POCO Platform

Quick Start Guide

Version 0.2 (preliminary)

Purpose of This Document

This document guides developers interested in the Applied Informatics POCO Platform through the first steps working with the POCO Platform C++ libraries and tools.

The document is targeted at developers and development/technical managers wanting to get an overview of the functionality and features offered by the Applied Informatics POCO Platform. Familiarity with the C++ programming language is assumed.

Validity of This Document

This document covers release 2009.1 and later releases of the Applied Informatics POCO Platform.

Copyright, Trademarks, Disclaimer

Table of Contents

1 Welcome

Thank you for choosing the Applied Informatics POCO Platform for your project and welcome to the powerful C++ libraries and tools of the POCO Platform. This document will help you in getting a smooth ride while installing and setting up the POCO Platform software, and going the first steps with the C++ libraries and tools of the POCO Platform.

If at any point during the evaluation you have any questions or need support, please contact us via email at *support@appinf.com*.

2 An Overview of the POCO Platform

The POCO Platform consists of a number of C++ Libraries and tools (such as code generators and other utilities) providing readily available building blocks for C++ based applications targeting different operating system and hardware platforms.

2.1 High-Level Overview

Figure 1 gives a high-level overview of the areas of functionality covered by the POCO Platform. Please note that, depending on which packages you have licensed, not all features described in the following may be available to you.

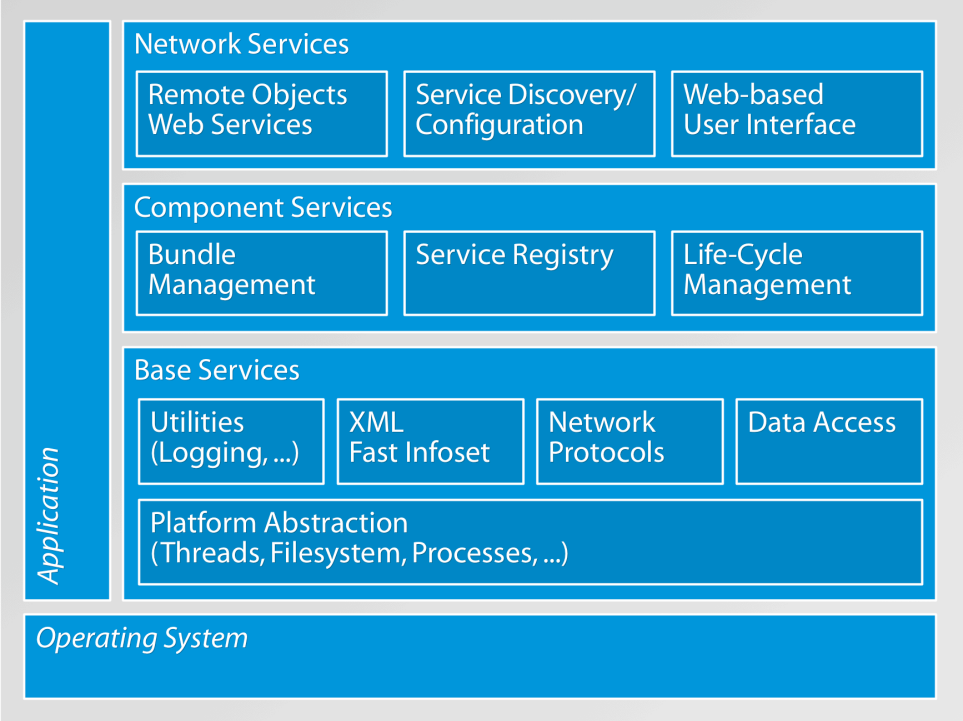

Figure 1: High-level overview of the POCO Platform

At the core of the POCO Platform, the so-called *Base Services* offer platform abstraction by providing uniform application programming interfaces (APIs) for working with the filesystem, threads, processes, shared libraries and other operating system resources. This allows it to create applications targeting different operating systems in a write-once, compile-and-run everywhere fashion. Also part of the Base Services are classes for working with XML, implementations of various network protocols (sockets, HTTP server and client, FTP, SMTP, POP3, SSL/TLS), libraries for accessing different SQL database management systems in a uniform fashion and various utility classes providing features such as logging or configuration data management.

On top of the Base Services, a *Component Services* layer provides a runtime system that allows the development and deployment of dynamic applications that can be extended using a powerful plug-in mechanism based on so-called bundles. Bundles combine executable code (shared libraries), configuration data and other resources in a single file. The dynamic nature of this system allows the creation of applications that can be extended and upgraded at runtime – even when deployed on a device in the field.

