The POCO C++ Libraries
for Device Software Development

White Paper

Version 2.0

Purpose of This Document

This document gives an overview of the Applied Informatics POCO C++ Libraries, a collection of class libraries and frameworks that greatly simplifies the development of network-centric and platform-independent applications in C++.

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 C++ Libraries. Familiarity with the C++ programming language is assumed.

Validity of This Document

This document covers release 1.3 and later releases of the Applied Informatics POCO C++ Libraries.

Copyright, Trademarks, Disclaimer

Table of Contents

1 Executive Summary

This document gives an overview of the Applied Informatics POCO C++ Libraries (formerly C++ Portable Components). The POCO C++ Libraries are a collection of open-source class libraries that simplify the development of network-centric, portable applications in C++. The libraries integrate perfectly with the C++ Standard Library and fill many of the functional gaps left open by it. Their modular and efficient design and implementation makes the POCO C++ Libraries extremely well suited for device software development, an area where the C++ programming language is becoming increasingly popular, due to its suitability for both low-level (device I/O, interrupt handlers, etc.) and high-level object-oriented development.

The POCO C++ Libraries help developers to focus on the unique core features of the product they are developing – the features that will ultimately sell the product. By reusing the tried and tested components provided by the POCO C++ Libraries, developers do not need to waste valuable time and resources re-inventing the wheel.

The classes provided by the Applied Informatics POCO C++ Libraries provide support for multi-threading, streams, logging and error reporting, accessing the filesystem, shared libraries and class loading, configuration file and command line handling, security, network programming (TCP/IP sockets, HTTP, FTP, SMTP, SSL/TLS, etc.), XML parsing (SAX2 and DOM) and generation, as well as SQL database access.

2   Introduction

C++ is slowly but steadily replacing C as the programming language of choice for embedded or device software development. While C++ has long been considered prohibitively resource intensive for embedded devices, with today's available powerful embedded hardware this is certainly no longer the case. For example, today's smart phones have roughly the same computing power (in terms of processor performance and memory capacity) than engineering workstations from 12 years ago, and 32-bit RISC microprocessors are state-of-the-art for embedded platforms.

What makes C++ quite unique among programming languages is that it covers the full range from low-level programming (interrupt service routines, direct hardware access) to high-level programming (classes, generic programming) in one language, without the need to connect both ends via awkward artificial interfacing mechanisms. With the software for embedded devices becoming more and more sophisticated, the high level abstractions of C++ are certainly needed in addition to its suitability for low-level programming.

What is missing from C++, however, is a rich standard library comparable to the Java and .NET class libraries and frameworks. The C++ standard library is a first step in this direction, but it still lacks important features such as filesystem access and multithreading support. While there is a large number of both commercial and open source class libraries filling this gap available today, many of these libraries integrate very poorly with the standard library. Even more so, they do not play very well with each other. They often bring their own string and collection classes, thus complicating their integration in existing code bases. Furthermore, since they duplicate the functionality of the standard library, they significantly increase the code size of the applications that use them. This is an issue specifically for embedded applications, where memory footprint is an important factor.

The Applied Informatics POCO C++ Libraries (formerly known as the C++ Portable Components) have been designed to fill the gaps left open by the C++ standard library. Unlike other class libraries, the POCO C++ Libraries have been designed for extreme modularity. You only take those classes and frameworks that you need for your particular project from the library, so the stuff that you do not need will not unnecessarily bloat your binaries. To accomplish this goal, the POCO C++ Libraries are delivered in source code form. Furthermore, an easy-to-use configuration builder application (*PocoBuilder*) is included that allows you to easily create your custom version of the POCO C++ Libraries, containing only those features you need for your specific project.

The Applied Informatics POCO C++ Libraries help you and your developers to focus on your product's unique core features – the features that will ultimately sell your product. Developers do not need to waste valuable time and resources re-inventing the wheel.

3   Libraries and Frameworks

The POCO C++ Libraries consist of a number of tried and tested class libraries and frameworks that greatly simplify the development of portable, network-centric C++ applications. While the POCO C++ Libraries also support typical desktop and server platforms (Windows, Mac OS X, Linux, Solaris, HP-UX, Tru64), they are extremely well suited for embedded platforms, such as embedded Linux, QNX or VxWorks. The POCO C++ Libraries have been designed so that they allow you to take out only those components needed for your particular application – there is no need to always embed the overhead of the entire package to your application unless you really need it.

