

POCO 라이브러리를 사용한 채팅 서버 개발

목차

- 네트워크 프로그래밍을 위한 기초 이론
- POCC 라이브러리 소개
- POCC.Net 소개
- 채팅 서버 만들기

네트워크 프로그래밍을 위한 기초 이론



TCP, UDP

- 네트워크 통신의 기본은 **TCP**
- UDP가 TCP 보다 빠르다는 말에 현혹되면 안됨
- TCP로 할 수 없을 때만 UDP

	TCP	UDP
연결 방식	연결형 프로토콜 연결 후 통신 1:1 통신 방식	비연결형 프로토콜 연결 없이 통신 1:1, 1:N, N:N 통신 방식
특징	<ul style="list-style-type: none"> - 데이터의 경계를 구분 안함 - 신뢰성 있는 데이터 전송 - 데이터의 전송 순서 보장 - 데이터의 수신 여부 확인 - 패킷을 관리할 필요 없음 - UDP보다 전송속도가 느림 	<ul style="list-style-type: none"> - 데이터의 경계를 구분함 - 신뢰성 없는 데이터 전송 - 데이터의 전송 순서가 바뀔 수 있음 - 데이터의 수신 여부를 확인 안함 - 패킷을 관리해야함 - TCP보다 전송속도가 빠름
관련 클래스	.Socket .ServerSocket	.DatagramSocket .DatagramPacket .MulticastSocket

출처: <https://shjz.tistory.com/98>

- [\[Network\] TCP/UDP](#)
- [TCP, UDP의 공통점 차이점, 특징](#)
- [TCP와 UDP \[동작원리/헤더/차이점\]](#)

TCP



UDP



UDP는 FPS 게임처럼 초당 패킷 전송이 많고, 최대한 빠르게 통신해야 하는 경우 사용한다



MTU

MTU는 한번의 데이터 전송에서 전송 가능한 **[IP 데이터 그램 최대 값]** 이다.

예를 들면 Ethernet 타입의 LAN 환경에서는 Ethernet 프레임워크가 최대 1518 byte 이므로 Ethernet 헤더(14byte)와 FCS(4byte)를 제외한 1500byte가 MTU 사이즈가 된다

Ethernet 타입 LAN에서의 MTU 계산				
Ethernet frame	-	Ethernet header & FCS	=	MTU
1518		18		1500

MSS

MSS는 TCP가 저장하는 유저 데이터로 「수신 가능한 세그먼트 사이즈의 최대 값」이다.

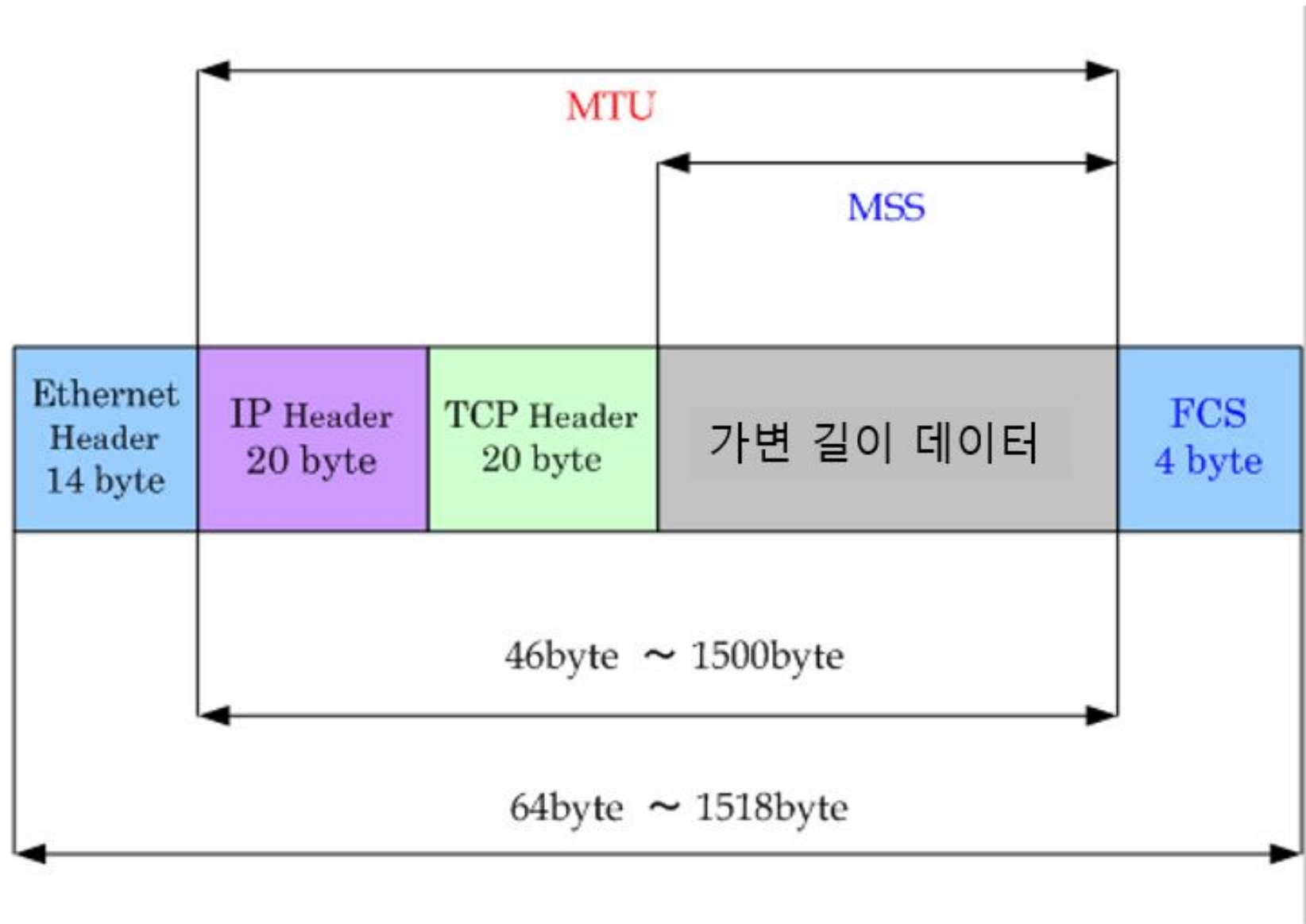
Ethernet 타입 LAN에서는 Ethernet 프레임워크가 최대 1518byte 이므로 Ethernet 헤더(14byte), FCS(4byte), TCP 헤더(20byte), IP 헤더(20byte)를 제외한 **1460byte가 MSS** 사이즈가 된다.

MSS는 MTU에서 TCP/IP 헤더(40byte)를 마이너스한 값으로 보통 「**MSS = MTU - 40**」가 된다.