Finally, the *Network Services* layer offers advanced networking features such as remote objects and SOAP/WSDL web services, automatic network service discovery, remote configuration of network devices based on the NETCONF protocol, and support for rich browser-based user interfaces.

2.2 Libraries, Frameworks and Tools

Figure 2 shows the different C++ libraries that make up the POCO Platform.

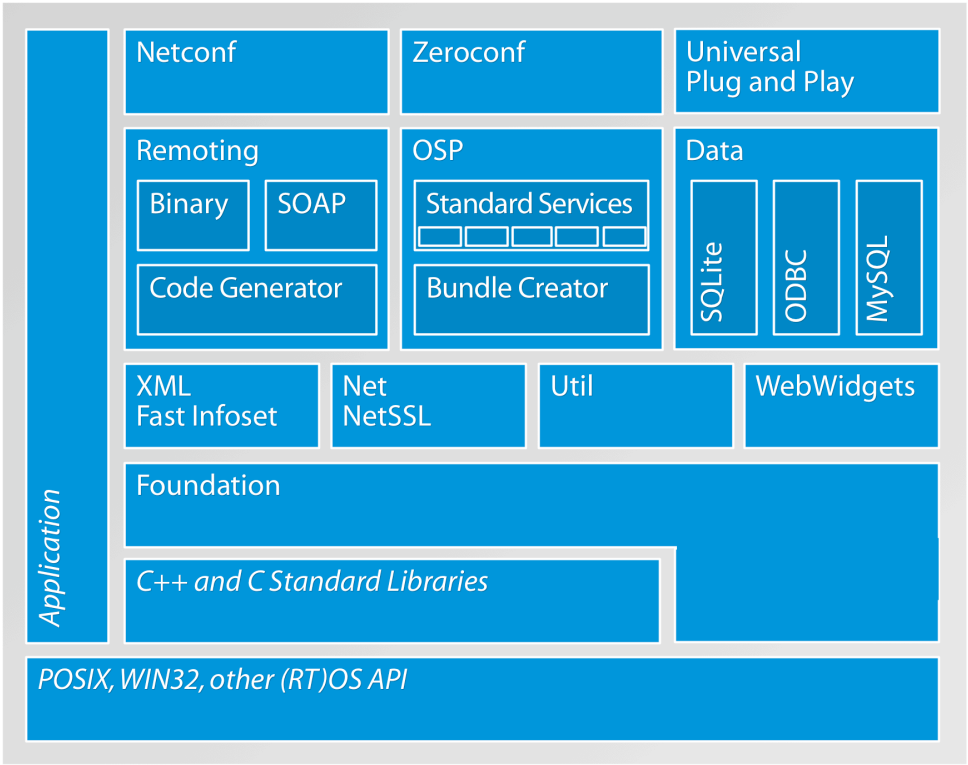

Figure 2: Libraries and Tools of the POCO Platform

The heart of the POCO Platform is the **Foundation** library. Among other useful classes and frameworks, the Foundation library provides numerous classes that shield the programmer from the underlying operating system application programming interface (API), and thus enable true cross-platform programming in a write once – compile anywhere sense. While embedded development is still often a tightrope walk between high-level abstractions – fostering maintainability and portability – and maximum code efficiency, especially maintainability and code portability are becoming increasingly important. After all, hardware platforms and operating systems

change more frequently than the software systems that run on them. The Foundation library includes a deliberately thin platform abstraction layer that is extremely well suited for embedded platforms, putting only very little overhead between your software and the target hardware.

Built upon the Foundation library are various libraries providing higher-level functions. The **Net** library provides implementations of various network protocols and servers like HTTP, FTP, SMTP and others. The **NetSSL** library adds SSL/TLS support to the Net library. The **XML** library provides an XML parser and writer, supporting the industry-standard SAX2 (Simple API for XML, Version 2[1]) and DOM (Document Object Model[2]) interface specifications. The **Fast Infoset** library provides a C++ implementation of the ITU-T Rec. X.891 and ISO/IEC 24824-1 standards for a binary representation of the XML information set — the contents of an XML document, based on the SAX2 and DOM interfaces from the XML library. The **Util** library offers classes for processing configuration files and command line arguments, as well as a framework for creating server applications, implemented as Unix daemons or Windows services. Finally, the **Data** library provides uniform access to various SQL databases. For embedded applications, the *SQLite* database engine is supported. For enterprise applications, ODBC and MySQL are directly supported.

