|  |
| --- |
|  |
| **Einsteiger Tutorial für Cybernetics Oriented Programming (CYBOP)** |
|  |
|  |
|  |
| **20.04.2011** |

|  |
| --- |
|  |

**Inhaltsverzeichnis**

[1 Einleitung 3](#_Toc291676711)

[2 Installation CYBOP 3](#_Toc291676712)

[2.1 HTTP: Download der aktuellen Version von http://cybop.berlios.de/ 3](#_Toc291676713)

[2.2 SVN: Download der aktuellen Version via SVN von BerliOS Developer 4](#_Toc291676714)

[3 Kompilieren von CYBOI 4](#_Toc291676715)

[3.1 Scriptbefehl autogen.sh: 4](#_Toc291676716)

[3.2 Scriptbefehl configure: 5](#_Toc291676717)

[3.3 Kommando make clean: 5](#_Toc291676718)

[3.4 Kommando make: 6](#_Toc291676719)

[4 Beispiele 6](#_Toc291676720)

[4.1 „HelloWorld“ Ausgabe aus einer CYBOL-Datei 7](#_Toc291676721)

[4.2 Exit-Befehl aus zusätzlicher Textdatei (CYBOL- und Textdatei) 8](#_Toc291676722)

[4.3 Exit-Befehl aus zusätzlicher CYBOL-Datei 9](#_Toc291676723)

[4.4 Programmflusssteuerung mittels Verzweigung (if… else…) 9](#_Toc291676724)

[4.5 Programmflusssteuerung mittels Zählschleife (loop until) 10](#_Toc291676725)

[5 Fazit 12](#_Toc291676726)

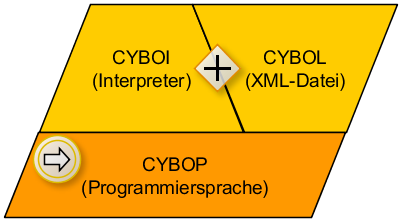
# Einleitung

CYBOP steht für Kybernetik Oriented Programming und ist eine neue Theorie für die Software-Entwicklung, aufbauend auf Konzepten, die aus der Natur entnommen sind. Es besteht aus den zwei Kernelementen:

* CYBOL
* CYBOI

CYBOL ist ein XML-basiertes Application Programming Spezifikationssprache und somit völlig plattformunabhängig.

CYBOI ist der entsprechende Interpreter, der benötigt wird, um Systeme in CYBOL definiert auszuführen.



# Installation CYBOP

Der Quellcode von CYBOP ist auf zwei verschieden Wegen erhältlich:

## HTTP: Download der aktuellen Version von <http://cybop.berlios.de/>

Beispiel: [cybop-0.10.0.tar.bz2](http://prdownload.berlios.de/cybop/cybop-0.10.0.tar.bz2)

* Ordner für CYBOP Projekt anlegen:

Beispiel: **/home/cybop**

* In den Ordner wechseln und Dateien Entpacken:

**tar -xvjf /[home]/[user]/[download\_ordner]/cybop-0.10.0.tar.bz2**

## SVN: Download der aktuellen Version via SVN von BerliOS Developer

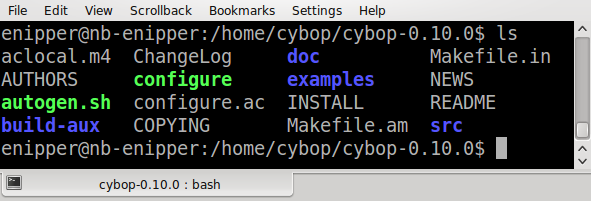
* Ordner für CYBOP Projekt anlegen:

Beispiel: /home/cybop

* In den Ordner wechseln und Dateien auschecken:

svn checkout svn://svn.berlios.de/cybop/trunk

Nach dem Download sollten Sie folgende Dateien in Ihrem Projekt-Ordner vorfinden.