Figure 1 gives a coarse overview of the architecture of the POCO C++ Libraries. The heart of POCO 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. With hardware platforms being commodities these days, it is the software that contains your company's valuable intellectual property. The POCO C++ Libraries include 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.
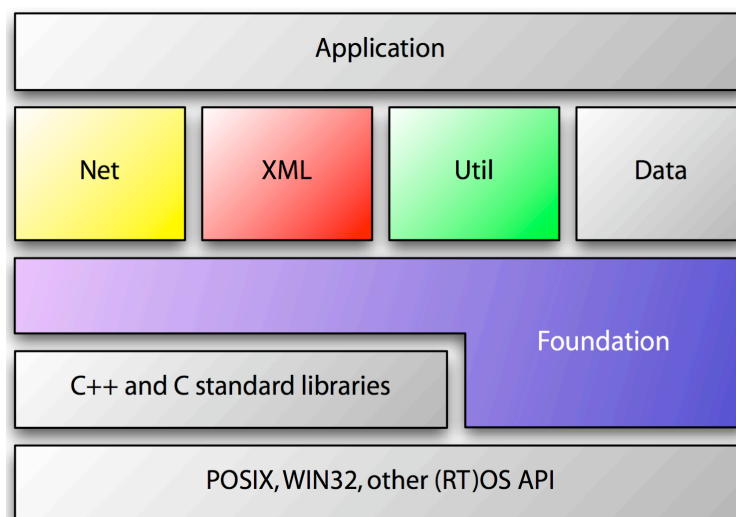

Figure 1   *POCO C++ Libraries Overview*

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 **XML**

library provides an XML parser and writer, supporting the industry-standard SAX2 (Simple API for XML, Version 2) and DOM (Document Object Model) interface specifications. 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.

The POCO C++ Libraries are entirely based on the C++ standard library, including the Standard Template Library (STL). Therefore, unlike many other libraries, the POCO C++ Libraries do not bring in their own string and collection classes, which eases their integration into your own projects. However, extensive use of STL classes can lead to increased code size, due to code duplication caused by template instantiations. To avoid code bloat, the POCO C++ Libraries only use a subset of the classes provided by the C++ standard library – strings, I/O streams and some of the container classes. Some C++ compilers for embedded platforms bring their own customized versions of the C++ standard library, containing only a subset of the functionality defined by the standard, with the goal of reducing the memory footprint of the library. It is possible to use the POCO C++ Libraries with these restricted libraries (although restrictions might apply, depending on what is provided by the actual standard library implementation).

The POCO C++ Libraries consistently use 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.

3.1 The Foundation Library

The Foundation library is the heart of the POCO C++ Libraries. Figure 2 gives an overview of its components.
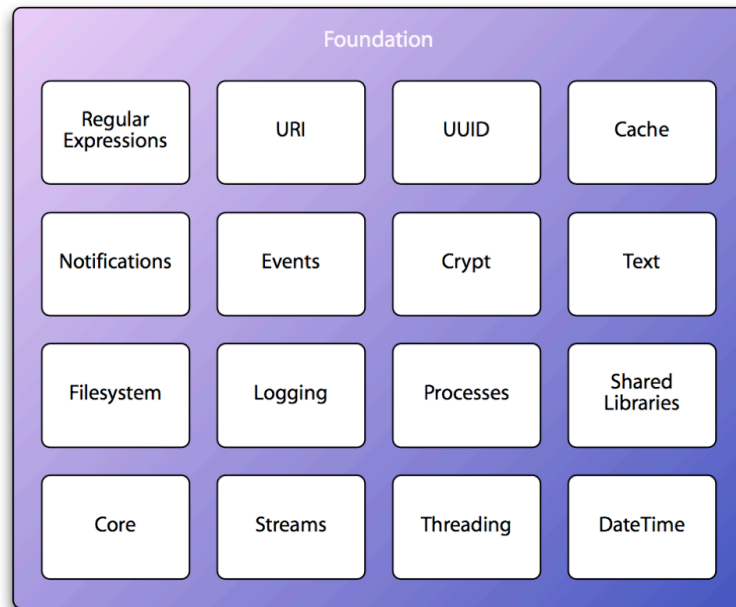
Figure 2   *The Foundation library*

The various components and the functionality they provide are outlined in the following paragraphs.