The **Remoting** toolkit makes it easy to build distributed applications in C++. Remoting consists of a code generator (*RemoteGen*), a runtime library, as well as libraries implementing transport protocols. The Binary transport protocol provides an efficient proprietary transport protocol for communication between Remoting-based C++ applications. This makes the Binary transport well-suited for implementing high-level, object-based inter-process communication (IPC). The SoapLite transport protocol provides SOAP 1.1 and WSDL 1.1 support and allows to build web services in C++ that can be invoked from Java or .NET based applications.

The **Open Service Platform** (OSP) is a collection of C++ class libraries and tools for building dynamically extensible applications based on a powerful plug-in and services model. OSP enables applications that can be dynamically upgraded with new features and managed remotely, using web-based or console-based administration facilities. OSP implements a dynamic module system for C++ applications, similar in concept to the OSGi technology[3] for Java. At the core of OSP lies a powerful software component model based on the concept of bundles. A bundle is a deployable entity, consisting of both executable code and the necessary configuration, data and resource files required for running the code. Bundles extend the functionality of an application by providing features to other bundles, end-user functionality or web services. A central Service Registry allows bundles to discover the services provided by other bundles. Bundles can be added, updated, started, stopped or removed from an application without the need to terminate and restart the application. OSP consists of a runtime library, various bundles implementing standard services, as well as a tool for creating

[1] http://www.saxproject.org/
[2] http://www.w3.org/DOM/
[3] OSGi is a trademark or a registered trademark of the OSGi Alliance in the United States, other countries, or both. See http://www.osgi.org for more information on the OSGi Service Platform.

bundles. Also included are bundles implementing both a web-based and a console-based administration interface.

The POCO Platform Libraries are entirely based on the C++ standard library, including the Standard Template Library (STL). Therefore, unlike many other libraries, the POCO Platform does not bring in its own string, collection and stream classes, which eases their integration into your own projects.

The POCO Platform consistently uses C++ exceptions for error handling, thus making life easier for developers. Without exceptions, error handling is often done insufficiently (for example, function return values denoting success or failure are often ignored), leading to hard-to-track-down bugs. With modern C++ compilers, the runtime overhead caused by exception handling is minimal, and the advantages of exception handling far outweigh its costs. Consequent use of exceptions leads to more stable and better maintainable code.

3 Setting Up The POCO Platform

The C++ libraries and tools of the POCO Platform are delivered in full source code. This means that you have to build the POCO Platform C++ libraries and tools before you can use them the first time.

3.1 Source Code Distribution Format

The source code for the POCO Platform is delivered in a ZIP file for Windows users and/or in a compressed TAR file (*.tar.gz* or *.tar.bz2*) for Unix/Linux users. Both files contain the same files, the only difference is that all text files in the ZIP files have line endings suitable for Windows (*CR-LF*), while the text files in the TAR file have line endings suitable for Unix/Linux (*LF* only).

All libraries and tools of the POCO Platform follow a common convention for the directory layout. This directory layout is shown in Figure 3.

```
build/                      the build system for Unix and utility scripts
    config/                 build configurations for various Unix platforms
    rules/                  common make rules for all platforms
    scripts/                build and utility scripts

bin/                        all executables (DLLs on Windows)

doc/                        additional documentation

lib/                        all libraries (import libraries on Windows)

CppUnit/                    project and make files for CppUnit
                            (the unit testing framework used by POCO)
    doc/                    additional documentation for CppUnit
    include/
        CppUnit/            header files for CppUnit
    src/                    implementation files (.cpp) for CppUnit
    WinTestRunner/          Windows GUI for CppUnit
        include/            header files for WinTestRunner
        src/                implementation files for WinTestRunner
        res/                resource files for WinTestRunner

Foundation/                 project and make files for Foundation library
    include/
        Poco/               header files for the Foundation library
    src/                    implementation files (.cpp) for Foundation lib
    testsuite/              project and make files for Foundation testsuite
              src/          implementation files for Foundation testsuite
              bin/          test suite executables
    samples/                sample applications for the Foundation library

XML/                        project and make/build files for the XML library
    include/
        Poco/
            XML/            header files for the XML library
    src/                    implementation files (.cpp) for the XML library
    testsuite/              project and make files for XML testsuite
              src/          implementation files for the XML testsuite
              bin/          test suite executables
    samples/                sample applications for the XML library
```

Figure 3: General directory layout for the POCO Platform source code distribution

3.2 External Dependencies

Some parts of the POCO Platform require third-party open source software being installed before they can be built. For example, the NetSSL library requires OpenSSL[4], ODBC support in the Data package requires ODBC header files and libraries and the MySQL support in the Data packages requires the MySQL client header files and libraries.