# Kompilieren von CYBOI

Das Kompilieren ist unabhängig von der Art des Quellcode-Zugangs.

**HTTP/SVN:** In den Ordner **/home/cybop/cybop-0.10.0** wechseln

## Scriptbefehl autogen.sh:

Nach Ausführen der Datei **./autogen.sh** sollten folgende Statusmeldungen auf der Konsole erscheinen.

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/sh /bin/sh ./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... gawk

checking whether make sets $(MAKE)... yes

checking how to create a ustar tar archive... gnutar

checking build system type... x86\_64-suse-linux-gnu

checking host system type... x86\_64-suse-linux-gnu

checking Host cpu... x86\_64

checking For operating system... Linux

checking for gcc... gcc

checking whether the C compiler works... yes  
 …

config.status: executing libtool commands

Making all in src

make[1]: Entering directory `/home/cybop/src'

Making all in controller

make[2]: Entering directory `/home/cybop/src/controller'

make[2]: Leaving directory `/home/cybop/src/controller'

make[2]: Entering directory `/home/cybop/src'

make[2]: Leaving directory `/home/cybop/src'

make[1]: Leaving directory `/home/cybop/src'

make[1]: Entering directory `/home/cybop'

make[1]: Leaving directory `/home/cybop'

## Scriptbefehl configure:

Mit **./configure** wird Kompilierung fortgesetzt und folgende Meldungen auf der Konsole ausgegeben.

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... gawk

…

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

## Kommando make clean:

Mit dem Kommando **make clean** wird das Clean-Target des Source-Makefile aufgerufen und alle generierten Dateien (Bibliotheken, Objektdateien) werden gelöscht.

Making clean in src

Making clean in controller

rm -f cyboi

rm -rf .libs \_libs

rm -f \*.o

rm -f \*.lo

Making clean in .

rm -rf .libs \_libs

rm -f \*.lo

Making clean in .

rm -rf .libs \_libs

rm -f \*.lo

## Kommando make:

Mit dem Befehl **make** wird das CYBOP Projekt automatisiert angepasst, so dass aus vielen verschiedenen Dateien mit Quellcode das fertige, ausführbare Programm entsteht.

Making all in src

Making all in controller

gcc -DPACKAGE\_NAME=\"cybop\" -DPACKAGE\_TARNAME=\"cybop\"

-DPACKAGE\_VERSION=\"0.10.0\" -DPACKAGE\_STRING=\"cybop\ 0.10.0\"

-DPACKAGE\_BUGREPORT=\"[christian.heller@tuxtax.de](https://mail.ba-leipzig.de/src/compose.php?send_to=christian.heller%40tuxtax.de)\" -DPACKAGE\_URL=\"\"

-DPACKAGE=\"cybop\" -DVERSION=\"0.10.0\" -DSTDC\_HEADERS=1

…

libtool: link: gcc -I/usr/include -DGNU\_LINUX\_OPERATING\_SYSTEM -g -O2 -o

cyboi cyboi.o -lpthread -lglut -lX11 -lGLU –lGL

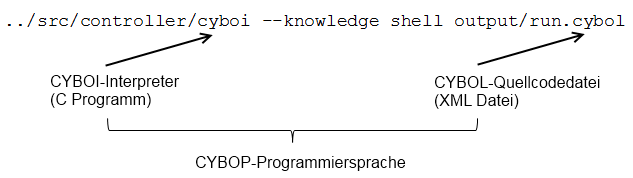
Nach der Kompilierung von CYBOI wurden die Ordner “src” und “src/controller” angelegt. Damit ist die Installation von CYBOI abgeschlossen. Im weiteren können erste CYBOP-Programme ausgeführt werden.

