Project AN.ON

Development Environment

Kuno G. Grün
Rolf Wendolsky
Derek Daniel
Elmar Schraml

Version: 3.10

from: 6. November 2007

Projekt AN.ON
Hannes Federrath, Universität Regensburg
Stefan Köpsell, TU Dresden

http://www.anon-online.de

Summary

This document describes how to set up and use the recommended development environment for the AN.ON project.

Document history

version	status	date	authors	description
0.01	in progress	21.03.04	Rolf Wendolsky	First version
0.02	in progress	23.03.04	Rolf Wendolsky	Inserted chapter „IDEs“
0.03	in progress	24.03.04	Rolf Wendolsky	Finished „IDEs“
0.04	in progress	25.03.04	Rolf Wendolsky	Added 'Compilation and installation of external libraries' Index for 'Test environment'
0.05	in progress	30.03.04	Rolf Wendolsky	Added 'CppUnit' configuration chapter
0.06	in progress	31.03.04	Rolf Wendolsky	Corrections in 'CppUnit'
0.07	in progress	08.04.04	Rolf Wendolsky	Added chapter about CVS
0.08	in progress	08.04.04	Rolf Wendolsky	Added 'Testconfiguration mixproxy'
0.09	in progress	15.04.04	Rolf Wendolsky	Different corrections in 'Test environment' and 'Testconfiguration'
0.10	in progress	09.09.04	Rolf Wendolsky	Different corrections
1.00	released	25.11.04	Rolf Wendolsky	New libraries added
1.01	released	01.08.05	Rolf Wendolsky	JDKs updated
2.0alpha	in progress	04.08.05	Kuno G. Gruen	First translation into English
2.0beta	in progress	15.09.05	Kuno G. Gruen	Added configurations for Visual Studio .NET 2003 Eclipse 3.1; different updates and additions
2.01	in progress	18.09.05	Rolf Wendolsky	Fixed some errors
2.02	in progress	18.09.05	Kuno G. Gruen	Fixes some errors
2.03	in progress	17.10.05	Derek Daniel	Corrected english version
3.00	released	17.10.05	Rolf Wendolsky	Final review
3.01	released	22.11.05	Rolf Wendolsky	Version corrected
3.02	released	22.12.05	Kuno G. Gruen	Version corrected / updated
3.03	released	16.01.06	Rolf Wendolsky	Corrected MixConfig infos
3.04	released	26.01.06	Jonas Schießl	Added solution for compiling proxytest under Windows using Eclipse and Cygwin
3.05	in progress	09.02.06	Jonas Schießl	Description of FatJar plugin.
3.06	released	06.04.06	Rolf Wendolsky	Major corrections and reintegration of old german JBuilder instructions
3.07	released	09.05.06	Jonas Schießl	How to convert linebreaks
3.08	released	28.02.07	Elmar Schraml	translated Jbuilder instructions, added payment instande

				installation and configuration
3.09	released	14.03.07	Elmar Schraml	updated database installation; added AI configuration for prepaid system
3.10		06.11.07	Elmar Schraml	merged previous additions and fixes to AI/PIG installation

Table of Contents

1 Introduction

This document is an introduction to the development tools and code base of the AN.ON project. We recommend some IDEs and give some help on their installation and usage. Configuration files for the IDEs are kept up to date by our project maintainer. The proposed IDEs are platform neutral and can be used with Windows, Linux and MacOS. AN.ON itself is split into Java and C++ components that can be compiled and run on any of the mentioned systems.

Using Linux, JBuilder X/2005 is recommended for Java, but both programming languages (Java and C++) can be covered with Eclipse 3.1. The project settings for C++ are based on the Unix-native GNU make in combination with CDT 3.0.0.

Using Windows, you will need either JBuilder X/2005 (recommended) or Eclipse 3.1 for Java development. C++ development may be done with Microsoft Visual Studio .NET 2003. Old project settings for C++ Builder X are provided, too, but will be replaced by an Eclipse 3.1 configuration in the future.

Using MacOS X, JBuilder is being used for Java development. For C++, Eclipse 3.1 should work, although it has not yet been tested. It may cause some difficulties with the make process.

Once again: you are welcome to develop with any IDE you like. Just keep in mind that you may have to invest some time for generating the project settings.

Windows *(in any version NT / 2000 / XP)*	Java IDE	JBuilderX / Eclipse 3.1
	C++ IDE	Eclipse3.1 + Cygwin Visual Studio 2003 C++ Builder X
SuSE Linux 9.3 *or any other >= kernel 2.4*	Java IDE C++ IDE	JBuilder X / Eclipse 3.1 Eclipse 3.1 & CDT 3.0.0
MacOS X *(not shown in this document)*	Java	Eclipse 3.1 JBuilder X
	C++	Eclipse 3.1 & CDT 3.0.0

The IDEs have been tested with Windows XP Professional and SuSE Linux 9.3.

2 Components

2.1 Mix (CVS-module: proxytest)

Mix is the virtual core of AN.ON. With a Mix, the process of anonymization is done. Although the CVS component's name is *proxytest* (due to historic reasons), the executable program is named *Mix*. Since this is a development documentation, we will normally refer to it as *proxytest*. Due to past performance reasons, *proxytest* is written in C++.

2.2 JAP client (CVS-module: Jap)

The *Jap* client is used as local proxy that manages the communication with *Mix cascades* and *Infoservices*. The client has to run on as many platforms as possible, so it is written as a Java application. It uses an old 1.1.8 Java runtime environment for this reason as well. This runtime version is also available on older operating systems (e.g. MacOS 9 and OS/2).

2.3 InfoService (CVS-module: Jap)

The *Infoservice* is needed to register new Mixes and to connect JAP to existing Mix cascades. It also enables JAP to find the IP addresses of active Mix cascades. *Infoservice* is also written completely in Java.

2.4 Payment Instance (CVS-module: Jap)

The *Payment Instance* administrates user accounts for the AN.ON payment system.

2.5 MixConfig tool (CVS-module: MixConfig)

With this Java application, a configuration file for a Mix can be created and maintained. It is also used to configure the Mix-on-CD live-CD. Again, this tool is written as a Java application.

2.6 Java Mix (CVS-module: JavaMix)

An experimental Mix implementation.

2.7 Anon library

This Java library contains all classes that are needed by JAP, Infoservice and Mixconfig. Changes to this central library potentially affect the other components and have to be tested with each of them.

3 External libraries

Each of the mentioned components needs some external libraries that are not part of the AN.ON project. The library versions given here have been tested with each AN.ON component. Older or newer library versions may work also but might cause problems.

Abbreviations for the libraries are used in the IDE config files. Note that the reference names are case sensitive! Keeping these names is recommended because they are used throughout the project. Getting the sources again from CVS will overwrite a different name resolution.

IDE reference	***library***	***used for***
Java:		
ApacheFTPClient	ApacheFTPClient.jar	FTP-client libraries
ApacheBZIP2	ApacheBzip2.jar	Bzip2 libraries
Apache XML-RPC	apache-xmlrpc-1.1.jar	Apache XML-RPC libraries
BouncyCastleLightForJAP	BouncyCastleLightForJAP.jar	crypto libraries
BouncyCastleLightForMixConfig	BouncyCastleLightForMixConfig.jar	crypto libraries
http	http.jar	HTTP-Client libraries
Jama	Jama.jar	matrices libraries
JavaBeans Activation Framework	activation.jar	needed by JavaMail
JavaMail	mail.jar	blocking resistance
JAI 1.1.2	jai_core.jar	blocking resistance
junitx	junitx-5.1.jar	Unittests for protected and private members
Log4j	log4j.jar	Logging library
mac	MRJClasses.zip	MRJ library for Macintosh
Swing	swingall.jar	exclusively for Java 1.1; installed in Java 1.2
XML-1.0	xml.jarXML related functions	XML related functions
XML-1.1	xml-1.1.jar	XML related functions
XML-2.4	xml-2.4_min.jar	XML related functions
C++:		
cppunit	>= cppunit-1.8.0	Unittests
openssl	openssl-0.9.7e	Data encryption
pthreads	pthreads	Thread programming
xerces	Xerces-c2_5_0	XML-Parser

3.1 Java library installation

You may download the Java libraries from the [ANON] homepage and copy them to any directory you want. We recommend the directory name `ext_lib_jap`.