Ethernet 타입 LAN에서의 MSS 계산

Ethernet frame	-	Ethernet header & FCS	-	IP header	-	TCP header	=	MSS
1518		18		20		20		1460

Ethernet 프레임



MTU 보다 큰 패킷을 보낸다면

IP Fragmentation and Reassembly

Example

- ❑ 4000 byte datagram
- ❑ MTU = 1500 bytes

1480 bytes in data field

offset =
 $1480/8$

length	ID	fragflag	offset
=4000	=x	=0	=0

One large datagram becomes several smaller datagrams

length	ID	fragflag	offset
=1500	=x	=1	=0

length	ID	fragflag	offset
=1500	=x	=1	=185

length	ID	fragflag	offset
=1040	=x	=0	=370

length	ID	fragflag	offset
=4000	=x	=0	=0

One large datagram becomes
several smaller datagrams

TCP가 분할해서 보낼 때의 문제는 이 중 하나라도 분실되면 모두 다시 보내야 한다

length	ID	fragflag	offset
=1500	=x	=1	=0

length	ID	fragflag	offset
=1500	=x	=1	=185

length	ID	fragflag	offset
=1040	=x	=0	=370

클라이언트 개발자의 말빨 을 위한 컴퓨터 네트워크 기초

네티션
배현직

1 of 40

KGC 2014: 클라이언트 개발자를 위한 컴퓨터
네트워크 기초 배현직

31,365 views

TCP/IP Review



IO Model

NHN NEXT

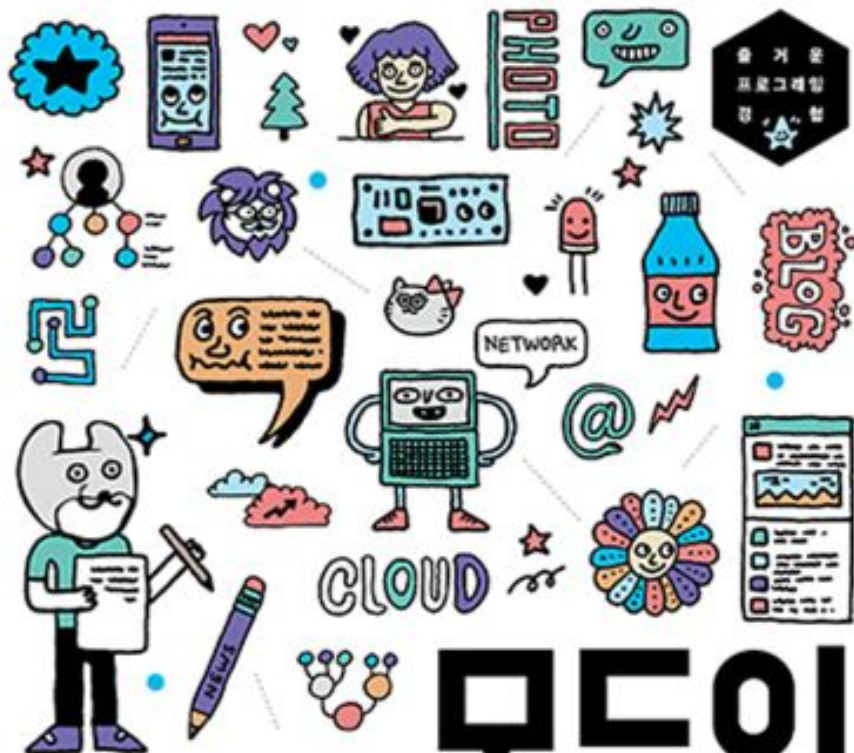
남현욱



<http://www.slideshare.net/namhyeonuk90/tcp-ip-io-model>



<https://www.slideshare.net/skytear7/tcpip-284152>
20



10일 만에
배우는
네트워크 기초

이준구씨 키스아 지음
이승룡 옮김

모두의 네트워크

이준구씨 키스아 지음 이준구씨 키스아 지음 이준구씨 키스아 지음 이준구씨 키스아 지음 이준구씨 키스아 지음

11월

POCO 라이브러리 소개

SIMPLIFY C++ DEVELOPMENT

The **POCO C++ Libraries** are powerful cross-platform C++ libraries for building network- and internet-based applications that run on desktop, server, mobile, IoT, and embedded systems.

[LEARN MORE >](#)[GET POCO ↻](#)

Latest Release: 1.10.1 [[Changelog](#)]

The POCO C++ Libraries are being used by C++ developers worldwide to build challenging and mission-critical applications.

Whether building automation systems, industrial automation, IoT platforms, air traffic management systems, enterprise IT application and infrastructure management, security and network analytics, automotive infotainment and telematics, financial or healthcare, C++ developers have been trusting the POCO C++ Libraries for 15+ years and deployed it in millions of devices.

<http://pocoproject.org/>

SIMPLIFY C++ DEVELOPMENT

GET THE POCO C++ LIBRARIES

TABLE OF CONTENTS

- [GitHub](#)
- [Conan](#)
- [Vcpkg](#)
- [Signed Packages](#)
- [Building POCO](#)
- [External Dependencies](#)

GITHUB

The official POCO C++ Libraries repository is on [GitHub](#). The master branch always reflects the latest release (1.10.1).

```
$ git clone -b master https://github.com/pocoproject/poco.git
```

On GitHub you can also find the [CHANGELOG](#).

CONAN

The POCO C++ Libraries are also [available](#) on [Bintray](#) via the [Conan C/C++ Package Manager](#).

```
$ conan install Poco/1.10.1@pocoproject/stable
```

THE POCO C++ LIBRARIES ARE POCOPROJECT.COM

EXTERNAL DEPENDENCIES

OPENSSL

Most Unix systems already have OpenSSL preinstalled. If your system does not have OpenSSL, please get it from the [OpenSSL website](#) or another source. You do not have to build OpenSSL yourself — a binary distribution is fine.

On Debian'ish Linux systems, you can install OpenSSL with:

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

Starting with El Capitan (10.11), macOS no longer includes OpenSSL. The recommended way to install OpenSSL on Mac OS X is via [Homebrew](#):

```
$ brew install openssl
```

The easiest way to install OpenSSL on Windows is to use a binary (prebuild) release, for example the installer from [Shining Light Productions](#).

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, or edit the Visual C++ projects if the installed OpenSSL libraries have different names than specified in the project file.

ODBC

The Data library requires ODBC for the ODBC connector.

On Windows platforms, ODBC should be readily available if you have the Windows SDK.

On Unix/Linux platforms, you can use [iODBC](#) or [unixODBC](#).