# Beispiele

Um einen leichten Einstieg in CYBOL zu ermöglichen, werden im Weiteren 3 Varianten von Beispielen demonstriert:

* „HelloWorld“ Ausgabe aus einer CYBOL-Datei
* Exit-Befehl aus zusätzlicher Textdatei (CYBOL- und Textdatei)
* Exit-Befehl aus zusätzlicher CYBOL-Datei

Bei der runtergeladenen CYBOP-Software finden sich im Ordner **/cybop/examples** erste Beispiele zum Ausprobieren. Das Kommando zum Starten der Beispiele setzt sich aus dem CYBOI-Interpreter, der Option [--knowledge] für „Wissensbaum“ und der auszuführende CYBOL-Datei zusammen.



## „HelloWorld“ Ausgabe aus einer CYBOL-Datei

Um eine Ausgabe von „HelloWorld“ auf der Konsole zu erhalten, muss zunächst in das Verzeichnis **/cybop/examples** gewechselt werden. In diesem Verzeichnis befindet sich das Einstiegsbeispiel **/shell\_output** mit der Startdatei **run.cybol**, welche den XML-Quelltext enthält. Durch den Befehl

../src/controller/cyboi --knowledge shell\_output/run.cybol

wird der Interpreter (CYBOI) mit der XML-Datei (run.cybol) gestartet. Dies führt zu folgendem Ergebnis auf der Konsole.

Hello, World!

Information: Exit cyboi normally.

Die XML-Datei run.cybol enthält die Zeichenkette „HelloWorld“ und den Exit-Befehl, um den Interpreter nach Ausführen der Datei zu beenden. Dies kann man ebenfalls am Inhalt der run.cybol Datei erkennen.

<model>

<part name="send\_model\_to\_output" channel="inline"

abstraction="operation/plain" model="send">

<property name="channel" channel="inline" abstraction="text/plain"

model="shell"/>

<property name="language" channel="inline"

abstraction="text/plain" model="text/plain"/>

<property name="message" channel="inline" abstraction="text/plain"

model="**Hello, World!**"/>

<property name="new\_line" channel="inline"

abstraction="logicvalue/boolean" model="true"/>

</part>

<part name="exit\_application" channel="inline"

abstraction="operation/plain" model="**exit**"/>

</model>

## Exit-Befehl aus zusätzlicher Textdatei (CYBOL- und Textdatei)

Das Beispiel **/exit\_text\_file** soll demonstrieren, wie der EXIT-Befehl aus einer separaten Textdatei geladen und ausgeführt wird, so dass auf der Konsole folgende Ausgabe erscheint:

Information: Exit cyboi normally.

Das Programm wird mit dem Kommando gestartet:

../src/controller/cyboi --knowledge exit\_text\_file/run.cybol

Das Besondere an diesem Beispiel ist das Einlesen einer Textdatei mit einem EXIT-Befehl über den File‑Channel. Dieser Befehl wird als Signal an den CYBOI Interpreter gesendet und beendet diesen.

Die zu startende XML Datei „run.cybol“ beinhaltet folgenden Quellcode:

<model>

<part name="startup" channel="inline" abstraction="operation/plain" model="send">

<property name="channel" channel="inline" abstraction="text/plain"

model="signal"/>

<property name="message" channel="file" abstraction="operation/plain"

model="exit\_text\_file/exit.txt"/>

</part>

</model>

Durch die Angabe der Abstraktion (Typ) „operation/plain“ wird der Inhalt der exit.txt als Kommando eingesetzt und ausgeführt. In der exit.txt steht nur die Zeichenkette "exit" ohne weitere Informationen.

## Exit-Befehl aus zusätzlicher CYBOL-Datei

Eine weitere Möglichkeit einen modularen Aufbau zu erhalten, besteht durch die Verwendung mehrerer CYBOL-Dateien. Das Beispiel **/exit\_cybol\_file** verdeutlicht diese Variante des Aufbaus durch mehrere XML-Dateien.

In der exit.cybol Datei wird mittels eines Inline-Channel die EXIT-Operation zur Verfügung gestellt.