Since Java is platform independent, you can use the same libraries in both Windows and Linux. You should download the libraries needed by JAP from the [ANON] homepage. If the libraries are not available at the [ANON] homepage or you need newer ones, you must take care of project settings and library names when updating a library. We recommend to put all needed Java libraries together into one directory, for example `libs_java`. You will need to add this directory and the libraries, respective, to the classpath variable of your IDE (described later).

3.2 C++ library installation on Linux

Please use the package manager of your Linux distribution to install the needed C++ libraries. You will need the following C++ libraries in the above mentioned versions. Be sure to get the right ones for your distribution. (The following are the package names from SuSE 9.3. Other distributions may differ in the package names.)

- cppunit
- cppunit-devel
- openssl
- openssl-devel
- xerces-c
- xerces-c-devel

Also be aware that your distributor may deliver some buggy packages. If the package does not work, you will have to get the sources and compile it on your own. To do so, you can normally follow the instructions included with the source code. We mention this because we experienced some problems with OpenSSL and SuSE 9.1.

3.3 C++ library installation on Windows

Using Windows, we recommend Eclipse running on Cygwin as the development platform for C++. Microsoft Visual Studio .NET 2003 will also do. If you decide to use Eclipse for C++ development, the following steps do not apply to you, because you can use the Cygwin package manager to install them.

3.3.1 Xerces-C

Due to some changes with the namespaces in Xerces-C 2.6.0, a new proxytest version needs to be compiled with a Xerces-C source version lower than 2.6.0. A configuration with Xerces-C 2.5.0 has been tested and works properly. To get a running version of the Xerces-C 2.5.0 libraries you have to follow these steps:

- Since Xerces-C 2.5.0 is no longer the current version, you will need to go to:
  http://archive.apache.org/dist/xml/xerces-c/Xerces-C_2_5_0/
  and download
  `xerces-c-src_2_5_0.zip`
- Unpack the downloaded archive to any directory; we recommend for example:
  `d:\coding\libsc\xerces-c-src_2_5_0`
  This chosen directory will be referenced as:
  `[xerces_home]`
- Start Visual Studio 2003 .NET and, by selecting
  File -> Open -> Project, open the file:
  `[xerces_home]\Projects\Win32\VC7\xerces-all\xerces-all.sln`
  If asked to convert the files, you should answer with '**Yes**'.
- Make sure that '**Debug**' is chosen as the '**Solution Configuration**'
- Start building the binaries with:
  Build -> Build Solution (or: **Ctrl+Shift+B**)
  Compilation and building will take a while.
- Now choose a '**Release**' build in the '**Solution Configuration**'
- Once again:
  Build -> Build Solution (or: **Ctrl+Shift+B**)
  Compilation and building will again take a while.

- Depending on the chosen build, you will find the executables in the subdirectories:
  `[xerces_home]\Build\Win32\VC7\Debug`
  `[xerces_home]\Build\Win32\VC7\Release`

- Add the following environment variables to your system:
  `INCLUDE_XERCES_C = [xerces_home]\src`
  `LIB_XERCES_C_DEBUG = [xerces_home]\Build\Win32\VC7\Debug`
  `LIB_XERCES_C_RELEASE = [xerces_home]\Build\Win32\VC7\Release`

3.3.2 OpenSSL

Proxytest needs an OpenSSL version 0.9.7e. Testing caused some problems with other versions. To get a running version of the OpenSSL 0.9.7e, you will have to compile OpenSSL yourself.

Since the config files are generated by a Perl script, you first have to get a running Perl environment onto your Windows system.
There is a free distribution from http://www.activestate.com/, which is simple to set up and will suffice for our purposes. To get Perl running, just go to the above mentioned website and look for a download link to **ActivePerl 5.8.7** or newer. Activestate asks you to register but registration is not mandatory. Proceed to the download with the '**NEXT**' button and then choose a binary package with either the MSI installer or a zipped package. You can keep the default options given by the installation procedure. After installation, do not forget to add the path of the `\bin` subdirectory of your Perl installation to your `PATH` environment variable! Unfortunately, this is not done automatically during the installation. Indeed, it is necessary to be able to use the Perl interpreter from anywhere.

- To get a source distribution of OpenSSL, go to:
  http://www.openssl.org/source/
- Unpack this archive into any directory, for example:
  `d:\coding\libsc\openssl-0.9.7e`
  The chosen directory will be referenced as:
  `[openssl_home]`
- Open a command prompt window ("DOS box") and change into this directory
- Automatic generation of the config file is initiated by executing:
  `Perl Configure VC-WIN32`
- For the following step, it is necessary that you have the ml.exe Microsoft Macro Assembler in your path. Although this should have occurred automatically upon installation of Visual Studio 2003, it sometimes causes problems. So, if the following commandcauses error messages, you can fix the problem by executing the batch file:
  `C:\Programs\Microsoft Visual Studio .NET 2003`

  `\Common7\Tools\vcvars32.bat`
  This batch file will set all environment variables properly so that VS2003 works again. Be sure to call this batch file from the same command prompt window! Environment variables set from a command prompt window are only set for that command prompt instance!
- To finish the configuration process of OpenSSL, you need to run the following from the command prompt:
  `ms\do_masm`
- This will take some seconds. After the script has run, you can start the make process:
  `nmake -f ms\ntdll.mak`
  This will start the command line compiler of Visual Studio 2003 and generate the libraries.

- These libraries will be placed in:
  `[openssl_home]\out32dll`
- Add the following environment variables to your system:
  `INCLUDE_OPENSSL = [openssl_home]\inc32`
  `LIB_OPENSSL = [openssl_home]\out32dll`

3.3.3 CppUnit

For compilation of CppUnit, the source code of version 1.8.0 or higher is needed. For this document, version 1.10.2 was used. You can get CppUnit as a zipped archive from:
http://sourceforge.net/projects/cppunit/
Documentation for CppUnit needs to be generated using Doxygen. A short explanation of how to do this is given in chapter 6.

Using Visual Studio .NET 2003

To compile CppUnit using Visual Studio .NET 2003, you should unzip to any directory (referenced as [cppunit_home]). Compilation with VS2003 is quite simple:

- Start VS2003 and open:
  `[cppunit_home]\examples\examples.sln`
- If asked if you want to convert you should answer '**Yes to all**'.
- Make HostApp the 'StartUp Project' and compile all.
  You can do so by pressing <F5>, which will compile/build all.

*** ToDo**
*** missing more detailed instructions**
*** how to run and test**
--- snip (from the CppUnit docu) ---
- in VC++, Tools/Customize.../Add-ins and macro files/Browse...
- select the file lib/TestRunnerDSPlugIn.dll and press ok to register the add-ins (double-click on failure = open file in VC++).