3.1.1 Core

The Core component, as its name implies, provides the most basic functionality of the Foundation library. It consists of five subcomponents that provide the following services:

3.1.1.1 Platform Abstraction

Included in the platform abstraction layer are platform-independent fixed-size data types (8, 16, 32 and 64 bit signed and unsigned integers), utility classes for converting between big-endian and little-endian integers, mechanisms for getting information about the operating system platform, and an interface to the system debugger.

3.1.1.2 Memory Management

This subcomponent includes a smart pointer and a shared pointer template class that implements reference counting-based garbage collection and a template class that simplifies the proper instantiation and destruction of singleton objects.

3.1.1.3 String Utilities

Handy functions for case-insensitive string comparison, case conversion, whitespace removal, formatting, character transformation and concatenation, as well as a string tokenizer class.

3.1.1.4 Error Handling

Classes, functions and macros for error handling, including various exception classes and useful macros for assertions and null pointer checks that enable you to write self-testing code. On some platforms, the library can be configured to automatically break into the debugger when an assertion is violated. The POCO C++ Libraries themselves make extensive use of assertions, allowing you to find subtle bugs in your code (for example, invalid method parameters or missing initializations) as soon as possible. Also, a special debug mode is supported where additional runtime checks are enabled.

3.1.1.5 Hash Tables

An efficient implementation of a linear hashtable provides the base for hash-based set and map implementations.

3.1.2 Streams

The Streams component provides various I/O stream and corresponding stream buffer classes, including base classes that make implementing your own custom stream and stream buffer classes much easier. All stream classes are based on and fully compatible with the standard I/O streams provided by the C++ Standard Library. Included are base classes for buffered and unbuffered streams, encoders that encode binary data in Base64 or hex-binary encoding (suitable for embedding binary data in XML documents or e-mail messages) and the corresponding decoders, a stream counting characters, lines and column numbers, as well as stream-based data compression/decompression based on the popular open source *zlib* data compression library.

3.1.3 Threading

Writing multithreaded code is hard. The platform independent classes for creating and managing threads, synchronizing threads (mutexes, conditions, events, reader/writer locks and semaphores), thread pool management and timers considerably simplify this task. A common programming error in multithreaded code is to forget to unlock a mutex that has been locked earlier in the code (for example, if a block is exited in the middle due to an exception or a "hidden" return statement). The scoped lock classes provided by the threading library ensure that mutexes are always properly unlocked when the program block where they have been locked is left. They therefore help to avoid a common source for deadlocks in multithreaded applications.

The multithreading classes also support the concept of *active objects*. An active object has methods that automatically execute in their own threads. Examples are *activities* – long running methods that perform background tasks and *active methods* – methods that execute asynchronously to their caller. Active objects relieve the programmer from thread management issues and help to reduce the complexity of multithreaded code.

3.1.4 DateTime

Timestamp and stopwatch classes for time keeping and measurement with up to microsecond (64-bit) accuracy. The actual time resolution depends on the underlying operating system platform. Classes for working with time spans and time zones and a class for working with calendar dates.

3.1.5 Filesystem

Easy to use classes for working with paths, files and directories in a platform independent manner. A directory iterator class that allows iteration over the files in a directory in C++ style. A path class that supports all commonly used notation styles for paths (Unix, Windows, OpenVMS). Classes that simplify working with temporary files. A Glob class for finding files using Unix-style patterns.

3.1.6 Logging

Logging is a valuable debugging aid especially for multithreaded applications. The Foundation library provides a powerful and extensible logging framework that is unobtrusive to use. The framework supports a variety of different log destinations (console, log files, Unix Syslog, Windows Event Log, remote logging, etc.). Based on a concept of cascaded log channels, filters and message formatters and configurable both programmatically and via configuration files (with help from the Util library).

3.1.7 Processes

Creation and management of processes, as well as synchronization among cooperating processes in the form of global mutexes and global events. Also supported are pipes for communicating with child processes.

3.1.8 Shared Libraries

Classes for working with shared libraries in a platform independent way. Including a template-based class loader that loads C++ classes dynamically at runtime from shared libraries, similarly to the class loader available in Java. This class loader frees a developer from many of the burdens when building plug-in architectures for applications.