<model>

<part name="exit" channel="inline" abstraction="operation/plain" model="exit"/>

</model>

In der run.cybol Datei wird die exit.cybol Datei über den File-Channel und der Abstraktion (Typ) „text/cybol“ folgendermaßen eingebunden.

<model>

<part name="startup" channel="inline" abstraction="operation/plain"

model="send">

<property name="channel" channel="inline" abstraction="text/plain"

model="signal"/>

<property name="message" channel="file" abstraction="text/cybol"

model="exit\_cybol\_file/exit.cybol"/>

</part>

</model>

Nach der Ausführung des Beispiels mit:

../src/controller/cyboi --knowledge exit\_cybol\_file/run.cybol

wird die gleiche Ausgabe wie im vorigen Beispiel auf der Konsole ausgegeben.

Information: Exit cyboi normally.

Ähnliches Beispiel siehe examples/shell\_output\_sequence!

## Programmflusssteuerung mittels Verzweigung (if… else…)

Um eine Wenn-Dann-Abfrage zu realisieren, steht das Beispiel /shell\_output\_branch zur Verfügung. In der run.cybol Datei wird entschieden, ob die Abfrage wahr oder falsch ist. Bei Wahrheitswert „true“ wird der Code der Datei model\_**a**.cybol ausgeführt und bei Wahrheitswert „false“ wird die Datei model\_**b**.cybol ausgeführt.

<model>

<part name="simulate\_first\_case" channel="inline" abstraction="operation/plain" model="branch">

<property name="criterion" channel="inline" abstraction="logicvalue/boolean" model="**true**"/>

<property name="**true**" channel="file" abstraction="text/cybol" model="shell\_output\_branch/model\_a.cybol"/>

<property name="false" channel="file" abstraction="text/cybol" model="shell\_output\_branch/model\_b.cybol"/>

</part>

<part name="simulate\_second\_case" channel="inline" abstraction="operation/plain" model="branch">

<property name="criterion" channel="inline" abstraction="logicvalue/boolean" model="**false**"/>

<property name="true" channel="file" abstraction="text/cybol" model="shell\_output\_branch/model\_a.cybol"/>

<property name="**false**" channel="file" abstraction="text/cybol" model="shell\_output\_branch/model\_b.cybol"/>

</part>

<part name="shutdown" channel="inline" abstraction="operation/plain" model="exit"/>

</model>

Nach erfolgreichem Ausführen des Quellcodes erhält man folgende Ausgabe auf der Konsole:

/examples> ../src/controller/cyboi --knowledge shell\_output\_branch/run.cybol

Hello, A World!

Hello, B World!

Information: Exit cyboi normally.

## Programmflusssteuerung mittels Zählschleife (loop until)

Ein weiteres Beispiel /shell\_output\_loop zeigt, wie mittels einer Zählvariable (counter) eine Schleife nach einer definierten Anzahl von Durchläufen abgebrochen wird. In der run.cybol Datei werden die Zählvariable und Abbruchbedingung erzeugt und initialisiert:

<model>

<part name="create\_application" channel="inline" abstraction="operation/plain" model="create">

<property name="name" channel="inline" abstraction="text/plain" model="shell\_output\_loop"/>

<property name="abstraction" channel="inline" abstraction="text/plain" model="compound"/>

<property name="element" channel="inline" abstraction="text/plain" model="part"/>

</part>

<part name="create\_break" channel="inline" abstraction="operation/plain" model="create">

<property name="name" channel="inline" abstraction="text/plain" model="**break**"/>

<property name="abstraction" channel="inline" abstraction="text/plain" model="integer"/>

<property name="element" channel="inline" abstraction="text/plain" model="part"/>

<property name="whole" channel="inline" abstraction="path/knowledge" model=".shell\_output\_loop"/>

</part>

<part name="create\_count" channel="inline" abstraction="operation/plain" model="create">

<property name="name" channel="inline" abstraction="text/plain" model="**count**"/>

<property name="abstraction" channel="inline" abstraction="text/plain" model="integer"/>