MYSQL CLIENT

명령 줄 빌드

- build
- cmake
- contrib
- CppUnit
- doc
- Encodings
- Foundation
- JSON
- Net
- patches
- Util
- XML
- build_vs140.cmd
- build_vs150.cmd
- build_vs160.cmd
- buildwin.cmd
- CHANGELOG
- CMakeLists.txt
- components
- configure
- CONTRIBUTORS
- DLLVersion.rc
- libversion
- LICENSE
- Makefile
- MANIFEST
- NEWS
- README
- VERSION

Visual Studio 2019

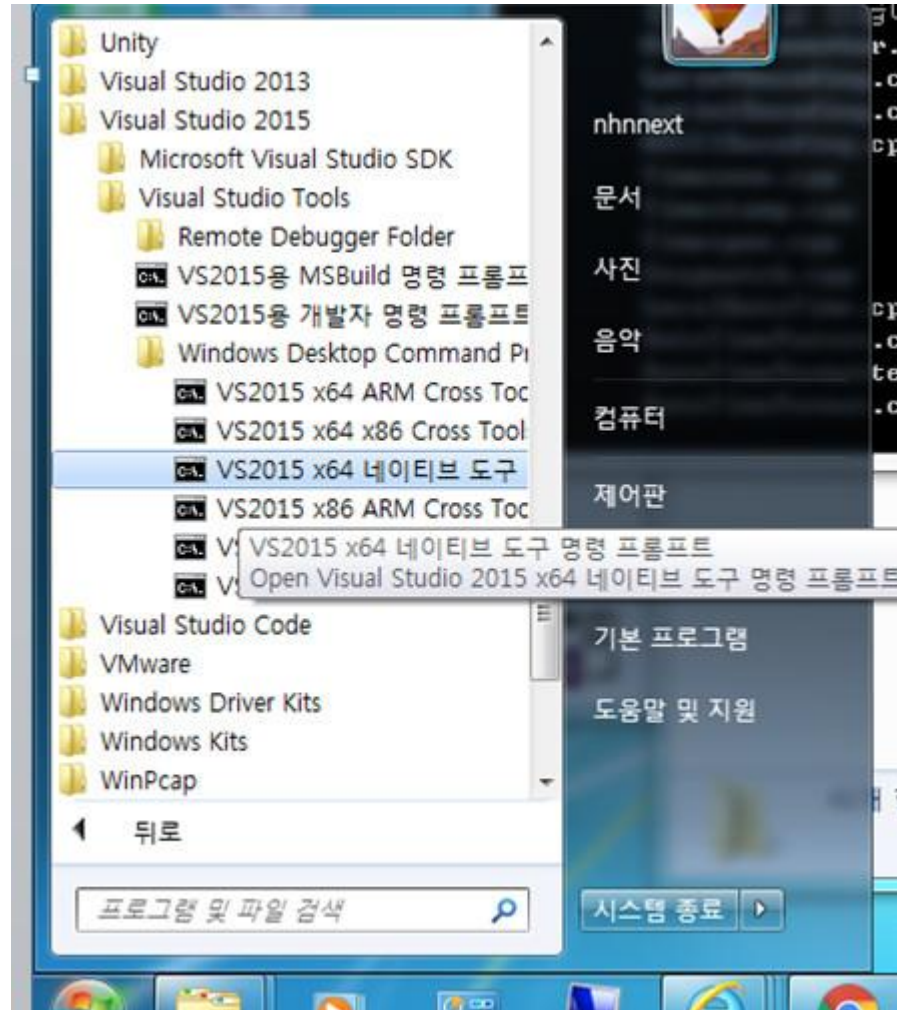


buildwin.cmd 160 rebuild static_md both x64

<http://stackoverflow.com/questions/30374503/building-poco-c-libraries-on-windows-from-command-line>

bin과 lib 디렉터리에 dll 파일과 lib 파일이 만들어져 있다.

혹시 커맨드라인 프로그램으로 64비트 버전을 빌드할 때 빌드할 수 없다는 오류가 발생한다면 Visual Studio의 Tools에서 제공하는 커맨드라인 프로그램을 사용하기 바란다.



```
선택 x64 Native Tools Command Prompt for VS 2019

----- Done building [CppUnit_vs160.vcxproj]

++++ Building [Foundation_vs160.vcxproj]

msbuild /clp:NoSummary /nologo /v:minimal /p:UseEnv=true /m /t:rebuild /p:Configuration=debug_static_md /p:Platform=x64
Foundation_vs160.vcxproj
D:\Program Files (x86)\Microsoft Visual Studio\2019\Enterprise\MSBuild\Microsoft\VC\v160\Microsoft.Cpp.WindowsSDK.targets(46,5): error MSB8036: Windows SDK 버전 8.1을(를) 찾을 수 없습니다. 필요한 버전의 Windows SDK를 설치하거나, 솔루션을
마우스 오른쪽 단추로 클릭하고 [솔루션 대상 변경]을 선택하거나 프로젝트 속성 페이지에서 SDK 버전을 변경하세요. [F:\C++\thirdparty\poco\Foundation\Foundation_vs160.vcxproj]

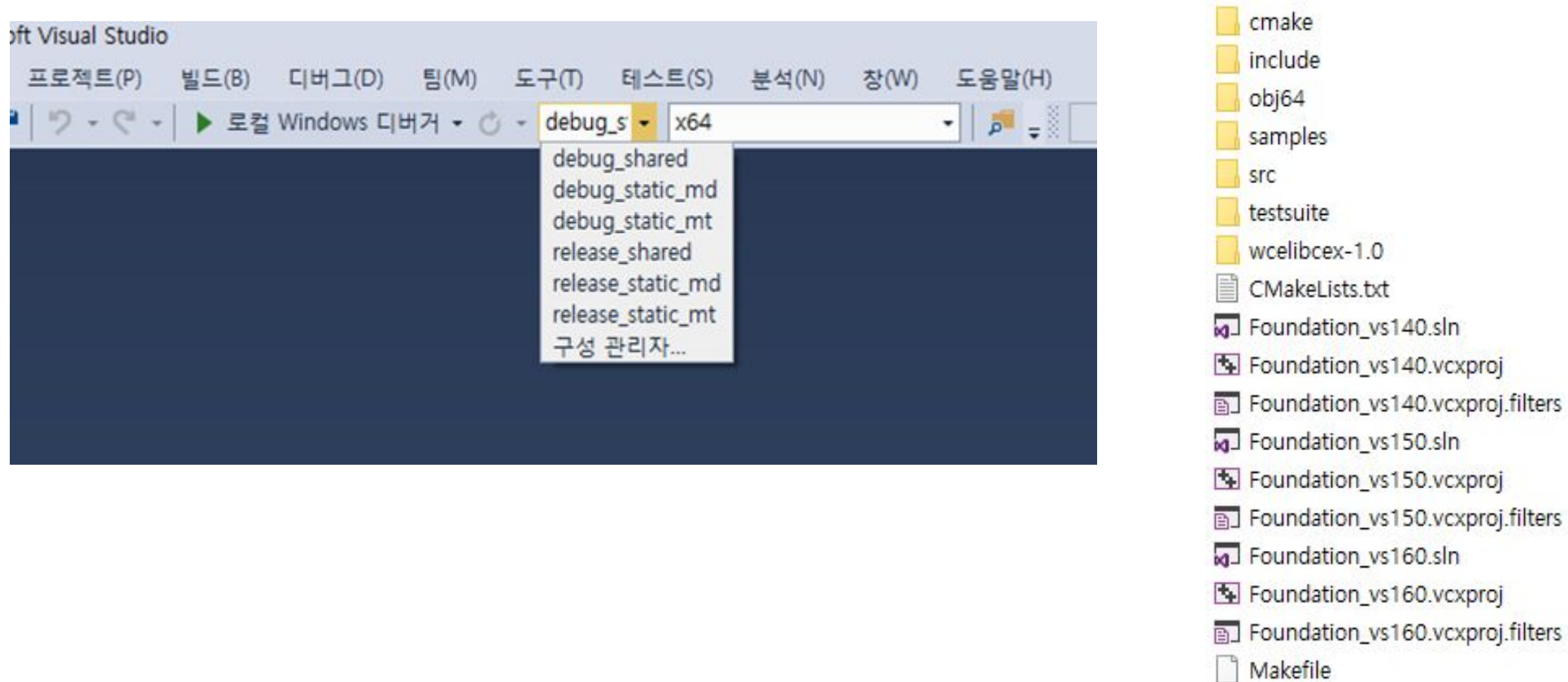
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXX  BUILD FAILED. EXITING.  XXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

F:\C++\thirdparty\poco>
```

