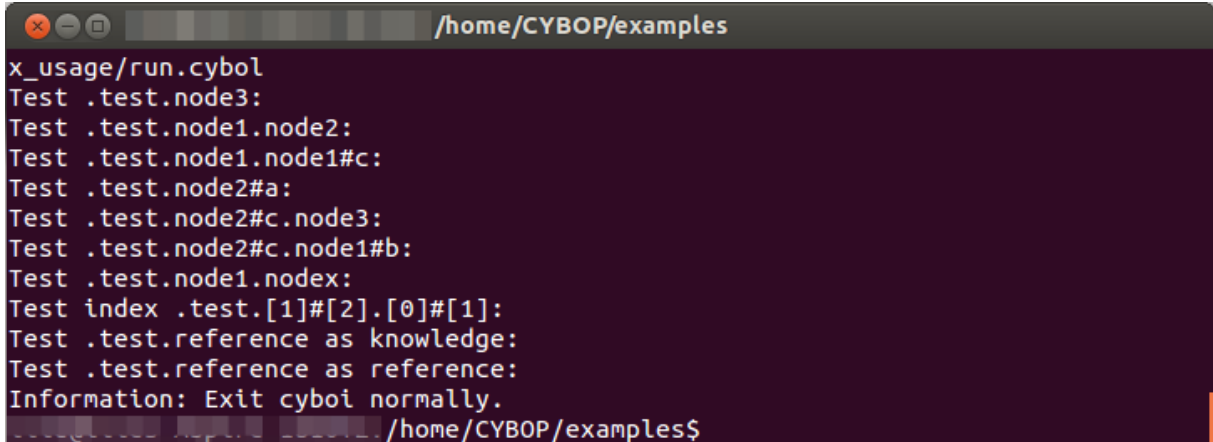
This examples reads from model files the true.cybol and false.cybol. It demonstrates the usage of the if-else-statement. Therefore you got two run-files with different output:



```
/home/CYBOP/examples
lse/run1.cybol
false
Information: Exit cyboi normally.
/home/CYBOP/examples$ sudo ../src/controller/cyboi if-e
lse/run2.cybol
true
Information: Exit cyboi normally.
/home/CYBOP/examples$
```

3.3.17index_usage

“Index usage“ tests hierarchical knowledge path names and indices by building them up and accessing them. Check the “run.cybol“ to view the graphical tree it follows, the output shows the nodes it passes:



```
/home/CYBOP/examples
x_usage/run.cybol
Test .test.node3:
Test .test.node1.node2:
Test .test.node1.node1#c:
Test .test.node2#a:
Test .test.node2#c.node3:
Test .test.node2#c.node1#b:
Test .test.node1.nodex:
Test index .test.[1]#[2].[0]#[1]:
Test .test.reference as knowledge:
Test .test.reference as reference:
Information: Exit cyboi normally.
/home/CYBOP/examples$
```

3.3.18programme_execution

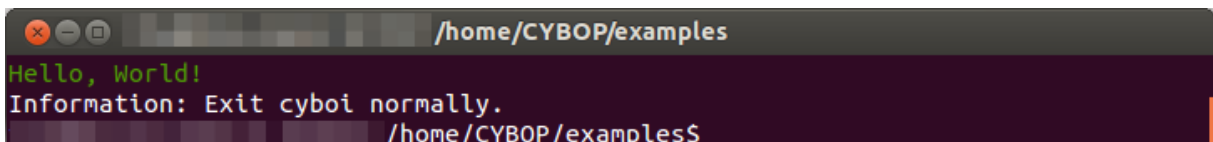
The programme_execution example launches a former installed program by the bash command. In case of the pre-settings the Midnight Commander will be started. Therefore of course it has to be installed and the environment variables have to be set correctly.

3.3.19shell_command_execution

A quite simple tool to run a unix shell (bash) command.

3.3.20shell_output

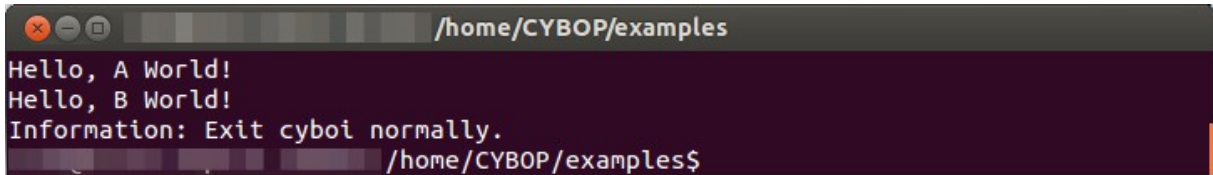
This program will simply print a string onto the screen. In the pre-edited example you will get the following output:



```
/home/CYBOP/examples
Hello, World!
Information: Exit cyboi normally.
/home/CYBOP/examples$
```

3.3.21 shell_output_branch

Like the previous example you get a bash console output with this program. In opposite it calls two external models and finally shuts down by calling an exit operation.