<property name="element" channel="inline" abstraction="text/plain" model="part"/>

<property name="whole" channel="inline" abstraction="path/knowledge" model=".shell\_output\_loop"/>

</part>

<part name="initialise\_break" channel="inline" abstraction="operation/plain" model="copy">

<property name="abstraction" channel="inline" abstraction="text/plain" model="integer"/>

<property name="source" channel="inline" abstraction="number/integer" model="0"/>

<property name="destination" channel="inline" abstraction="path/knowledge"

model=".shell\_output\_loop.break"/>

</part>

<part name="initialise\_count" channel="inline" abstraction="operation/plain" model="copy">

<property name="abstraction" channel="inline" abstraction="text/plain" model="integer"/>

<property name="source" channel="inline" abstraction="number/integer" model="0"/>

<property name="destination" channel="inline" abstraction="path/knowledge"

model=".shell\_output\_loop.count"/>

</part>

<part name="startup" channel="inline" abstraction="operation/plain" model="loop">

<property name="break" channel="inline" abstraction="path/knowledge" model=".shell\_output\_loop.break"/>

<property name="model" channel="file" abstraction="text/cybol" model="shell\_output\_loop/model.cybol"/>

</part>

<part name="shutdown" channel="inline" abstraction="operation/plain" model="exit"/>

</model>

Anschließend wird die model.cybol Datei aufgerufen, welche den eigentlichen Schleifenkörper darstellt. Durch Verwendung des Operators „greater\_or\_equal“ werden die break und count Variable bei jedem Schleifendurchlauf verglichen und die count Variable um den Wert 1 erhöht. Sobald count größer oder gleich 4 ist, wird die Schleife abgebrochen.

<model>

<part name="compare\_count" channel="inline" abstraction="operation/plain" model="**greater\_or\_equal**">

<property name="left" channel="inline" abstraction="path/knowledge"

model="**.shell\_output\_loop.count**"/>

<property name="right" channel="inline" abstraction="number/integer" model="**4**"/>

<property name="result" channel="inline" abstraction="path/knowledge"

model="**.shell\_output\_loop.break**"/>

</part>

<part name="output" channel="inline" abstraction="operation/plain" model="send">

<property name="channel" channel="inline" abstraction="text/plain" model="shell"/>

<property name="language" channel="inline" abstraction="text/plain" model="text/plain"/>

<property name="message" channel="inline" abstraction="text/plain" model="Hello, World!"/>

<property name="new\_line" channel="inline" abstraction="logicvalue/boolean" model="true"/>

</part>

<part name="increment\_loop\_count" channel="inline" abstraction="operation/plain" model="add">

<property name="abstraction" channel="inline" abstraction="text/plain" model="integer"/>

<property name="summand\_1" channel="inline" abstraction="path/knowledge"

model=".shell\_output\_loop.count"/>

<property name="summand\_2" channel="inline" abstraction="number/integer" model="1"/>

<property name="sum" channel="inline" abstraction="path/knowledge" model=".shell\_output\_loop.count"/>

</part>

</model>

Die Ausgabe auf der Konsole besteht aus 5 Schleifendurchläufen (0…4):

/examples> ../src/controller/cyboi --knowledge shell\_output\_loop/run.cybol

Hello, World!

Hello, World!

Hello, World!

Hello, World!

Hello, World!

Information: Exit cyboi normally.

# Fazit

Dieses Tutorial verdeutlicht die vielfältigen Möglichkeiten zur Erstellung von CYBOP‑Programmen. Die CYBOL -Dateien sind in einer durchgehend einheitlichen XML-Struktur und bilden dennoch Lösungen für verschiedenste Problemstellungen. Die mitgelieferten Beispiele wie z.B. ui\_control zur Erstellung von textuelle Oberflächen oder http\_communication für den Austausch von Daten zwischen Systemen geben weitere Einblicke für den Umgang mit CYBOP.