2021.02.21 이런 빌드 에러가 나오는군요...

Visual Studio 솔루션 빌드

Visual Studio 최신 버전을 지원하지 않을 때는 각 라이브러리 디렉토리 (Foundation, Net, Xml, Util)에서 VS 솔루션 파일을 열어서 직접 빌드한다.



!제 솔루션 | 1 오류 | 0 경고 | 0 메시지 | 빌드 + IntelliSense

코드	설명
MSB803	Windows SDK 버전 8.1을(를) 찾을 수 없습니다. 필요한 버전의 Windows SDK를 설치하거나, 솔루션을 마우스 오른쪽 단추로 클릭하고 [솔루션 대상 변경]을

Foundation 속성 페이지


구성(C): 모든 구성 플랫폼(P): 활성(x64) 구성 관리자(O)...

구성 속성

- 일반
- 고급
- 디버깅
- VC++ 디렉터리
- C/C++
- 리소스
- XML 문서 생성기
- 찾아보기 정보
- 빌드 이벤트
- 사용자 지정 빌드 단계
- 사용자 지정 빌드 도구
- 코드 분석

일반 속성

출력 디렉터리	<다른 옵션>
중간 디렉터리	obj64\WFoundation\\$(Configuration)\W
대상 이름	<다른 옵션>
구성 형식	<다른 옵션>
Windows SDK 버전	8.1
플랫폼 도구 집합	Visual Studio 2019 (v142)
C++ 언어 표준	기본값(ISO C++ 14 표준)
C 언어 표준	기본값(레거시 MSVC)



Foundation 속성 페이지

구성(C): 모든 구성 플랫폼(P): 활성(x64) 구성 관리자(O)...

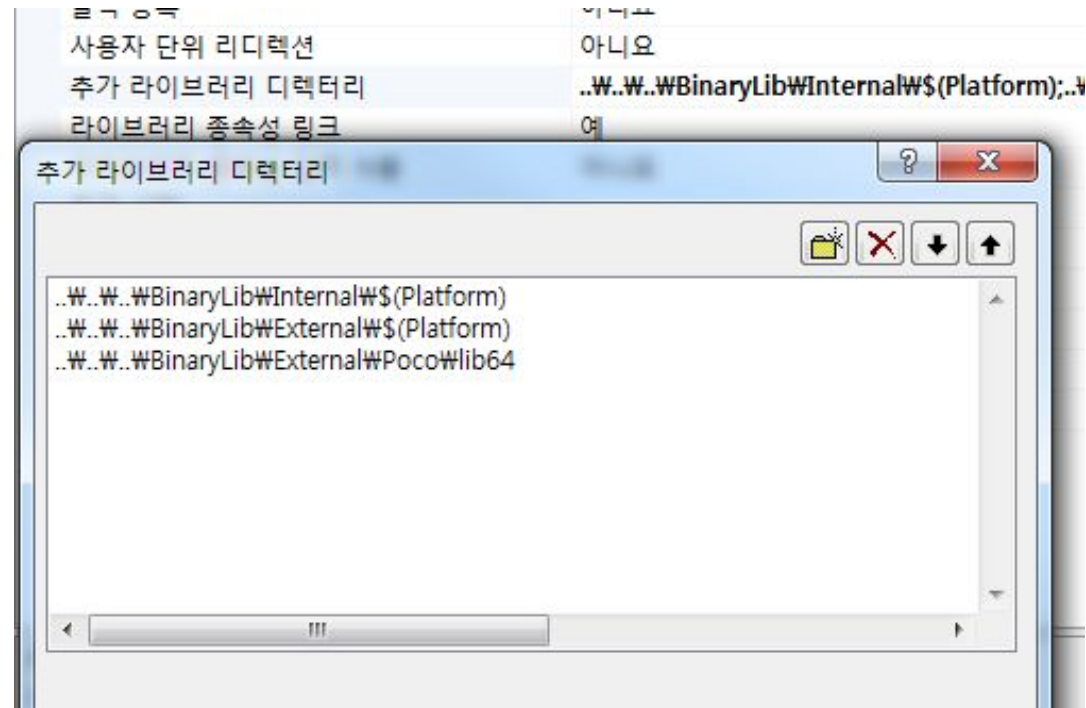
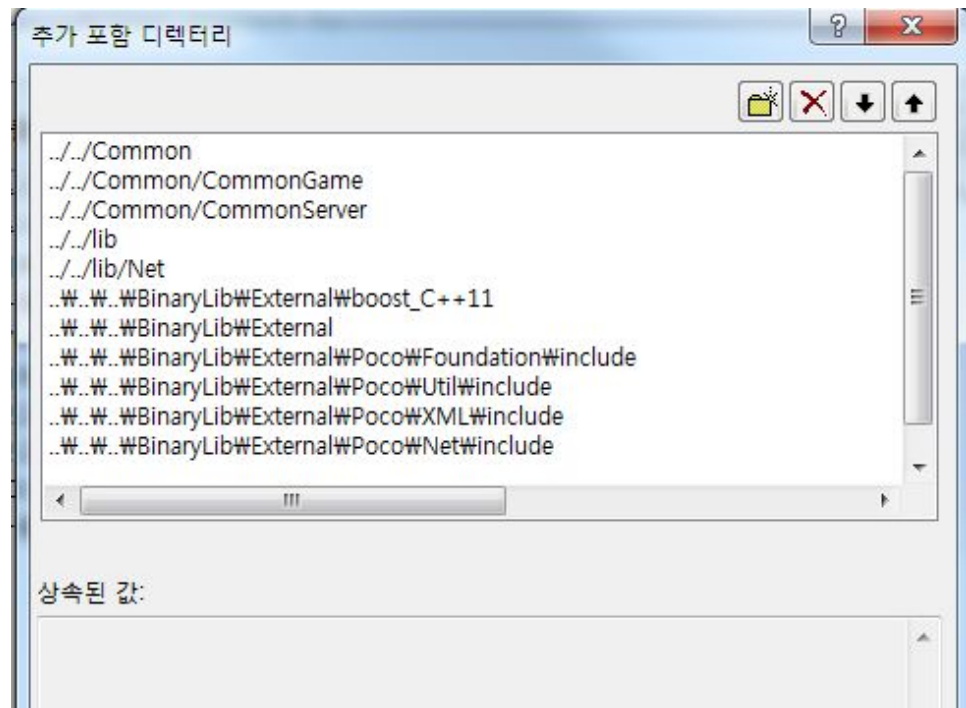
구성 속성