[4] http://www.openssl.org/

3.2.1 OpenSSL

Most Unix/Linux systems (including Mac OS X) already have OpenSSL preinstalled, or OpenSSL can be easily installed using the system's package management facility. For example, on Ubuntu (or other Debian-based Linux distributions) you can type

```
$ sudo apt-get install openssl libssl-dev
```

to install the necessary packages.

If your system does not have OpenSSL, please get it from http://www.openssl.org/ or another source. You do not have to build OpenSSL yourself – a binary distribution is fine.

The easiest way to install OpenSSL on Windows is to use a binary (prebuilt) release, for example the one from Shining Light Productions[5] that comes with a Windows installer. Depending on where you have installed the OpenSSL libraries, you might have to edit the build script (buildwin.cmd), or add the necessary paths to the INCLUDE and LIB environment variables, so that the OpenSSL header files and libraries can be found by the Visual C++ compiler and linker, respectively.

3.2.2 ODBC

The Data library requires ODBC support on your system if you want to build the ODBC connector (which is the default). On Windows platforms, ODBC should be readily available if you have the Windows SDK installed. On Unix/Linux platforms, you can use iODBC[6] (preinstalled on Mac OS X) or unixODBC[7]. On Linux, use your distribution's package management system to install the necessary libraries and header files. For example, on Ubuntu, type

```
sudo apt-get install libiodbc2 libiodbc2-dev
```

to install the iODBC library and header files.

The Data/ODBC and Data/MySQL Makefiles will search for the ODBC and MySQL headers and libraries in various places. Nevertheless, the Makefiles may not be able to find the headers and libraries. In this case, please edit the Makefile in Data/ODBC and/or Data/MySQL accordingly.

3.2.3 MySQL Client

The Data library requires the MySQL[8] client libraries and header files if you want to build the MySQL connector (which is the default). On Windows

[5] http://www.slproweb.com/products/Win32OpenSSL.html
[6] http://www.iodbc.org/
[7] http://www.unixodbc.org/
[8] http://dev.mysql.com/

platforms, use the MySQL client installer to install the necessary files. On Unix/Linux platforms, use the package management system of your choice to install the necessary files. Alternatively, you can of course build MySQL yourself from source.

3.2.4 Zeroconf (Avahi)

The Zeroconf library requires either the open source Zeroconf implementation from Apple, or, alternatively, Avahi (Linux only) to work.

For Windows, please download the Bonjour software and SDK for Windows (Bonjour for Windows 1.0.3, as well as Bonjour SDK for Windows 1.0.3, or a newer release, if one is available) from *http://developer.apple.com/networking/bonjour/download/*. Both the software and the SDK comes as a Windows installer, both of which you should run.

For Linux, please download the Bonjour Source Code v107.6 (or a later release) from the website given above. Follow the following instructions to build and install Bonjour on Linux.

Download *mDNSResponder-107.6.tar.gz*

```
[guenter@localhost ~]$ wget http://www.opensource.apple.com/darwinsource/
tarballs/other/mDNSResponder-107.6.tar.gz
...
[guenter@localhost ~]$ gunzip mDNSResponder-107.6.tar.gz
[guenter@localhost ~]$ tar –xf mDNSResponder-107.6.tar
[guenter@localhost ~]$ cd mDNSResponder-107.6
[guenter@localhost mDNSResponder-107.6]$ cd mDNSPosix
```

If you are using GCC 4.0 or newer, you have to modify the `Makefile` prior to compiling the software. Open the `Makefile` in the `mDNSPosix` directory in your favorite editor, and locate the following line (for 107.6, this is line 270):

```
LD = ld –shared
```

and change that line to

```
LD = gcc –shared
```

Then build the libraries and applications, and install them (as superuser):

```
[guenter@localhost mDNSResponder-107.6]$ make os=linux
[guenter@localhost mDNSResponder-107.6]$ sudo make os=linux install
```

Alternatively, if you are going to use the *Zeroconf_Avahi* library, install the development package for Avahi using your Linux distribution's package management tools.

3.3 Building on Windows

For use on Windows, using Microsoft Visual C++ (.NET 2003 or newer), the POCO Platform source code comes in a ZIP file. Unpack the contents of the ZIP file to a directory of your choice, but please make sure that the path to the POCO installation directory does not contain any whitespace characters. While building the Foundation library, the Microsoft Message Compiler is invoked, and this tool unfortunately does not support path names containing white space characters.

Microsoft Visual Studio 7.1 (2003), 8.0 (2005) or 9.0 (2008) is required to build the POCO Platform on Windows platforms. Solution and project files for all three versions are included.

Please make sure that the bin directory that will contain the POCO Platform dynamic link libraries and executables is in your systems executable search path (the PATH environment variable) before starting the build. For example, if you have extracted the POCO Platform sources to C:\POCO, make sure that the PATH environment variable contains C:\POCO\bin, otherwise certain projects will fail to build.