3.1.9 Notifications

Facilities for type-safe sending and delivery of notification objects within a single thread or from one thread to another, also well suited for the implementation of notification mechanisms. A notification queue class for distributing tasks to worker threads, simplifying the implementation of multithreaded servers.

3.1.10 Events

Support for both synchronous and asynchronous event notifications. Similar in style to the event mechanism in C# and thus easy to use.

3.1.11 Crypt

Implementations of various message digest algorithms (MD2, MD4, MD5, SHA-1) that work on streams or buffers, an implementation of the popular HMAC message signature algorithm as well as random number and random data generation.

3.1.12 Text

Classes for working with text in various encodings (ASCII, ISO Latin-1, UTF-8, UTF-16, etc.), as well as for converting text between encodings that work with the standard C++ string and I/O stream classes. Support for working with UTF-8 encoded Unicode text using the standard library's string class.

3.1.13 Regular Expressions

Perl-compatible regular expressions, based on the popular open source *PCRE* library and wrapped in an intuitive to use class.

3.1.14 URI

Classes for working with Uniform Resource Identifiers (URIs), including an URIStreamOpener class.

3.1.15 UUID

Classes for cross-platform UUID (Universally Unique Identifier) manipulation and generation (time-based, name-based and random-based).

3.1.16 Cache

A framework for implementing caches with various expiration strategies.

3.2   The Net Library

The Net library contains a variety of classes for network programming. Basic TCP/IP sockets, as well as higher level network protocols are supported. Figure 3 gives an overview of the Net library.
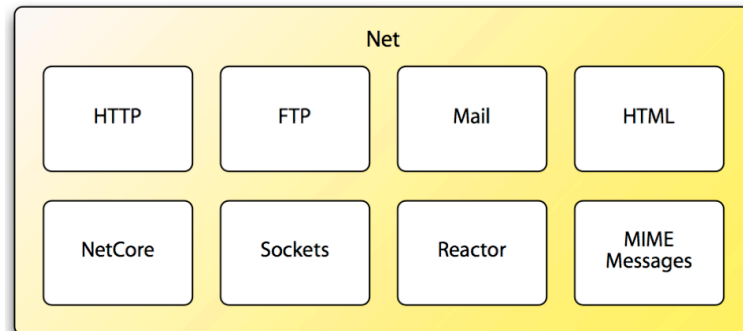

Figure 3   *The Net library*

Many embedded devices already have Ethernet network interfaces and support the TCP/IP protocol. Configuration, maintenance and monitoring functions for such devices can often be provided via a browser-based Web interface. Furthermore, many embedded devices will soon need to provide Web service interfaces, based on the Simple Object Access Protocol (SOAP) and HTTP. The classes provided by the Net library provide most of the building blocks required to implement these features.

3.2.1   NetCore

The NetCore classes provide the core functionality needed in every network application, such as handling of network addresses, and network name resolution (DNS).

3.2.2   Sockets

The socket classes provided by the POCO C++ Libraries take care of all the (often overlooked) subtleties that need to be dealt with when working with the Berkeley socket API, the industry's standard API for TCP/IP programming. Available are classes for TCP (stream) and UDP (datagram) sockets, TCP and UDP listeners and a multithreaded TCP server. Furthermore, support is provided for IPv4 and IPv6, as well as UDP broadcast and multicast. The TCP server, as well as network stream classes that work with sockets simplify the development of applications that provide a Telnet (RFC 854) text-based user interface.

3.2.3   Reactor

Support for the popular Reactor, Acceptor and Connector patterns useful for high-performance network applications.

3.2.4   MIME Messages

Classes for dealing with MIME (Multipart Internet Mail Extensions, RFC 2045 and RFC 2387) multi-part messages.

3.2.5   HTTP

A HTTP (Hyper Text Transfer Protocol, RFC 2616) client library, as well as an extensible, multithreaded HTTP server framework are. Both the server and the client support HTTP 1.0 and HTTP 1.1.

3.2.6   FTP

A FTP (File Transfer Protocol, RFC 959) client for file transfer applications.

3.2.7   Mail

Classes for sending e-mail messages via SMTP (Simple Mail Transfer Protocol, RFC 2821) servers with support for e-mail attachments, as well as classes for downloading e-mail messages from POP3 (Post Office Protocol Version 3, RFC 1939) servers.