- 일반
- 고급
- 디버깅
- VC++ 디렉터리
- C/C++
- 리소스
- XML 문서 생성기
- 찾아보기 정보
- 빌드 이벤트
- 사용자 지정 빌드 단계
- 사용자 지정 빌드 도구
- 코드 분석

일반 속성

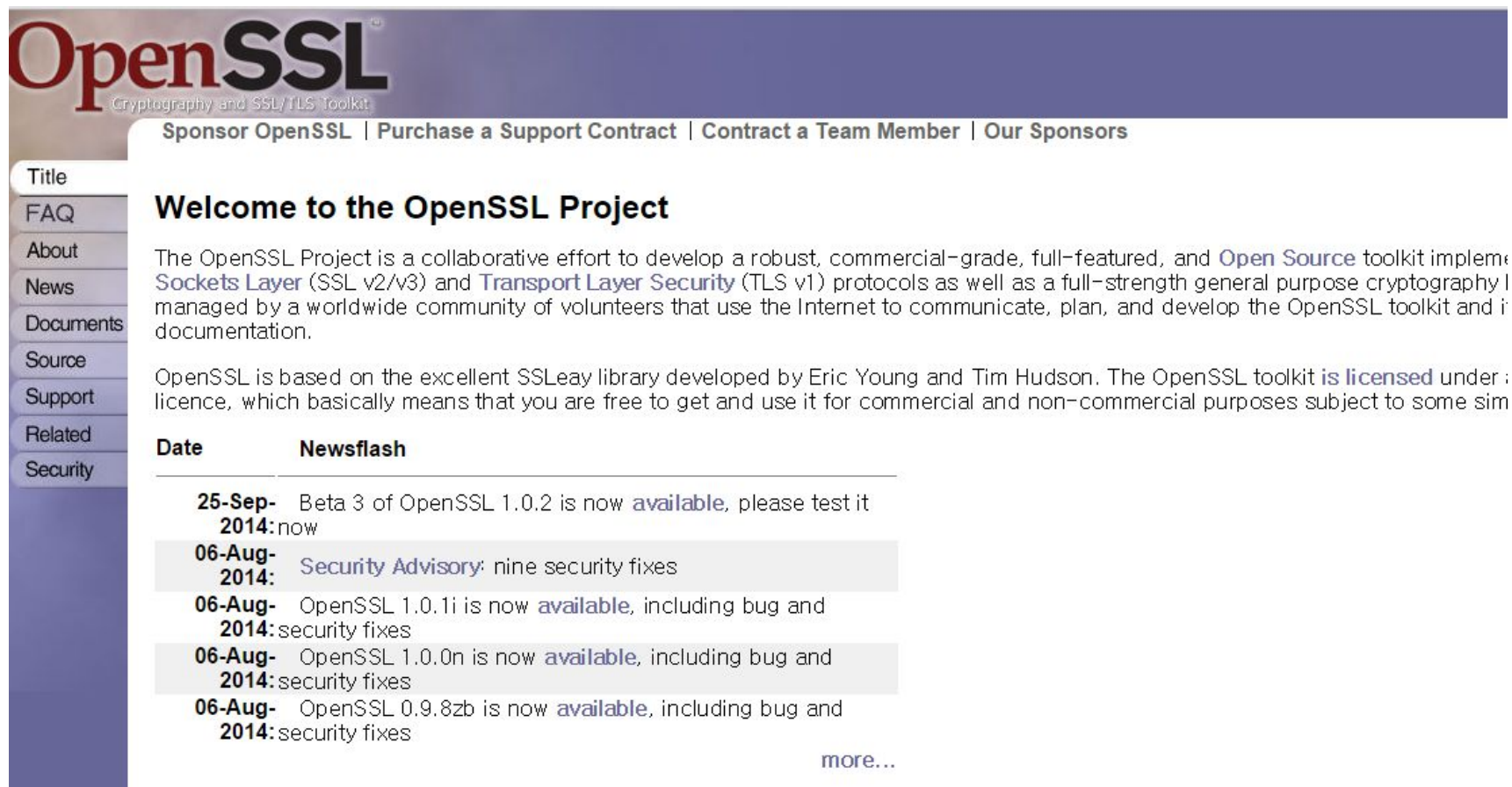
출력 디렉터리	<다른 옵션>
중간 디렉터리	obj64\WFoundation\\$(Configuration)\W
대상 이름	<다른 옵션>
구성 형식	<다른 옵션>
Windows SDK 버전	10.0(최근 설치된 버전)
플랫폼 도구 집합	Visual Studio 2019 (v142)
C++ 언어 표준	기본값(ISO C++ 14 표준)
C 언어 표준	기본값(레거시 MSVC)

Visual Studio 설정



[VC++ 10] 프로젝트의 디렉토리 설정
<http://jacking.tistory.com/748>

OpenSSL



The screenshot shows the OpenSSL website homepage. At the top, the OpenSSL logo is displayed in a large, stylized font, with the tagline "Cryptography and SSL/TLS Toolkit" underneath. Below the logo, a navigation bar contains links: "Sponsor OpenSSL", "Purchase a Support Contract", "Contract a Team Member", and "Our Sponsors". On the left side, there is a vertical menu with links: "Title", "FAQ", "About", "News", "Documents", "Source", "Support", "Related", and "Security". The main content area features a "Welcome to the OpenSSL Project" section, followed by a paragraph describing the project as a collaborative effort to develop a robust, commercial-grade, full-featured, and Open Source toolkit implementing Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols. Below this, a "Newsflash" section lists recent updates, including the availability of Beta 3 of OpenSSL 1.0.2 and several security advisories from August 2014. A "more..." link is provided at the bottom of the newsflash section.

OpenSSL
Cryptography and SSL/TLS Toolkit

[Sponsor OpenSSL](#) | [Purchase a Support Contract](#) | [Contract a Team Member](#) | [Our Sponsors](#)

[Title](#)
[FAQ](#)
[About](#)
[News](#)
[Documents](#)
[Source](#)
[Support](#)
[Related](#)
[Security](#)

Welcome to the OpenSSL Project

The OpenSSL Project is a collaborative effort to develop a robust, commercial-grade, full-featured, and **Open Source** toolkit implementing **Sockets Layer** (SSL v2/v3) and **Transport Layer Security** (TLS v1) protocols as well as a full-strength general purpose cryptography library, managed by a worldwide community of volunteers that use the Internet to communicate, plan, and develop the OpenSSL toolkit and its documentation.