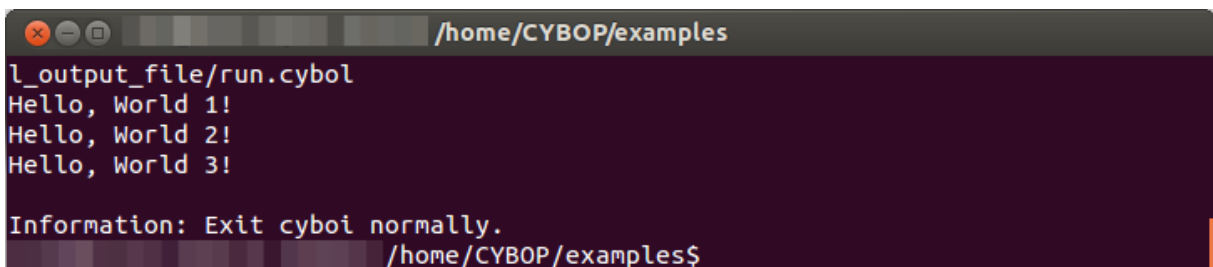


```

/home/CYBOP/examples
Hello, A World!
Hello, B World!
Information: Exit cyboi normally.
/home/CYBOP/examples$
```

3.3.22 shell_output_file

The shell output for this example is read from a local file "text.txt".



```

/home/CYBOP/examples
l_output_file/run.cybol
Hello, World 1!
Hello, World 2!
Hello, World 3!
Information: Exit cyboi normally.
/home/CYBOP/examples$
```

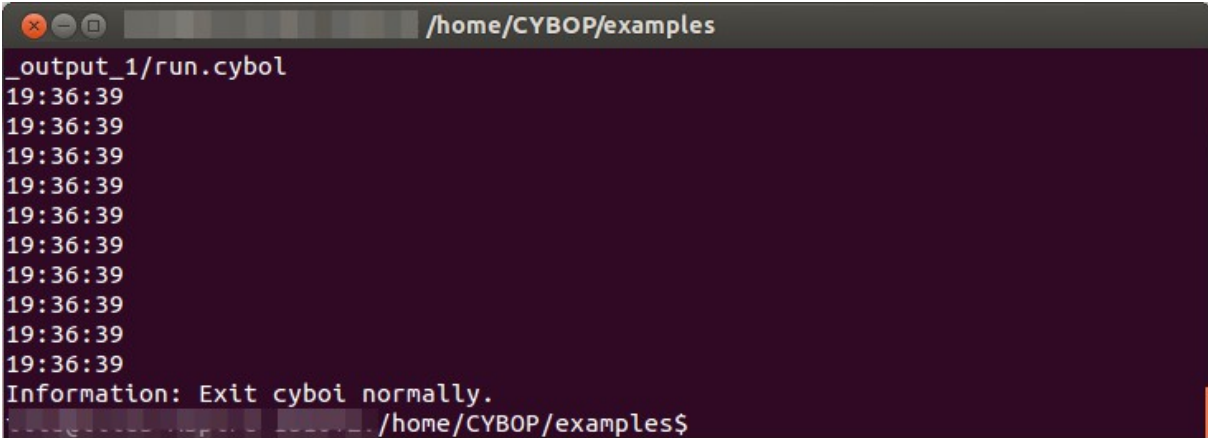
3.3.23 shell_output_sequence

Once more a shell output, generated by a called sequence of commands addressed an external model. Again we have the output:

Hello World!

3.3.24time_output_1

For a repeatedly print of the current timestamp this example calls a bash command in a loop.



```
output_1/run.cybo1
19:36:39
19:36:39
19:36:39
19:36:39
19:36:39
19:36:39
19:36:39
19:36:39
19:36:39
19:36:39
19:36:39
Information: Exit cybo1 normally.
/home/CYBOP/examples$
```