--- snip (from the CppUnit docu) ---
-
-

3.3.4 zLib (Windows)

Proxytest only needs the static libraries of zlib-1.2.3. Compiling zlib-1.2.3 from source code is not necessary, and we explicitly recommend against doing so, since it is a bit tricky. There is a binary distribution for VS2003 that completely suffices for our purposes. However, you will need to get the zlib-1.2.3 sources, as you need the header files. To hook this distribution into proxytest you have to follow these steps:

- Download the binaries from
  http://www.winimage.com/zLibDll/zlib123dll.zip
- To get the sources of zlib-1.2.3, unpack this zip archive to any directory, for example to
  `d:\coding\libsc\zlib-1.2.3`
  This directory is referenced in this document as [zlib_home]
  http://www.zlib.net/zlib-1.2.3.tar.gz
  Unpack these sources into
  `[zlib_home]\src`
- Change into
  `[zlib_home]\static32`
- Rename the file
  `zlibstat.lib` into `zlib.lib`
  This is necessary because the file is referenced from VS2003 as zlib.lib and nothing else. Alternatively, you could change the reference in VS2003, but each new checkout of the proxytest sources will overwrite your changes.

- Add the following environment variables to your system:
  ```
  INCLUDE_ZLIB = [zlib_home]\src
  LIB_ZLIB = [zlib_home]\static32
  ```

As an alternative to this procedure, you can instead download an archive that has already been set up from: http://www.anon-online.de/zlib/zlib.zip
The archive at that location has already been prepared according to the above instructions, and placed in a zipped directory. This archive can be unpacked into any directory. The chosen directory is referred as [zlib_mix]. It is not necessary to rename anything. You simply have to add the following to your environment variables:

```
INCLUDE_ZLIB = [zlib_mix]
LIB_ZLIB = [zlib_mix]\static32
```

+++ ToDo – provide the mentioned zip-file / upload it to the webserver +++

In Unix or Linux, the zLib library is normally already included. If it is not, you should get it with your distribution's package manager.
If you want or need to build the zlib-1.2.3 libraries completely on your own, you will run into problems with the dependencies of proxytest. As mentioned, building zlib-1.2.3 on your own is tricky and sources from various different sites are necessary.

3.3.5 pthreads (Windows)

Proxytest needs both the static and dynamic linked libraries and the header files of pthreads-win32. It's not necessary that you compile the libraries yourself - there are pre-compiled win-32 versions that work perfectly. We tested both 1.5.0 and 2.7.0 and they both run well. So versions in between are also likely to run well.

- To get the binaries and header files, go to:
  ftp://sources.redhat.com/pub/pthreads-win32/
  Choose one of the pthreads-w32-x-xx-x-release versions
- Run the sfx and choose an empty directory by using the 'Browse' button. Then click on '**Extract**'.
- We recommend, for example
  `d:\coding\libsc\pthreads-2-7-0`
  The chosen directory will be referenced in this document as `[pthreads_home]`
- Change into the directory
  `[pthreads_home]\Pre-built.2\lib`
  and rename the file `pthreadvc2.lib` to `pthreadvc.lib`
  Again, this is a problem caused by the references from the VS2003 project file. As previously mentioned for zlib, you can change the VS2003 references, but they will be
  overwritten each time you checkout new versions of *proxytest* from CVS.
- add the following environment variables to your system:
  `INCLUDE_PTHREADS = [pthreads_home]\Pre-built.2\include`
  `LIB_PTHREADS = [pthreads_home]\Pre-built.2\lib`

3.3.6 Setting the path variables

If you have followed the instructions above on compiling the libraries for Windows and you added all of the environment variables as described, most of the configuration is already done.
You just have to add as a global variable:

```
LIB_PROXYTEST = %LIB_XERCES_C_RELEASE%;%LIB_XERCES_C_DE-
BUG%;

    %LIB_OPENSSL%;%LIB_ZLIB%;%LIB_PTHREADS%;
```

and finally add to your path variable:

```
PATH = %PATH%;%LIB_PROXYTEST%
```

4 Compiler

4.1 C++

Using Windows, it is not necessary to install an extra compiler since it is already included in VS2003 or Cygwin. Using Linux or MacOS, you need g++ from the GNU compiler collection in version 2.95 or higher. This compiler is usually shipped with every installation of Unix or Linux.

4.2 Java

For the purpose of compatibility on many operating systems, the JAP client has to be compiled with a Java compiler no newer than JDK 1.1.8. This version is the last available Java version for MacOS 9. Since there are some bugs and difficulties in the 1.1.8 JDK version for Linux, we recommend using version 1.4.0 or higher for execution and testing.
As a developer, you are restricted to use classes that are also available in the 1.1.8 JDK version of Java despite the bugs. Thus, the 1.1.8 version from the Blackdown Java port is recommended for compilation.

system	***version***	***source***
Windows	1.1.8_10	http://java.sun.com/products/archive/jdk/1.1.8_010/
		http://java.sun.com/products/archive/jdk/1.1.8_010/jre/
Linux	1.1.8_v3	http://www.ibiblio.org/pub/mirrors/blackdown/JDK-1.1.8/i386/v3/
MacOS	1.1.8	http://developer.apple.com/java/classic.html

For installation, follow the automated scripts or install procedures. If automatic installation is not available, you will need to copy both the compiler and the runtime into one directory. We recommend adding the version number of your JDK to the directory name, e.g. `jdk1.1.8` Newer JDKs are recommended for testing and debugging purposes. As mentioned previously, it is also possible to install more recent JDKs/JREs on your system for testing purposes. Just be sure not to use libraries / classes / methods that are not available in Java 1.1.8.

5 IDEs

The Eclipse IDE is written in Java and available for all relevant platforms (Linux, Windows, MacOS). As it supports both Java and C++, we recommend it for developing of all AN.ON sub-projects. For Windows, we also use Microsoft Visual Studio 2003 for Mix development, but the installation of the needed libraries is tricky and we only recommend to use this IDE if you are a very experienced programmer. Our personal favourite for Java development is Borland JBuilder - if you do not mind using two different IDEs for Java and C++, and if you are not against closed-source programs, this will be your best choice for the Java sub-projects.

5.1 Specific development requirements on Microsoft Windows

For Java development, either Eclipse or JBuilder are a good choice and both easy to install. If you use Eclipse for C++ development, too, you have to run it under the Unix emulator Cygwin, which prodives the tools and libraries needed to checkout and compile the C++ code. Instead, you may use Microsoft Visual Studio for C++ development and an arbitrary extra tool for checking out the code from the source repository.

5.1.1 Installation of Cygwin

Download and install Cygwin from http://www.cygwin.com/ first. Execute `setup.exe` and confirm the default settings (Install from Internet, installation path etc) until you see the package selection window. Keep in mind that this installation might easily consume 500MB, so try to exclude unwanted packages. You will, however, need all the compiler tools (especially `gcc` and `make`) AND the C++ libraries mentioned in section 3.

5.1.2 Setting up a CVS client

If you use Microsoft Visual Studio for development, you will first need to manually download a CVS client. A really easy-to-use client that integrates completely in the Windows Explorer is *TortoiseCVS* (http://www.tortoisecvs.org/).

In the following, we provide short insturctions for the use of another tool, *WinCVS*, (http://www.wincvs.org/), which is distributed under the LGPL license:

Choose a current version (see the 'Latest Recommended Release' on the website). Download a release with an installer. Choose a mirror near your location. Start the installer and follow its instructions. Nothing additional is needed.

Now, to get the source code from CVS, follow these instructions:

- Start WinCVS and select a directory in which to put your copy of the code
- Enter this directory in the upper dropdown field, which also allows you to browse through your system directory

- To checkout a project / component choose **'Remote'** -> **'Checkout module'**

WinCVS – Settings for proxytest	

- In the 'Checkout settings' window that appears, enter:
  Module... : proxytest *(or any other module name – see above)*
  Local folder... : *any directory you want to put the sourcecode to*
  CVSROOT : :pserver:anonymous@cvs.inf.tu-dresden.de:/home/sk13/cvssource

- Be especially careful about the colon, which has to be the first character before "pserver"
- Clicking the 'OK' button will start downloading the sources
- This procedure will work for each of AN.ON's modules (ie. JAP, MixConfig, InfoService)

5.1.3

5.1.4 C++ development using Visual Studio 2003 .NET

The project files with working settings for Visual Studio 2003 .NET are shipped with the *proxytest* code.