3.2.8   HTML

Classes for working with HTML forms on both the client and the server side. Included is support for HTTP file uploads.

3.3   The NetSSL Library

The NetSSL library extends the Net library with support for secure, encrypted communication based on SSL (Secure Sockets Layer) and TLS (Transport Layer Security).

3.4   The XML Library

The eXtensible Markup Language (XML) library provides classes and frameworks for reading and writing XML data. The XML parser is based on the well-known open source *Expat* parser. On top of that parser sits a thin and lightweight layer that implements a C++ adoption of the SAX2 (Simple API for XML, version 2) interface for event-based XML parsing. A DOM (Document Object Model) implementation is provided for document-based XML manipulation. The DOM implementation has been optimized for low memory footprint. It provides a complete implementation of the DOM Level 2 Events specification, making it suitable for interactive applications. Figure 4 gives an overview of the library.
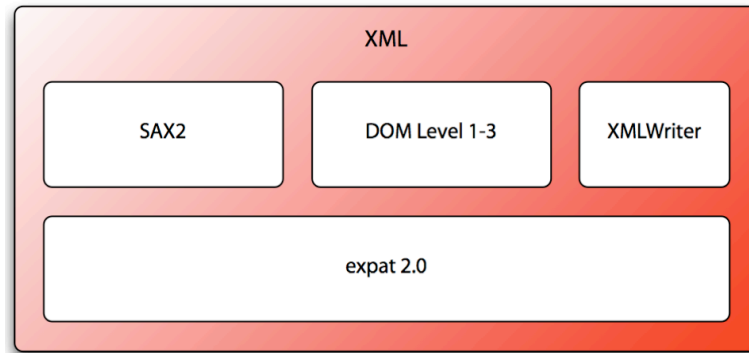
Figure 4   *The XML library*

While the XML library is normally used on top of the Foundation library, it can also be used (with some restrictions) on its own, therefore avoiding the small overhead incorporated by the Foundation library.

The XML library fully supports Unicode and it can be used in two ways. For one, it can be used with narrow (`char`-based) strings. In this case, all characters are UTF-8 encoded. Alternatively, the library can be used with wide (`wchar_t`-based) strings. UTF-16 encoding is used in the latter case. Mode selection is done at compile time, via preprocessor macros.

The XML parser is non-validating, which is a good choice for embedded platforms, as fully validating parsers have a considerable higher runtime and memory overhead.

3.4.1   SAX2

The POCO XML parser fully implements the SAX2 specification for event-based XML parsing. Also supported are most of the optional SAX2 Extensions (1.0 and 1.1). While the SAX2 interface is somewhat less convenient to use when compared with DOM, SAX2 is the interface of choice for efficient parsing of XML data and if the available memory is limited.

3.4.2   DOM

The DOM interface is suitable for in-memory storage and manipulation of XML data. Its implementation has been optimized for a lowest possible memory footprint (while not sacrificing performance) and the DOM Level 2 Events specification is fully supported, making the DOM implementation suitable for interactive applications.

The DOM implementation provided by the POCO XML library adheres to the following W3C specifications:

- Document Object Model Level 1
- Document Object Model Level 2 Core
- Document Object Model Level 2 Events
- Document Object Model Level 2 Traversal and Range

3.4.3   XMLWriter

The XML writer class simplifies the creation of XML data. Based on the SAX2 interfaces, it supports namespaces and formatted output in addition to ensuring that the written XML is well formed. Different text encodings are supported.

3.5   The Util Library

The POCO Utilities library makes available classes for working with configuration files and command line arguments, as well as a framework for building command line tools and server applications.

3.5.1   Configuration Files

The Util library supports unified handling of configuration files in various formats. The supported configuration file formats are Java-style property files, Windows-style initialization (.ini) files and XML files. Regardless of the file format, the interface for accessing the configuration data is always the same. Also available are mechanisms to combine configuration data from multiple sources. The framework can easily be extended with support for new configuration data sources, for example a database or the Windows registry.