You can either build each project separately from within Visual Studio (*Build->Batch Build->Select All;Rebuild*) or build everything from the command line. To build from the command line, start the Visual Studio .NET 2003 (or 2005/2008) Command Prompt and change to the directory where you have extracted the POCO Platform sources. Then, simply start the *buildwin.cmd* script and pass as argument the version of visual studio (71, 80 or 90). You can customize what is being built by *buildwin.cmd* by passing appropriate command line arguments to it. Call *buildwin.cmd* without arguments to see what is available. Per default, buildwin.cmd will attempt to build all projects in all configurations (*shared_debug*, *shared_release*, *static_debug* and *static_release*). However, not all project files contain build configurations for *static_debug* and *static_release*. An error message will be displayed while attempting to build these configurations. However, everything else will be built.

To disable certain components (e.g., NetSSL_OpenSSL or Data/MySQL) from the build, edit the file named "components" and remove the respective lines.

Certain libraries, like NetSSL_OpenSSL, Crypto or Data/MySQL have dependencies to other libraries. Since the build script does not know where to find the necessary header files and import libraries, you have to either add the header file paths to the INCLUDE environment variable and the library path to the LIB environment variable, or you'll have to edit the *buildwin.cmd* script, where these environment variables can be set as well. Alternatively, you can also modify the global header file and library search paths in Visual Studio.

3.4 Building on Unix/Linux

For building on Unix platforms, the POCO C++ Libraries come with their own build system. The build system is based on GNU Make 3.80, with the help from a few shell scripts. If you do not have GNU Make 3.80 (or later) installed on your machine, you will need to install it using your Linux distribution's package management system, or download it from http://directory.fsf.org/devel/build/make.html, build and install it prior to building the POCO Platform.

You can check the version of GNU Make installed on your system with

```
$ gmake –version
```

or

```
$ make --version
```

Once you have GNU Make up and running, the rest is quite simple. To extract the sources and build all libraries, testsuites and samples, simply

```
$ gunzip poco-X.Y.tar.gz
$ tar -xf poco-X.Y.tar.gz
$ cd poco-X.Y.tar.gz
$ ./configure
$ make -s
```

See the configure script source for a list of possible options.

For starters, we recommend *--no-tests* and *--no-samples*, to reduce build times. On a multicore or multiprocessor machine, use parallel makes to speed up the build (*make -j4*).

You can omit certain components from the build. For example, you might want to omit Data/ODBC or Data/MySQL if you do not have the corresponding third-party libraries (iodbc or unixodbc, mysqlclient) installed on your system. To do this, use the *--omit* argument to *configure*:

```
$ ./configure --omit=Data/ODBC,Data/MySQL
```

3.5 Frequently Asked Questions

Q: Under linux what is the command to undo a "configure" session to get back to the initial state?

A: All that configure does is create a config.make file in the current directory. To undo a configure, you can just delete this file. Or simply run configure again, with different arguments.

Q: What does a "make install" actually do? I notice the built libraries were copied to /usr/local/lib and the includes under /usr/local/include. I also noticed the sample bundles went under bin. I take it I need to remove the ones I do not want. What else went into bin?

A: make install creates a directory hierarchy (*include*, *lib*, *bin*) under <prefix> (*/usr/local*, */opt/poco*, etc.) and copies header files, executables and libraries there. That's all. Tools like *bundle*, *cpspc* (PageCompiler), and *RemoteGen* are also supposed to go into *bin*.

However, I wouldn't use install, as it has some drawbacks:

- you lose the connection between libraries and source code, which makes debugging harder
- it makes it harder to rebuild POCO or parts of it in case you have to (e.g., due to local modifications or patches from us)

install was mainly meant to be a help for people creating packages for POCO using various packaging systems (e.g., RPM, dpkg, ports, etc.). I wouldn't use it to install POCO for development purposes.

Q: Where do you recommend the distribution/build directory be installed/moved for multiple developers to reference on the same server?

A: Actually, I would treat POCO as just another source component in your project's source tree. This also implies that you put the POCO sources into your configuration management system. Given proper configuration of your SCM system (line endings), you can then use the same source tree for Windows and Linux. A project organization that has worked well for us and some of our customers is the following:

```
/YourProject/
    poco/
        Foundation/
            include/
            src/
        XML/
        lib/
        ...
    YourLibrary1/
        include/
        src/
        ...
    YourLibrary2/
        include/
        src/
        ...
    YourApplication1/
        include/...
        src/...
    lib/
    bin/
    ...
```