Visual Studio 2003 .NET – Opening proxytest and setting the 'Startup project'	

- Start Visual Studio 2003 and open the file `proxytest.sln` as shown above
- Choose `proxytest` as your startup project and start a compile / make
- This should lead to a binary in the subdirectory `\windows\Debug_Build_Test`
- On errors, you can change settings and the paths to header / library files by manipulating the project settings by choosing: **Project** -> **Properties** from the upper menu bar
- For example, the paths to header files are set at:
  C/C++ -> **General** -> **Additional Include Directories**
- And libraries are set at:
  Linker -> **Input** -> **Additional Dependencies**

Visual Studio 2003 .NET – Project settings

5.2 Eclipse IDE

5.2.1 Java Runtime Environment

Eclipse 3.1 requires the Java 1.5 (J2SE 5.0) runtime environment. If Java 5.0 is not yet provided by your distribution as a package, you can download it from:
http://java.sun.com/j2se/downloads/
Please follow the installation instructions shipped with the archive. If an older version is already installed on your system, this is not a problem. You may install as many JDKs simul-

taneously as you want in different directories on your system. Normally they will not interfere with each other.

5.2.2 IDE installation

After having set the basic runtime environment, you may proceed with the installation of the Eclipse IDE. To get Eclipse, go to:

http://www.eclipse.org/downloads/index.php

To accelerate your download, choose a mirror near your location. After having finished the download, unpack the archive. For Linux, you may do this like this:

```
tar xfz eclipse-SDK-3.1-linux.gtk.tar.gz
```

Then, move the new folder to your programs directory, for example to `C:\Program Files\eclipse`, `/opt/eclipse` or maybe `/usr/share/eclipse`

On Linux, you may need root access to move the directory:

```
su
mv ./eclipse /opt/eclipse   (for example)
exit
```

Open a shell to create a link `/usr/bin/eclipse` to the Eclipse program file to get a quick access to it. If you use Windows, you have to open the Cygwin shell. A link may be created by

```
ln -s /cygdrive/c/eclipse/eclipse.exe /usr/bin/eclipse (for example)
```

if Eclipse was copied to c:\eclipse on a Windows system. Now simply type “eclipse” anywhere in a shell to start Eclipse. You may also create a graphical link for your desktop or start menu: On Windows, create a text file /usr/bin/eclipse.sh with the content

```
/usr/bin/eclipse
```

Make it executable by typing

```
chmod a+x /usr/bin/eclipse.sh
```

Now, create a desktop link with the following call (replace %cygdir% with your Cygwin installation path)%:

```
%cygdir%\bin\bash %cygdir%\bin\eclipse.sh
```

For Linux, please refer to your window manager help on how to generate desktop links.

5.2.3 Configuration for Java sub-projects

To get the current development version of JAP, there is a CVS with anonymous checkout permissions. Eclipse 3.1 provides a built-in CVS client that is easy to use.

- Start Eclipse, open the 'File' menu, choose 'New' and select 'Project'
- In the 'New Project' window, choose the 'CVS' and 'Checkout Projects from CVS' wizard
- Proceed by clicking the '**Next >**' button

- Enter the following values and proceed by clicking '**Next >**' again

```
Host:                          cvs.inf.tu-dresden.de
Repository path:  /home/sk13/cvssource
User:                          anonymous
Connection type:  pserver
```

- Finally, choose the module that you want to import (e.g. 'JAP', 'MixConfig',)

Eclipse – CVS Checkout settings for JAP	

- It may take a few seconds until the code is downloaded and imported into Eclipse 3.1

Since there are no configuration files for Eclipse 3.1 in the CVS source yet, you have to adapt JAP to your systems settings. For running and compiling JAP, the following libraries are needed:

ApacheBZIP2	ApacheXMLrpc	HttpClient	JunitX 5.1	Mail (??)
ApacheFTPClient	BoucyCastleLight-ForJAP	Jama	Log4J	MRJClasses

As mentioned previously, you can download all of these libraries in the proper versions from the AN.ON website. Choose the packages for your desired component.
http://anon.inf.tu-dresden.de/develop/sources_en.html
Please save these libraries into one common directory. Then change back to Eclipse and open the Project -> Properties and navigate to 'Java Build Path' and choose the 'Libraries' tab. Get rid of all the 'unbound' references by selecting each of them and pressing the 'Remove' button on the right. After this again add the external libraries which you have downloaded before by pressing the 'Add External JARs...' button and choosing the folder where you saved the libraries. Select all of them and press 'Open' in the file dialog.
Do not forget to add the 'JRE System Library' you want to execute the program on. In the screenshot below this is for example 'Java-1.5.0.sun-1.5.0_03' (the last entry). Do so by pressing the 'Add Library...' button, selecting 'JRE System Library' and normally choosing your 'Workspace default JRE'. Confirm with the 'Finish' button.
This JRE settings would allow you to test the proper execution of changed / improved JAP versions on older Java runtimes.

Eclipse – Properties - Project / Libraries for JAP

After having done this, the project can be compiled / run. The tests should run correctly as well. To start JAP on your machine, open the '**src**' and then the '(default package)' from the '*Package explorer*' on the left side. Choose `JAP.java` or `JAPMacintosh.java` and press the right mouse button. Then choose: '**Run as**' -> '**Java application**' or use the key combination **<Shift> + <Alt> + <J> <X>** to start it.

Installing FatJar Plugin

Eclipse doesn't automatically integrate external libraries into your .jar when exporting your Project. This functionality is provided by an plugin called “FatJar”. To install this plugin, go

to http://sourceforge.net/projects/fjep and download it. Unzip the “plugins” folder and move it to the plugins folder of Eclipse. You will have to start Eclipse with the „clean“ option („eclipse -clean“), otherwise the plugin won't be found.
Right click on your project and you will see the new entry „Build fat jar“. This is also available under File -> Export. Use this instead of the standard jar.
For more information on this plugin, visit http://fjep.sourceforge.net/ (description) and http://fjep.sourceforge.net/fjeptutorial.html (tutorial, just have a look).

5.2.4 Configuration for C++ sub-project 'proxytest'

Installing CDT

For C/C++ development, you will need to download CDT, a plugin for Eclipse that provides a fully functional C and C++ IDE for the Eclipse platform. More information about CDT can be found at:

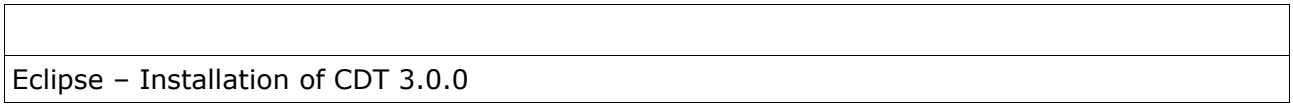
http://www.eclipse.org/cdt/

Downloading and integrating CDT into Eclipse is simple. You can use the built-in update manager.