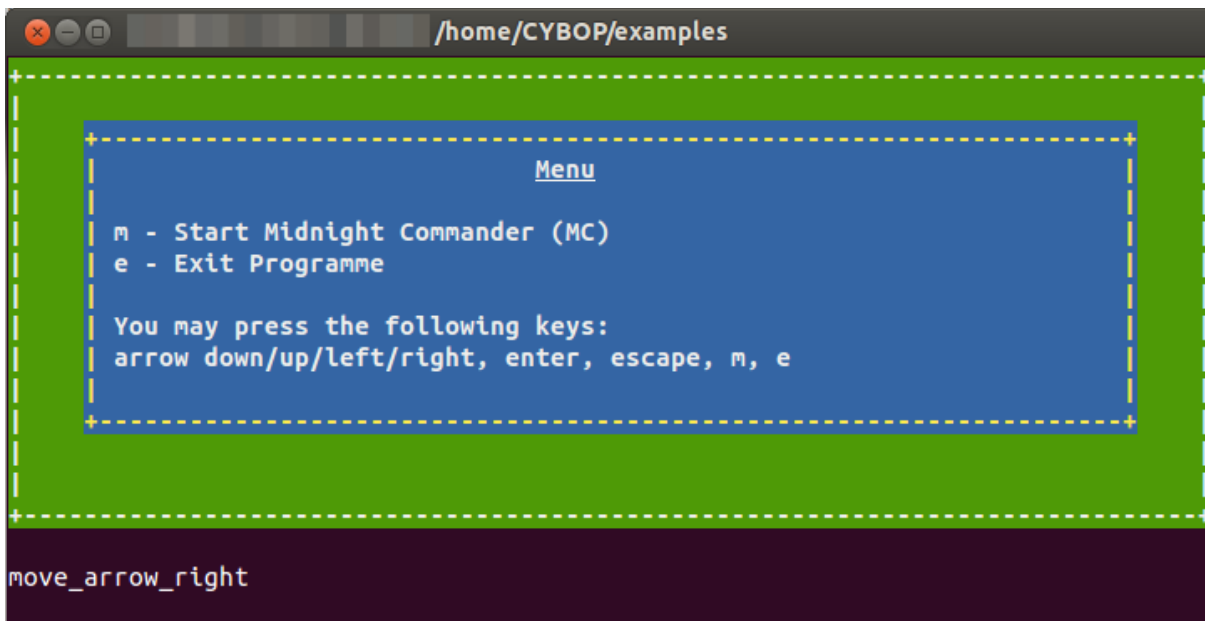
3.3.25time_output_2

This one does principally the same as the previous example. But the time stamps are first printed to a file and read from there afterwards to be put out on the console.

3.3.26time_output_3

The last time example output looks again close to the first. A loop prints the current time with a delay of one second which is created by another bash command.

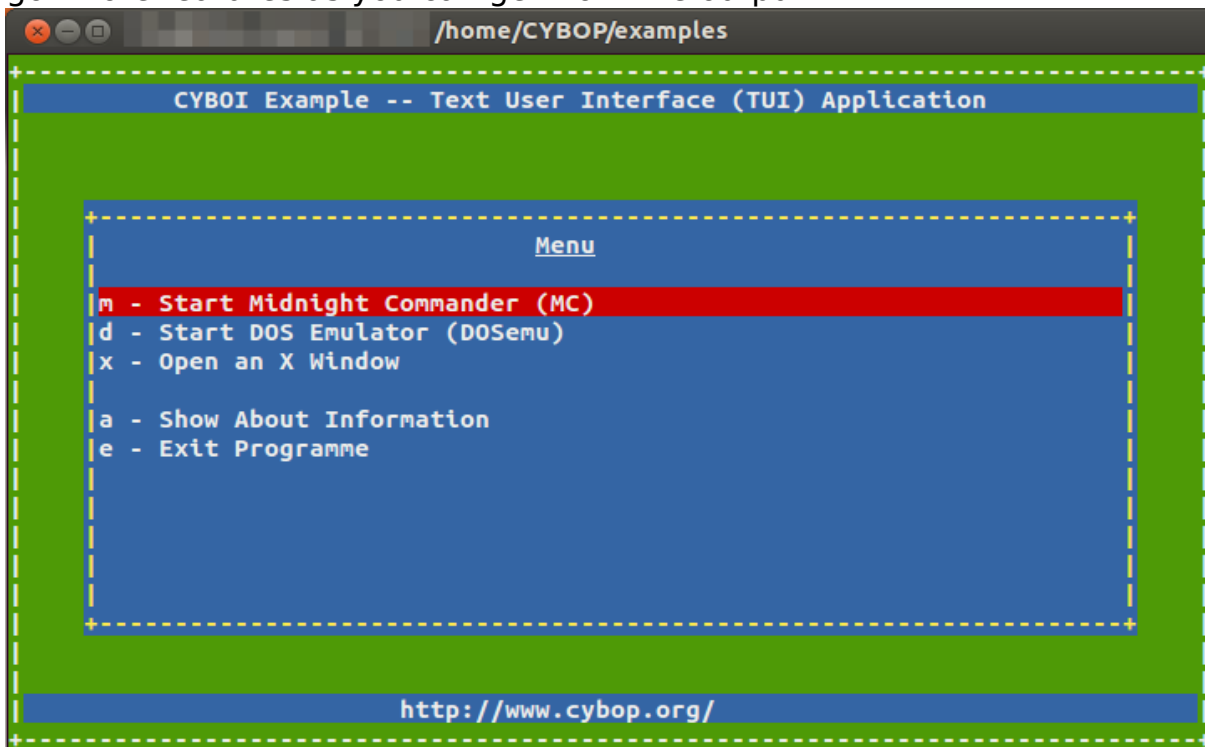
3.3.27tui



The tui (text user interface) example is a small compilation of some above explained programs. It receives command from the user input and can even star another program (Midnight Commander) from the bash.

3.3.28ui_control

This is one of the most extensive examples. It is comparable to the tui but got more features as you can get from the output:



4 upshot

Hopefully this manual introduced you well to the world of CYBOP. As it uses a logical structure by the XML-based language, it's capable to find solutions for a huge variety of problems. The examples give an insight for the possibilities CYBOP and its components provide. You may even get interested in starting development in CYBOL yourself.