OpenSSL is based on the excellent SSLeay library developed by Eric Young and Tim Hudson. The OpenSSL toolkit is **licensed** under the OpenSSL licence, which basically means that you are free to get and use it for commercial and non-commercial purposes subject to some simple conditions.

Date	Newsflash
25-Sep-2014:	Beta 3 of OpenSSL 1.0.2 is now available , please test it now
06-Aug-2014:	Security Advisory: nine security fixes
06-Aug-2014:	OpenSSL 1.0.1i is now available , including bug and security fixes
06-Aug-2014:	OpenSSL 1.0.0n is now available , including bug and security fixes
06-Aug-2014:	OpenSSL 0.9.8zb is now available , including bug and security fixes

[more...](#)

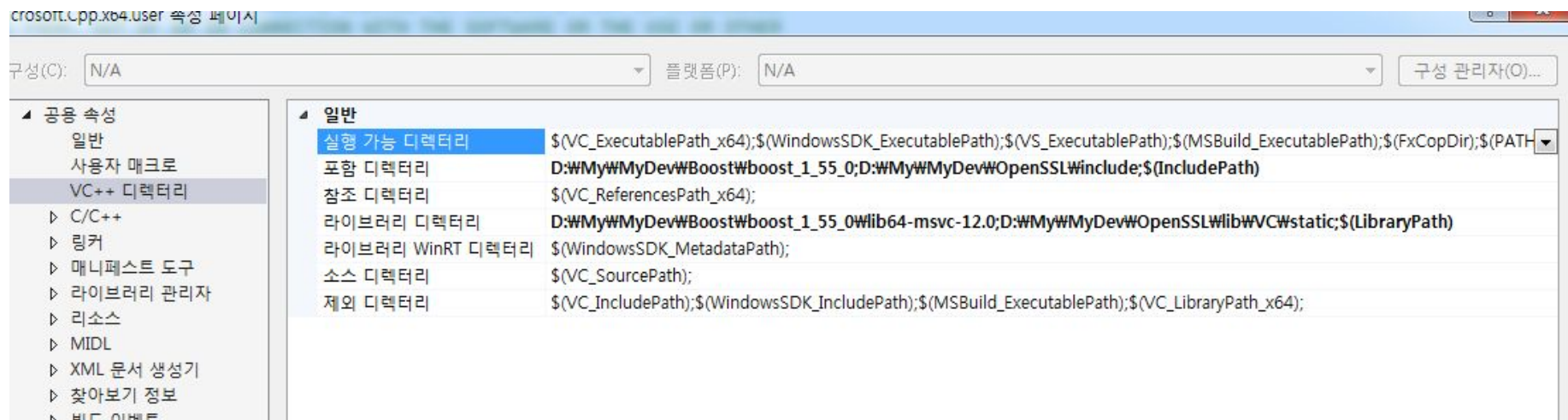
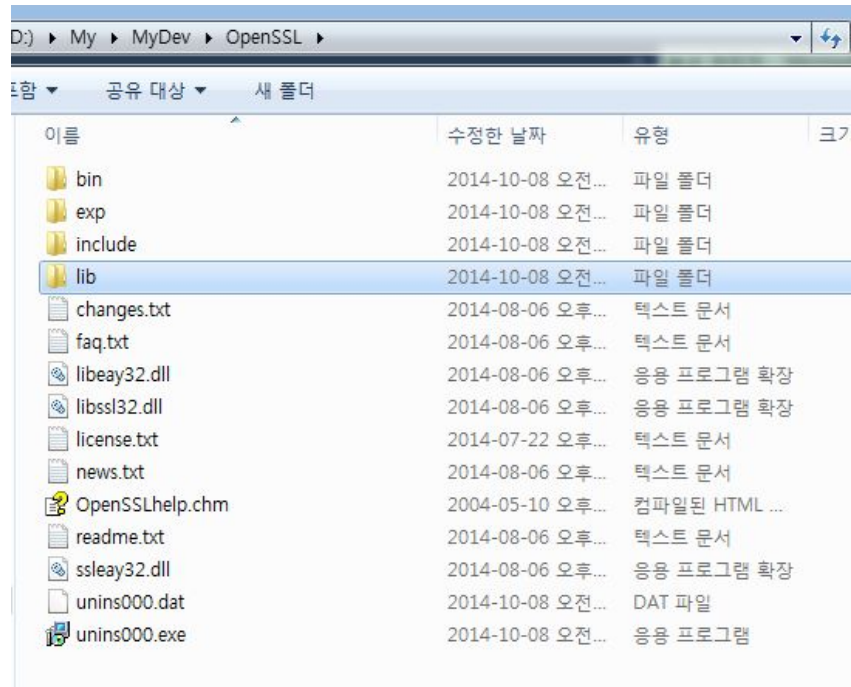
<http://www.openssl.org>

/

		OF THIS INSTALLER, THIS IS THE CORRECT VERSION TO INSTALL.
Win64 OpenSSL v1.0.0n	16MB Installer	Installs Win64 OpenSSL v1.0.0n (Only install this if you are a software developer needing 64-bit OpenSSL for Windows. Only installs on 64-bit versions of Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.
Win32 OpenSSL v0.9.8zb Light	1MB Installer	Installs the most commonly used essentials of Win32 OpenSSL v0.9.8zb (Recommended for most users by the creators of OpenSSL). Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.
Visual C++ 2008 Redistributables	1.7MB Installer	Having problems with error messages when trying to run OpenSSL? This will likely fix the problem. Only works with Windows 2000 and later. Although there is a "newer version" of this installer, this is the correct version to install.
Visual C++ 2008 Redistributables for Windows 9x/NT4	4.6MB ZIP file	Having problems with error messages when trying to run OpenSSL? This will likely fix the problem. Only use under Windows 95, 98, Me, and NT4. Install this, then install "OpenSSL Light" using the '/bin' option, and then run FixSSL_9xNT4.bat contained within the /bin directory.
Win32 OpenSSL v0.9.8zb	8MB Installer	Installs Win32 OpenSSL v0.9.8zb (NOT recommended for software developers by the creators of OpenSSL). Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.
Win64 OpenSSL v0.9.8zb Light	1MB Installer	Installs the most commonly used essentials of Win64 OpenSSL v0.9.8zb (Only install this if you need 64-bit OpenSSL for Windows. Only installs on 64-bit versions of Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.
Visual C++ 2008 Redistributables (x64)	1.7MB Installer	Having problems with error messages when trying to run 64-bit OpenSSL? This will likely fix the problem. Only works with Windows 2003 Server and later. Although there is a "newer version" of this installer, this is the correct version to install.
Win64 OpenSSL v0.9.8zb	8MB Installer	Installs Win64 OpenSSL v0.9.8zb (Only install this if you are a software developer needing 64-bit OpenSSL for Windows. Only installs on 64-bit versions of Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.