- Start Eclipse, open the 'Help' menu, choose 'Software Updates' and select 'Find and Install...'
- In the 'Install / Update' window, choose the second point, 'Search for new features to install' and click '**Next >**'
- Click the first button 'New Remote Site...' and enter the following into the window that appears:

  Name: CDT 3.0.0 for Eclipse 3.1
  URL: http://download.eclipse.org/tools/cdt/releases/eclipse3.1

  then proceed by clicking the 'OK' button



Eclipse – Installation of CDT 3.0.0

- In the new window, be sure that the checkbox '*CDT 3.0.0 for Eclipse 3.1*' is selected or nothing will be installed
- Proceed by clicking the '**Finish**' button
- In the new window, select the checkbox '*CDT 3.0.0 for Eclipse 3.1*' again,
- The branches will be automatically selected. To proceed, click '**Next >**' again
- Accept the license agreement and click '**Next >**'
- Finally, you need to confirm where you want to install CDT and click '**Finish**'
- It takes a while until all the components have been downloaded
- In the 'Verification' window that may appear, you can proceed by clicking '**Install All**' even though CDT has no digital signature yet
- This verification may not be necessary in the future when the package has been signed
- Restart Eclipse so that the configuration is updated

Getting the code from CVS

Having set up the IDE for C++, you can now get the proxytest source code from the CVS server. If you have checked out the 'JAP' code before, there should already be an existing profile for the repository settings. If you have not done this yet, you can follow the instructions given above. You only have to exchange the module name with 'proxytest'.

Eclipse – Checkout proxytest from CVS

Be sure to put the source code in a new directory! It cannot be placed in your Eclipse workspace! To put the source code in a new directory, in the last window at 'Select the project location', be sure to **deselect** 'Use default workspace location' and enter another directory within your /home that you have already created.
This step is very important because otherwise it is not possible to create a new 'Standard C/C++ Project'.

After these steps, close the project by selecting it in the left 'Navigator' perspective and choosing 'Close Project' from the context menu. To remove the entry too, select 'Delete' from the context menu. Be sure that 'Do not delete contents' is selected when you click 'Yes' to confirm it.

Checking out the code on a windows platform you will have to use dos2unix to convert *config.h.in* from windows linebreaks to unix linebreaks. To do so, open cygwin and call
dos2unix /path/to/config.h.in

Because the dependencies on libraries and code for the Eclipse C++ project are based on the 'proxytest' make file, you have to run the configure script before you import the project into Eclipse.
To do so, open a console and change into the directory where you put the sources from CVS.

```
cd proxytest        (change into the directory)
./configure         (run the configure script)
```

It will take some time until the make file is generated.
Now you can import the source code as a 'Standard C/C++ Project'.

Eclipse – Create new 'C++ Make Project'

- Be sure to **deselect** 'Use default' and enter the directory where you put the code that you checked out from CVS
- Proceed by clicking the '**Finish**' button

- If you are asked whether you want to open the C/C++ perspective, you can answer with '**Yes**'

Compiling and Running proxytest

If everything worked, you should now be able to open each class and the syntax should be highlighted. As you start to program, you will realize that code completion is enabled from the Java perspective of Eclipse.
Although it should be disabled by default, when changing the perspective, the 'Build automatically' option could be enabled. This setting won't work with C++ because its build process is quite different from the Java build process. To change the settings, open the 'Project' menu from the upper menu bar and disable this option manually.
If the right versions of the necessary libraries are installed on your system in the correct manner, you should now be able to build the project.
To build a binary, select the project in the left-side 'Navigator' and choose 'Build Project' from the context menu.

Eclipse – Building 'proxytest'

If all went well, you should now be able to run the MIX server. Before starting, you have to set the configuration files. You can choose to run MIX as the first or last MIX in a cascade, with or without logging functionality. Additional or alternative configuration files can easily be generated with the MixConfigTool. The shipped configuration files can be found in:

```
/proxytest/documentation/SampleConfiguration/
```

Starting the MIX with a specific configuration from within Eclipse 3.1 is done as follows:

- Set the focus to the 'Navigator' perspective and click 'mix'. A small input box will appear at the bottom of that window and the cursor will jump to the file name that most closeley matches your input (this is a built-in search function)
- You should find an entry like
  ```
  >mix(Binary)
  ```
- Select this and choose from the context menu '**Run as**' and '**Run...**' (the last menu point)
- In the 'Run Manager' window that opens, navigate to the 'Arguments' tab and, for example, to run the MIX as last Mix, enter:
  ```
  --config=/documentation/SampleConfiguration/LastMix.xml
  ```
- Click the '**Run**' button
- In the '*Console*' perspective, you should now see the starting messages from the MIX

Eclipse – Running 'proxytest'	

•

5.3 Borland JBuilder IDE

The *JBuilder* IDE from Borland is a good choice for Java development, as it is a platform independent, well-engineered piece of software and avaliable both in free and commercial versions.

5.3.1 Download and Installation

Download the program and a registration file from the following address:

Tool	***Version***	***Quelle***
JBuilder	X, Foundation	http://www.borland.com/products/downloads/download_jbuilder.html

You will have to register at Borland to start the download. We recommend that you also install the JBuilder help files, that are available in an extra package.

If you have problems to run the installer under Linux, you should set execution rights via `chmod +x`.

5.3.2 Programmstart

When first starting Jbuilder, it needs to be registered by reading the registration file. In the registration wizard, click „Have Activation File“, then „next“, and then enter the path to the registration file. Clicking on „Finish“ then completes registration.
Links to the IDEs are automatically created during the installation. For Jbuilder on Linux, the link needs to be modified in order to insure that the JDK 1.1.8 is found correctly.: `JAVA_HOME="" /usr/JBuilderX/bin/jbuilder` (The exact path to JBuilder depends on your system/Linux distribution and may therefor vary)

5.3.3 CVS Checkout

CVS stands for „Concurrent Versioning System. It is a system for version control of code., intended to help programming teams coordinate their work.. All code is saved on a central CVS-server, and downloaded (checked out) by IDEs or other clients.

Note for JBuilder on Linux: The CVS-Integration for JBuilder contains a Bug, which may lead to CVS not being found by JBuilder. This is most likely caused by a version conflict between several versions of CVS installed on the same Linux system. This problem can be resolved by replacing the CVS executable used by Jbuilder with a symbolic link:

```
su
cd [JBUILDER_HOME]\bin
rm cvs
ln -s usr/bin/cvs cvs
```

Before a component can be edited or compiled, you need to create a project and check out the source code from CVS. AN.ON consists of the following modules (Name is case-sensitive!):

Jap
MixConfig
proxytest

The corresponding Jbuilder project will be created by checking out the code like this:

Figure 1 "File->New"

Figure 2 "Pull Project form CVS"

Figure 3 Choose a target directory (has to be empty) and click on "Checkout - for development work"

Figure 4 Connect type: "PServer", Server: "cvs.inf.tu-dresden.de", User name: "anonymous"

The module „Jap“ will create several projects. To be able to update the each project via

Figure 5 If you are asked for a passwork, simply ignore it by leaving the password field blank and clicking "OK" .

CVS, activate it (by double clicking it in the explorer window on the left-hand side) and register it for CVS by choosing „Select Project VCS“ in the „Team“ menu.

Figure 6 Repository: "/home/sk13/cvssource", Module name: [Jap, proxytest, InfoService, MixConfig]; Case-sensitive!

Figure 7 Either choose "The main branch" or pick a suitable Branch in the repository by clicking on "Scan"

5.3.4 Formatting Options

Most formatting guidelines contained in the programming standards can be automated by the IDE. [PRICHTLINIEN]. Simply import the file „[IDE].codestyle“, and import it like this:

Figure 8 Project->Project Properties

Figure 9 Choose "Formatting" (JBuilder).

(Caution: Autoformatting in the C++-Builder is buggy, do not use it yet!)

Figure 10 (Formatierung in the C++Builder)

Figure 11 Open "[IDE].codestyle"

JBuilder then allows you to select single classes or whole packages using the class browser, and to format them using the „Format package“ command. C++Builder can only format single classes. To do so, open the class, and choose „Edit/Format all“.