3.5.2   Command Line Options

Also contained in the Util library are classes for platform-independent handling of command line options. The POCO C++ Libraries enable the creation of applications that work on different operating system platforms. Users on different platforms, however, expect to specify command line options in different, platform-specific ways. For example, users on Unix platforms expect to specify options using a dash (-) or double-dash (--) syntax. On the other hand, users on Windows or OpenVMS platforms expect to specify options using a slash (/) syntax. The classes for command line option handling automatically recognize options in the appropriate syntax for the respective platform, thus ensuring a consistent user experience on every supported platform. Furthermore, the classes take care of handling missing, duplicate, incompatible or ambiguously specified options appropriately, thus saving the developer tedious work. Automatic validation of command line argument values, based on regular expressions or numeric ranges is also supported.

3.5.3   Tools and Server Applications

Last but not least the Util library provides a framework for writing command line based tools or server applications. A server application is typically expected to run as a daemon process on a Unix platform, or as a service on a Windows platform. The application framework classes in the Util library implement the necessary glue code to achieve this transparently, allowing the same executable to run either from the command line, or as a daemon or service.

3.6   The Data Library

The Data library provides unified database access to various SQL-based databases. For accessing enterprise database systems like Oracle or SQL Server, a ODBC-based connector is available. For embedded scenarios, the *SQLite* embeddable SQL database engine is supported. Additional connectors (e.g., to MySQL) can be easily added on demand.

The Data library uses advanced C++ features to provide the illusion of embedding SQL statements directly in C++ code. Basic Object-Relational mapping is supported, so that C++ objects can be directly used in SL statements.

4   Portability and Platforms

The Applied Informatics POCO C++ Libraries are available on a variety of platforms. The platform-dependent and platform-independent parts of the library have been clearly separated, which makes porting the library to new platforms a relatively easy task. An extensive test suite ensures that everything works as expected on a new platform.

4.1   Supported Platforms

All major desktop and server platforms are readily supported, including Windows XP, Mac OS X, Linux, HP-UX, Tru64 and Solaris, in addition to embedded Linux, QNX Neutrino and other POSIX-compliant embedded operating systems.

4.2   Porting

Porting the POCO C++ Libraries to a new platform can be done by yourself, with optional support from Applied Informatics' developer team. Alternatively, Applied Informatics can be contracted to do the complete porting to a platform of your choice.

4.2.1   Target Platform Requirements

Target platforms for the POCO C++ Libraries must meet the following minimum requirements:

- a 32-bit CPU
- at least 8 MB of memory available for application code (RAM+ROM)
- optional FPU (or software-based floating point support)
- a TCP/IP stack for the Net library
- multithreading support for the threads and synchronization classes
- filesystem support is required for certain components
- an ANSI/ISO C++ conforming C++ compiler; see next section

4.2.2   C++ Compiler Requirements

To successfully build the POCO C++ Libraries, a C++ compiler must meet the following requirements:

- full ISO/IEC 14882 standards compliance (including templates, exception handling, runtime type identification)

- a conforming implementation of the C++ standard library, including the STL (minimum: strings, I/O streams and containers/iterators)

Following is an incomplete list of compilers (and platforms) known to work with the Applied Informatics POCO C++ Libraries:

- Microsoft Visual C++ 13.10 and 14.0 (Windows 2000/XP/CE)

- GCC 3.3-20030314 (Mac OS X 10.3)

- GCC 4.0 (Mac OS X 10.4)

- GCC 3.3.1 (Linux 2.4.21, QNX Neutrino 6.3)

- GCC 3.4.1 (Linux 2.4.21 and 2.6.3)

- Sun ONE Studio 11 C++[1] (Solaris 9)

- Compaq C++ 6.5-040 (HP Tru64 5.1b)

- HP ANSI C++ A.03.57 (HP-UX 11.11 PA-RISC)

- HP ANSI C++ A.06.12 (HP-UX 11.23 IA64)

---

[1] With restrictions – some of the class templates for tuples and hash tables in release 1.3 cannot be used due to compiler insufficiencies. Release 1.2 is fully supported.

5   Support, Licensing and Contact

The Foundation, XML, Net and Util libraries are available under the Boost open source license, which makes them free for both open source and commercial use.

For a complete description of the license regulations and support options please visit http://www.appinf.com/en/services.

Applied Informatics can be reached at one of the following addresses:

Applied Informatics Software Engineering GmbH

St. Peter 33
9184 St. Jakob im Rosental
Austria

Phone: +43 4253 32596
Fax: +43 4253 32096
E-Mail: info@appinf.com
Web: www.appinf.com