<http://slproweb.com/products/Win32OpenSSL.htm>

↓



테스트 해보기

[POCO] ini 파일 읽기

프로그래밍, GameDev

tag C++, ini, IniFileConfiguration, Poco, vc, vc++

ini 파일 형식으로 된 설정 파일을 읽을 때 사용한다.

```
[ZoneServer]
CHANNEL_COUNT = 10
CHANNEL_USER_COUNT = 200
```

헤더 파일

```
1 #include "Poco/Util/IniFileConfiguration.h"
```

사용 예

ini 파일 읽기에 실패하는 경우 예외가 발생하므로 예외 처리를 해야 한다(예외 종류에 따라서 실패 이유를 알 수 있다).

```
1 try
2 {
3     Poco::AutoPtr<Poco::Util::IniFileConfiguration> pZoneServerConf(new Poco::Util::IniFileConfiguration("te
4     m_ZoneConfig.nMaxChannelCount = pZoneServerConf->getInt( "ZoneServer.CHANNEL_COUNT" );
5     m_ZoneConfig.nMaxChannelUserCount = pZoneServerConf->getInt( "ZoneServer.CHANNEL_USER_COUNT" );
6 }
7 catch ( Poco::FileNotFoundException e ) // 파일이 없는 경우
8 {
9     SERVER_LOG("Init", LOG_LEVEL::error, "%s", e.displayText().c_str());
10    return false;
11 }
12 catch ( Poco::NotFoundException& e ) // 설정 키워드가 없는 경우
13 {
14     SERVER_LOG("Init", LOG_LEVEL::error, "%s", e.displayText());
15    return false;
16 }
```

<https://github.com/iacking75/CookBookPOCOCpp/blob/master/Foundation/iniFile.md>

Libraries And Linking Considerations

Automatic Linking of POCO C++ Libraries

The POCO C++ Libraries header files contain `#pragma comment` directives that enable automatic linking of the required POCO libraries to an application. For this to work, a few rules must be kept in mind.

- > The default is to link the POCO C++ Libraries dynamically (DLL).
- > To link statically, the code using the POCO C++ Libraries must be compiled with the preprocessor symbol `POCO_STATIC` defined.

Library Naming Conventions

The following naming conventions are used:

- > DLL import libraries are named `PocoLIB.lib` for the release build and `PocoLIBd.lib` for the debug build.
- > Static libraries built using the static multithreaded C/C++ runtime libraries are named `PocoLIBmt.lib` (release) and `PocoLIBmtd.lib` (debug).
- > Static libraries built using the DLL C/C++ runtime libraries are named `PocoLIBmd.lib` (release) and `PocoLIBmdd.lib` (debug).

32-bit libraries are placed in the `lib` directory. 64-bit libraries are placed in the `lib64` directory. DLLs are placed in `bin` (32-bit) or `bin64` (64-bit). 64-bit DLLs are named `PocoLIB64.dll` for release and `PocoLIB64d.dll` for debug, respectively.

<https://pocoproject.org/docs/99150-WindowsPlatformNotes.html>

```
#include <iostream>

#define POCO_STATIC
#include "Poco/Util/IniFileConfiguration.h"

int main()
{
    try
    {
        Poco::AutoPtr<Poco::Util::IniFileConfiguration> pZoneServerConf(new Poco::Util::IniFileConfiguration("ZoneServer.ini"));
        auto nMaxChannelCount = pZoneServerConf->getInt("ZoneServer.CHANNEL_COUNT");
        auto nMaxChannelUserCount = pZoneServerConf->getInt("ZoneServer.CHANNEL_USER_COUNT");

        std::cout << "Success" << std::endl;
    }
}
```


POCO C++ LIBRARIES DOCUMENTATION

TABLE OF CONTENTS

- [SDK Reference](#)
- [Slides](#)
- [Resources](#)

SDK REFERENCE

The SDK Reference contains User Guides and reference documentation for all classes.

[OPEN SDK REFERENCE](#) 

SLIDES

The following slides are a good starting point for beginners wanting to get familiar with basic POCO programming concepts and features.



POCO C++ Libraries

User Guides and Tutorials

- ▶ [Introduction](#)
- ▶ [POCO Data Library](#)
- ▶ [POCO PageCompiler](#)
- ▶ [POCO Zip Library](#)

Namespaces

- > [Poco](#)
- > [Poco::Crypto](#)
- > [Poco::Data](#)
- > [Poco::Data::Keywords](#)
- > [Poco::Data::MySQL](#)
- > [Poco::Data::ODBC](#)
- > [Poco::Data::PostgreSQL](#)
- > [Poco::Data::SQLite](#)
- > [Poco::Dynamic](#)
- > [Poco::Dynamic::Impl](#)
- > [Poco::Impl](#)
- > [Poco::JSON](#)
- > [Poco::JWT](#)
- > [Poco::MongoDB](#)
- > [Poco::Net](#)
- > [Poco::Net::Impl](#)
- > [Poco::Redis](#)
- > [Poco::Util](#)
- > [Poco::Util::Units](#)
- > [Poco::Util::Units::Constants](#)
- > [Poco::Util::Units::Internal](#)
- > [Poco::Util::Units::Units](#)
- > [Poco::Util::Units::Values](#)
- > [Poco::XML](#)
- > [Poco::Zip](#)
- > [std](#)

Packages

- ▶ [Crypto](#)
- ▶ [Data](#)
- ▶ [Data/MySQL](#)

User Guides And Tutorials

Introduction

- > [A Guided Tour Of The POCO C++ Libraries](#)
- > [Getting Started With The POCO C++ Libraries](#)
- > [How To Get Help](#)
- > [Acknowledgements](#)
- > [POCO C++ Libraries Release Notes](#)
- > [POCO C++ Libraries GNU Make Build System](#)
- > [POCO C++ Libraries Windows Platform Notes](#)
- > [POCO C++ Libraries Windows CE Platform Notes](#)
- > [POCO C++ Libraries VxWorks Platform Notes](#)
- > [POCO C++ Libraries Android Platform Notes](#)

POCO PageCompiler

- > [POCO C++ Server Page Compiler User Guide](#)

POCO Data Library

- > [POCO Data User Guide](#)
- > [POCO Data Connectors Developer Guide](#)
- > [POCO Data Release Notes](#)

POCO Zip Library

- > [POCO Zip User Guide](#)

API Reference by Namespace

Poco	Poco::Crypto	Poco::Data	Poco::Data::Keywords
Poco::Data::MySQL	Poco::Data::ODBC	Poco::Data::PostgreSQL	Poco::Data::SQLite
Poco::Dynamic	Poco::Dynamic::Impl	Poco::Impl	Poco::JSON
Poco::JWT	Poco::MongoDB	Poco::Net	Poco::Net::Impl
Poco::Redis	Poco::Util	Poco::Util::Units	Poco::Util::Units::Constants
Poco::Util::Units::Internal	Poco::Util::Units::Units	Poco::Util::Units::Values	Poco::XML
Poco::Zip	std		

POCO C++ Libraries 1.10.1-all
Copyright © 2020, Applied Informatics Software Engineering GmbH and Contributors

SLIDES

The following slides are a good starting point for beginners wanting to get familiar with basic POCO programming concepts and features.