5.3.5 Compiler integration

The complier needs to be registred with the IDE, and set for the project:

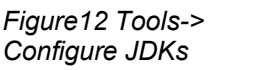


Figure12 Tools-> Configure JDKs

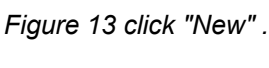


Figure 13 click "New" .

5.3.6 Integrating the libraries

Figure 14 Click on "..." and choose the path to the JDK.

Each Component depends on several external libraries, which first need to be set up in the IDE:

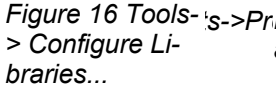


Figure 16 Tools-> Configure Libraries...

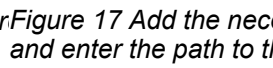


Figure 17 Add the necessary libraries using "New" . Then click „Add" and enter the path to the library's Jar file

Figure 18: Choose a library, and click "OK".

5.3.7 Compilation

If everything worked so far, you can now compile the project:

Figure 19 The "Build"-Button.

Figure 20 Any "Deprecated"-Messages can safely be ignored

Jbuilder will put the compiled files into the „classes“ directory, C++Builder into the directory Linux\[Debug]Build or Windows\[Debug]Build.
Tip: If you should enconter an error during compilation, it is recommended to delete the contents of those directories before compiling again or starting „Rebuild“.

5.3.8 Ausführen

You can now run the project by choosing *Run->Run Projects* .

6 Doxygen

Doxygen is an open source documentation tool for different programming languages. For graphical output it needs *Graphviz,* which is also open source and needs to be installed after *Doxygen*.

6.1 Download

tool	***version***	***source***
Doxygen	1.3.6	http://www.doxygen.org
GraphViz	1.1x	http://www.graphviz.org

6.2 Installation

Using Linux, we recommend installing Doxygen and Graphviz from the pre-compiled packages for your distribution.
Using Windows, you can download the binaries from the websites listed above. Convenient auto-installers are shipped with the Windows versions.

Run the main program by starting '*Doxywizard*'. In the GUI, select '*Load*' and choose the [proxytest_home] folder. From there, select the *.doxy file from which you want to generate the documentation.
In the source folder of each AN.ON component, there is a file called

`<components-name>.doxy`

This file has to be opened with the Doxywizard.
Finally, click the '*Start*' button to generate the documentation. This may take a while. You can follow the output of Doxygen in the lower 'Output' window. If errors occur, they will also appear there.

7 Unittest environments

7.1 JUnit

The unittests can be found in the "test" directory and can be started by right clicking on the desired class:

Figure 21 Run using "[project]"

Figure 22 This is what you should see if all tests pass.

Tip: Unfortunately, the JUnit support built into JBuilder cannot be used with JDK 1.1.8 instead of the previously described process. Use of a newer JDK would be necessary for that.

7.2 CppUnit

In C++Builder, double click on the "AllTests" project. and then execute the Unittests as you would a normal project:

Figure 23 Double click on the "AllTests" project. The test classes are in the "test" directory.

Figure 24 This is what you should see if all tests pass.

8 Running the tests

8.1 Architecture

On the local machine, we can simulate a Mix-cascade by running two Mixproxy instances in parallel. You will also have to start the JAP client and set it as proxy for your browser. If your web browser can access the data distributed by the Apache web server (e.g. the Apache documentation), the configuration is working properly.
There is no access possible from the internet.
The testing environment consists of the following elements:

- Infoservice – instance
- 2 Mixproxy instances (cascade)
- Apache Web server
- JAP – instance
- any Web browser (recommendation: Mozilla)

8.2 Configuration

8.2.1 Config files

type filter text
The necessary config files do not need to be created by hand but are shipped with the CVS module. You can find them at:

```
[proxytest_home]/docs/SampleConfiguration
```

The following configurations and directories can be found there:

- `logFirstMixWihth (d)`
- `logLastMix (d)`
- `FirstMix.b64.cer`
- `FirstMix.xml`
- `FirstMixWithLog.xml`
- `InfoService.properties`
- `jap.conf`
- `LastMix.b64.cer`
- `LastMix.xml`
- `LastMixWithLog.xml`
- `private.pfx`
- `public.cer`

8.2.2 InfoService

The following certificates and config files can be found at:
`[proxytest_home]/documentation/SampleConfiguration/`

- FirstMix.b64.cer
- LastMix.b64.cer
- private.pfx
- public.cer
- InfoService.properties

There, you will also find the logfiles for the InfoService component.

8.2.3 Mix cascade

The following config files for parts of a Mixcascade can be found at:
`[proxytest_home]/documentation/SampleConfiguration/`

- FirstMix.xml
- FirstMixWithLog.xml
- LastMix.xml
- LastMixWithLog.xml

Depending on the function of the MIX as first or last mix, the logfiles of the MIX server can be found in the `/logFirstMix` or `/logLastMix` subdirectory.

8.2.4 Apache HTTP Server (version 1.3.x oder 2.x)

Normally on Linux systems, the Apache server is running in the default configuration.
If it is not running yet on your system, it is quite simple to install with the help of your distribution's package manager.
Using a Windows system, you first have to get Apache from http://httpd.apache.org and install it.
You can proceed directly to this URL to get the 1.3.29 version for Windows.
http://www.apache.de/dist/httpd/binaries/win32/apache_1.3.29-win32-x86-no_src.exe
During the installation process, you should keep the preselected default options.

Starting and stopping the server is done as follows.

(Linux)
Open a console and enter:

```
su                                  (to get root)
/etc/init.d/apache start            (starting Apache)
/etc/init.d/apache stop             (stopping Apache)
```

(Windows)
There are shortcuts in your 'Start' menu for both starting and stopping.

8.2.5 JAP

The configuration file jap.xml (also in the CVS of the JAP project) has to be placed in the home directory of the user. If non-existent, the file will be generated automatically with default values upon the first use of JAP.
For running the tests, you will have to overwrite it with the values you need.
Debug- or logging output from JAP will be printed on an optional console, which has to be activated explicitly, or in the debug-view of your IDE. To turn on the debug output from JAP, activate the checkbox in 'Misc' in the JAP window.
The connection to your Mixcascade is set by activating the 'Anonymity' checkbox within the main window of JAP.

8.2.6 Web browser

Your browser has to be configured so that web access done exclusively through JAP. To do so, add JAP as http-proxy on address 127.0.0.1 at port 4001 (default).
Furthermore, you have to ensure that only this proxy is used for accessing the web (no other proxies should be set). You can ensure this by turning JAP off and trying to connect to any website that is not in your cache.
In order to avoid any external errors, we also advise turning off any caching in your browser.

8.3 Start

For the same reason of avoiding external errors, you should turn off any running firewall. Needless to say that tests should never be run on a production system!

Now the time has come to open and run the components in their native development environment with each component in its own instance if the IDE.
Apache can run the entire time as a daemon / background service.

- InfoService (JBuilder)
- First Mix (VC7)
- Last Mix (VC7)
- JAP (JBuilder)

Finally, you have to activate 'Anonymity' in the JAP client. JAP should not generate any errors in the console.

8.4 Test

The correct configuration of the test environment can be tested with a simple trick. First, start JAP and make sure that it is running as a proxy (activate 'Anonymity'). Now start your browser (after configuration as described in 8.2.6) and open the URL:

http://localhost

You should see a website (normally the documentation of Apache) generated by your locally running Apache.

Now uncheck the 'Anonymity' checkbox in JAP and in your browser reload the same address. An error message generated by your browser should appear. There is no longer a proxy running in the background, so your browser cannot find any websites.

If this does not happen, you either did not set JAP as a proxy in your browser (see 8.2.6), or there is still a way your browser has access to the internet. Another problem may be that the site is already cached by your browser. Clear the browser cache and try again.

9 Payment Instance Setup

9.1 System requirements:

- a working Java environment
- a running Postgresql server
- the JDBC driver for postgresql
- the OpenSSL tool (unless you already have a key file and certificate)

9.2 You will need:

- BI.jar (contains all Java code for the payment instance)
- the jar files of all necessary libraries
- a key file and certificate for the JPI
- a jpi configuration file
- a postgresql database and server

9.3 Creating the Jar file

If you just want to run the JPI without making changes to the code, simply download it from the project homepage / ask a project representative for the code.
To build a new jar file after making changes in Jbuilder:

1. Do a rebuild
   Menu “Project”, choose “Rebuild Project Bezahlinstanz.jpx”)
2. Set up the jarfile
   In the Project Pane, pick the “Project” tab. Find the file “BI.jar”, right-click on it, and choose “Properties”. Under “Content”, either pick “Include all classes and resources”, or include at least the filters “anon/crypto/*.*” and “jpi.*.*”. IMPORTANT: Under “Dependencies”, pick “Include All” for the Postgres and BouncyCastle packages.
3. Rebuild jar file
   In the jar file's context menu, click on “Rebuild”.

9.4 Creating the key file and certificate

Use the following commands in a shell:

```
openssl dsaparam -genkey 1024 -out bi.key
openssl req -x509 -new -out bi.cer -key bi.key -subj "/CN=BI/"
openssl pkcs12 -export -in bi.cer -inkey bi.key -name BI -nodes -noiter -out
bi.pfx
```

9.5 Creating the Postgres database

Use the following commands:

```
su postgres
createuser -A -D biuser
createdb -O biuser bidb
```

To confirm the creation of the database, log into your newly created database by typing

```
psql -U biuser -d bidb
```

The database will be empty, the necessary tables will be created by the JPI itself.

If conecting to the database does not work, try editing the `hba.conf` file (under linux, usually in `/etc/postgresql`, or in your postgres data directory) to allow connecting with a password. Example:

```
# All other connections by UNIX sockets
local   all         all                                               password
# All IPv4 connections from localhost
host    all         all         127.0.0.1         255.255.255.255   ident password
```

Note that Postgres looks only for the first line matching a connection in pg_hba.conf
After editing hba_conf, you need to restart postgres using `pg_ctl reload` (usally found in `/usr/lib/postgresql/bin`)

9.6 Creating the JPI configuration

The JPI configuration can be named anything you like and is a simple textfile with all lines in the format of option = value.A sample configuration named config.example is contained in the CVS project and can be used as a starting point.

Note that options regarding the volume plans and payment options offered are intended to be edited using the PIG (payment instance GUI) and will only be read from the configuration file if you start the JPI using the parameter new.

You will need to adjust the following options:

id	official identifier within the AN.ON system, used by JAP, AI and InfoService
name	human-readable name of the JPI
infoservices	Hostname and port of the InfoServices that the JPI will register itself with (Format: host:port, host2:port2,...)
japlisteners	How the JPI can be contacted by JAPs (Format: host:port, host:port,...).
ailistener	connection interface towards the AI (Format: host:port)
mclistener	connection interface towards the MixConfig tool
paypallistener	connection interface for Paypal to send IPN notifications – make sure the same value is set in your Paypal merchant account
virtualXXXlistener	Every listener interface has a corresponding virtual listener interface. If you do NOT use a cluster setup for the JPI, you can leave out the virtual listeners, or set it to the same as the real listener For cluster setups, the real (non-virtual) listener is what the JPI itslef uses to listen to; the virtual listener is used by the outside world to contact it, and what is sent to the InfoService
Database options	
dbhost	hostname or IP of the JPI's postgres database (most likely localhost)
dbport	Postgres port (most likely the default 5432)

dbname	Name of the dababase to be used by the JPI (should be bidb)
dbusername	Owner of <dbname>, should be biuser
dbpassword	<dbusername>'s password
Keyfile options	
keyfile	Path to the .pfx key file (just the filename if it's in the same directory as the JPI jarfile)
keyfilepassword	Password for <keyfile>, as entered when <keyfile> was created using OpenSSL. For extra security, leave blank to be asked for the password on startup.
Logging options	
logfilename	self-explanatory
logfilethreshold	accepted values: 1 up to 7 (7 = maximum detail)
logstderrthreshold	just like logfilethreshold, for messages to be written to stderr
nr_of_logfiles	The jpi keeps rotating log files, i.e. BI.log is the newest, older files get names like BI.log.1, BI.log.2,... Nr_of_logfiles determines how many old log files you want to keep, 10 is a good idea
size_of_logfiles	Size of a single logfile in bytes, suggested value: 5000000
Textfiles options	
termsfile	Name of the html file(s) containing the terms and conditions. Needs to be valid xhtml! Will be read once on startup, changes require a restart The JPI will load files with names of <termsfile>_<language>.<termsfile-extension> e.g. termsfile=terms.html, termslanguages=de,en => will load terms_en.html and terms_de.html
termslanguages	comma-separated 2-letter codes, e.g. de,en
policyfile	html file(s) containing the cancellations policy, ananlogous to termsfile
policylanguages	comma-separated 2-letter codes, e.g. de,en
Misc options	
maxcascadelength	Integer. Maximum no of mixes that will be paid in one cascade
log_payment_stats_enabled	True or false: set true if you want to log statistics about traffic per Mix or Jap and month.

For Details on how to set other options regarding prices and payment, see [DIPLOBAIER]. However, these options should really be set using the JPI-GUI. Additional Options needed for configuring the paysafecard or Call2pay payment options are described in [DIPLOSCHRAML].

9.7 Starting the JPI

It is recommended to run the JPI inside a screen. To create a screen, type `screen -R jpi` (or any other name). Then start the jpi inside the screen, and return to the normal shell using Ctrl+A+D. To return to the shell, use `screen -list` for an overview over existing screens, and `screen -r <name>` to go to an existing screen.

Your Java classpath needs to include the jar files of the necessary libraries (Every single .jar-file, not just the directory of the libraries!). Alternatively, you can integrate the libraries into the single BI.jar file by using the BI.jar file's context menu in Jbuilder, going to “Properties”, then “Dependencies”, and choosing “Include all” for all libraries.

To start the JPI, simply use `java -jar BI.jar config.example`
If you use another configuration file, replace config.example with the path and filename to your custom configurations file.

IMPORTANT: When starting the JPI for the first time, add a command line parameter `new` (e.g. `java -jar BI.jar config.example new`). This will create the database tables, and populate the paymentoption tables in the database from the configuration file.

10 Accounting Instance Setup

10.1 System Requirements

- A working, running Postgresql Server
- Ability to recompile the Mix code

10.2 You will need

- The mix code (CVS module proxytest)
- the postgres-devel packages (should be OK if you have Postgresql installed)

10.3 Compiling the first Mix

1. Configure the Mix using ./configure –enable-payment
2. Recompile

10.4 Setting up the database

1. Create the User
   In a shell, use `createuser -A -D aiuser`
2. Create the database
   In a shell, use `createdb -O aiuser aidb`
3. Create the necessary tables
   Log into the database by typing `psql -U aiuser -d aidb`
   Then run the following commands:

```
create table costconfirmations (
     accountnumber bigserial,
     bytes bigint,
     xmlcc varchar(2000),
     settled integer,
     cascade varchar(200),
     primary key(accountnumber,cascade)
);
create table prepaidamounts(
     accountnumber bigint unique not null,
     prepaidbytes integer,
     cascade varchar(200),
     primary key(accountnumber, cascade)
);
create table accountstatus(
     accountnumber bigint,
     statuscode integer
);
```

or use the prepared script: `psql -U aiuser -d aidb < mixtables.sql`
To confirm the creation of the database, log into your newly created database by typing `psql -U aiuser -d aidb.` If it worked, log out again using \q.

If connecting to the database does not work, try editing the `hba.conf` file (under linux, usually in `/etc/postgresql`, or in your postgres data directory) to allow connecting with a password. Example:

```
# All other connections by UNIX sockets
local   all         all                                           password
# All IPv4 connections from localhost
host    all         all         127.0.0.1         255.255.255.255   ident password
```

Note that Postgres looks only for the first line matching a connection in pg_hba.conf
After editing hba_conf, you need to restart postgres using `pg_ctl reload` (usally found in `/usr/lib/postgresql/bin`)

10.5 Payment Configuration

Use the Mixconfig tool. When editing the xml configuration file by hand, the relevant sections can be found under <Accounting>

General Options	
SoftLimit	Minimum amount of prepaid bytes at which a cost confirmation is demanded from the JAP
HardLimit	The JAP will be kicked out if the number of prepaid bytes drops to this level. Obviously, needs to be set LOWER than SoftLimit. WARNING: A distance between SoftLimit and HardLimit of at least 400 000 is recommended to give Jap enough time to reply with a CC even when surfing fast
SettleInterval	Interval in seconds: How often you want to contact the payment instance to cash in the cost confirmations received from JAPs
PrepaidIntervalKbytes	How many kilobytes are paid in advance, i.e. whenever softlimit is reached, AI will request a CC for (current transferred bytes + prepaidInterval)
Database options	
host, port, dbname, username,password	self-explanatory
Pricing	
Price (shown in Mixconfig tool)	NOT to be set manually; instead, use the Mixconfig tool to use a valid, signed price certificate. If editing manually, be sure to insert the complete xml node for a price certificate, including the JPI's signature

10.6 Price Certificates

In order to set a price, you will need to:

1. Request the price

Use the MixConfigTool. Load a Mix configuration that contains your operator certificate, which needs to be registered with the payment instance (i.e. stored in the JPI's database). To to the payment Tab, find the “PriceCertificate” panel, click on “Change”, and enter your new price.

2. Have the JPI's operator confirm your new price

This will be done by logging into the PIG, and signing your new price.

3. Store the signed price certificate in your Mix's configuration

In the MixConfigTool, load your Mix's configuration, go to the PriceCertificate panel on the Payment tab, click “Update”. Select your newly signed price certificate, and click on “Use”. Alternatively, you can edit the mixconfig-file manually. In that case, the <PriceCertificate> node goes under <Accounting>, and the JPI's certificate needs to be written as a block (no spaces before or after each line of the certificate).

11 Payment Instance GUI (PIG) setup

11.1 Ruby

Check if ruby is already installed by running `ruby -v`. If not, go to `ruby-lang.org` to download and install the correct version for your operating system. Note that the version of Ruby shipping with Mac OS X 10.4 does NOT run Rails well, installing a newer version is required. On Debian, `apt-get install ruby ri irb ruby1.8-dev libzlib-ruby zlib1g rdoc` will install Ruby 1.8
On RedHat/Fedora/CentOS, `yum install ruby ruby-libs ruby-devel ruby-rdoc ruby-irb ruby-ri ruby-docs` should work.

Of course, you can also compile from the source code. Excellent instructions on how to install Ruby and Rails from source can be found at http://hivelogic.com/narrative/articles/ruby-rails-mongrel-mysql-osx (meant for MacOS X, but also applicable to Linux)

11.2 Rails

Rails is available as a ruby gem (Gem is the package management system for Ruby).
To install Gem, download the source code from `rubygems.rubyforge.org`. On Linux, you can use `wget http://rubyforge.org/frs/download.php/17190/rubygems-0.9.2.tgz` Unzip, and install with `sudo ruby setup.rb`

To install Rails, simply type `sudo gem install rails --include-dependencies`

In case you should get an error like “Could not find rails in any repository”, do `gem list rails –remote` first. This updates the list of avaiilable packages, afterwards the regular gem install rails command should work.

If you should have multiple versions of the rails gem installed, you have to tell your application which one to use in `config/environment.rb`, e.g. `RAILS_GEM_VERSION = '1.2.1'`

11.3 Database configuration

The ruby adapter for postgresql is available under http://ruby.scripting.ca/postgres.
It is also available as a gem, to install use `gem install postgres`.
Note that in order to compile, postgresql needs to be installed on the same machine. If necessary, set non-standard installation directories with e.g. `gem install postgres -- --with-pgsql-include-dir=/usr/local/pgsql/include –with-pgsql-lib-dir=/usr/local/pgsql/lib`

Alternatively, a pure-ruby implementation called postgres-pr offers lower performance, but easier installation: simply type `sudo gem install postgres-pr`

To configure Rails for using the correct database, edit the file database.yml in the /config directory under the rails application's root directory.
Use the following values:
adapter: postgresql (or postgres-pr, if you went with the pure-ruby version)
database: bidb
username: biuser
password: yourpassword
host: localhost

The adapter is called “postgresql”, no matter if you use postgres or postgres-pr

Pay attention to YAML syntax! (e.g. use two spaces for indentation instead of Tabs, no superfluous newlines at the end of the file)

11.4 Plugins

Run the following command to install the graphing library Scruffy:
`gem install scruffy`
The project's website can be found at scruffy.rubyforge.org
The other plugins should work just by copying the contents of your application folder over to the server. To install from scratch, use
`gem install login_generator`
and
`script/plugin install http://svn.cardboardrocket.com/paginating_find`

11.5 Java Bridge

Download YAJB from http://raa.ruby-lang.org/project/yajb.
Unzip and run `ruby setup.rb` to install.

Find the global variable JBRIDGE_OPTIONS in the bigui rails application (should be in `config/environment.rb, controllers/prices_controller.rb`), and check that `:classpath` points to the Java libraries you intend to use (at least `Jap.jar` and `BouncyCastleLightForJAP.jar`).

Set the constant PATH_TO_JPI_CERT in controllers/prices_controller.rb to the absolute path, including the file name, to the private key file of the JPI

11.6 Server

Go to the application's root directory, and start the integrated Webrick server with `script/server`. Open your web browser and find your application at http://localhost:3000. Use option -p to user a different port, or -e for a different environment, e.g.
`script/server -p 8080 -e production`

If you have problems using Webrick, try the alternative Mongrel server.
To install, type `gem install mongrel` (needs ruby version 1.8.4 or higher)
To start, go to the pig root directory, and type `mongrel_rails start`

`mongrel_rails -h` will output all startup options, use `mongrel_rails stop` to shut the server down.

For production use (better performance, using SSL etc) , a “proper server” (i.e. Apache or Lighttpd) is recommended.
Apache + CGI is VERY slow, a fallback option only.
Apache + FastCGI is possible, but seems to never work properly (random errors, escessive database connection,...)
Lighttpd + FastCGI has not been tried yet.
Current setup is Apache for SSL with mod_proxy redirecting to Mongrel, works well.

12 Bibliography

textreference	link
[ANON]	http://www.anon-online.de
[CPPUNIT]	http://cppunit.sourceforge.net
[JUNIT]	http://www.junit.org
[PRICHTLINIEN]	http://anon.inf.tu-dresden.de/develop/codingstyle_de.html
[UNITTESTS]	Johannes Link: Unit Tests mit Java, dpunkt.verlag, 1.Auflage 2002 http://www.dpunkt.de/utmj/
[DIPLOBAIER]	Tobias Baier: Fertigstellung und Inbetriebnahme eines Bezahlsystems für den AN.ON-Anonymisierungsdienst (Diplomarbeit)
[DIPLOSCHRAML]	Elmar Schraml: Technische und organisatorische Fertisgstellung eines Bezahlsystems für den Internet-Anonymisiserungsdienst AN.ON und Überführung in die Produktivphase (Diplomarbeit)