Introduction and Overview

An Introduction to the POCO C++ Libraries.



Types and Byte Order

Types for fixed-size integers, byte order conversions and the Any/DynamicAny types.



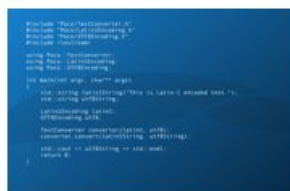
Error Handling and Debugging

Exceptions, error handling and debugging techniques.



Memory Management

Reference counting, shared pointers, buffer management, and more.



Strings, Text and Formatting

Working with strings, formatting, tokenizing, regular expressions and text encodings.



Platform and Environment

Getting information about the system you're running on.



Random Numbers and



Date and Time



The File System

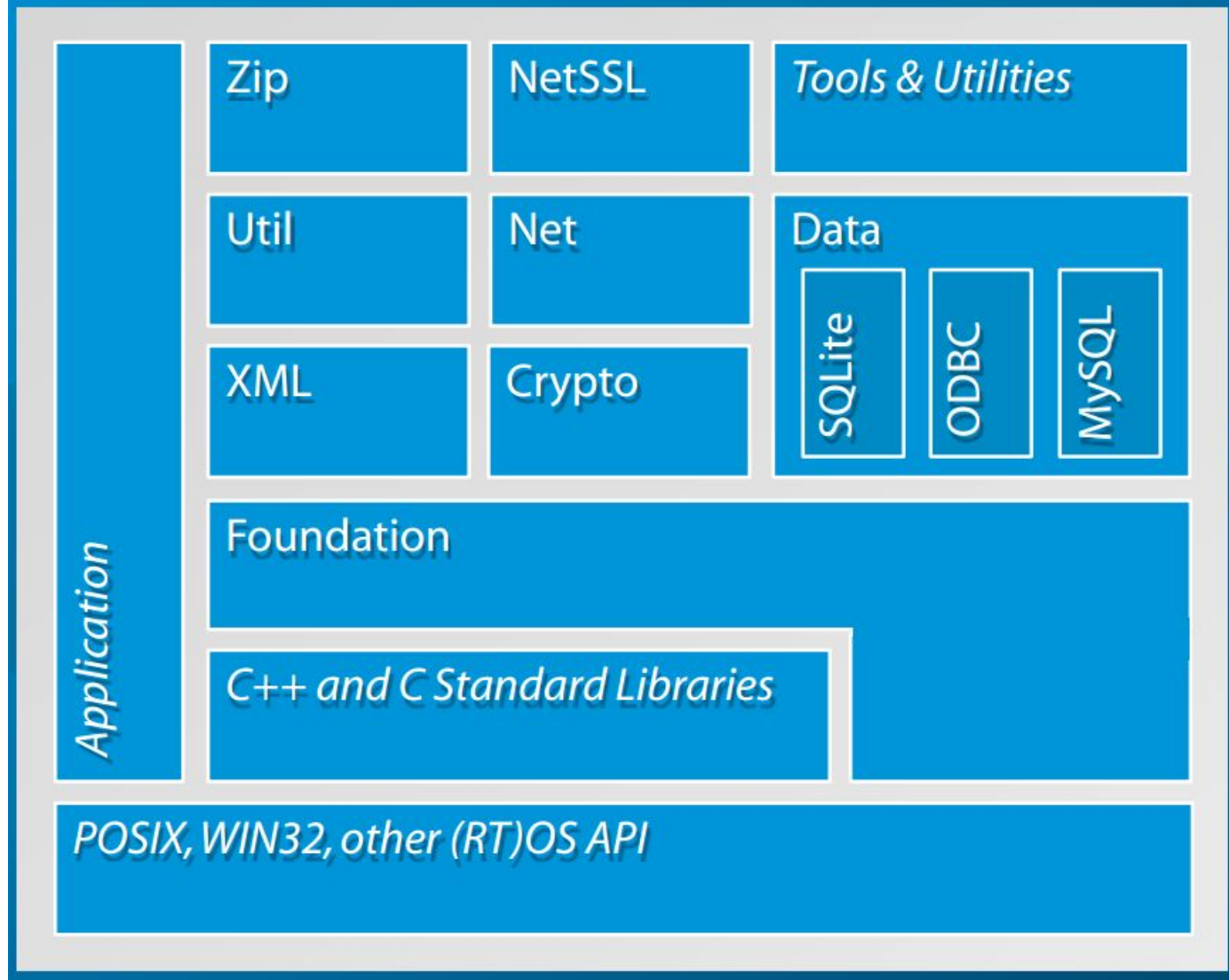
“좋은 라이브러리 없이 C++을 즐겁게
하는 것은 아주 어렵다.
그러나 좋은 라이브러리가 있다면
대부분은 간단하게 할 수 있다.”

Bjarne Stroustrup
(designer and original implementor of C++)

POCO 라이브러리는...

- Java의 클래스 라이브러리, .NET Framework, Cocoa(Apple)와 비슷한 C++ 라이브러리.
- “인터넷 시대”의 네트워크가 중심인 애플리케이션에 초점을 맞추고 있다.
- 최신 ANSI/ISO 표준 C++과 STL에 기초를 두고, 효율적으로 쓰여 있다.
- 이식성이 높고, 다양한 플랫폼에서 사용할 있다(iOS, Android도 지원).
- 오픈 소스이며 Boost 소프트웨어 라이선스를 채용하고 있다.
- 그래서 상용/비상용 모두 완전하게 공짜로 사용할 수 있다.

POCO C++ Libraries Overview



Features

- Any와 DynamicAny 클래스
- 캐시 프레임워크
- 암호 기술(암호 해시, OpenSSL에 토대를 둔 암호화)
- 날짜와 시간 클래스
- 이벤트(시그널/슬롯)와 통지 프레임워크
- 파일 전송을 위한 FTP 클라이언트
- 플랫폼에 의존하지 않는 path 조작과 파일 리스팅을 할 수 있는 파일 시스템 클래스
- HTML 조작
- HTTP 서버와 클라이언트(안전함), C++ 서버 페이지 컴파일러
- 로깅 프레임워크

Features (cont'd)

- 멀티스레딩: 스레드, 동기, 편리 기능(스레드 풀, 비동기 메시지, 지연 처리 등)
- 메일 수신용 POP3 클라이언트
- 플랫폼 추상화: 한번의 코딩으로 다양한 플랫폼에서 컴파일하고 동작 시킬 수 있다
- 프로세스간 통신
- Reactor 프레임워크
- PCRE를 기초한 정규 표현
- 메일 송신용 SMTP 클라이언트
- SQL 데이터베이스 조작 (SQLite, MySQL, ODBC)
- OpenSSL에 기초한 SSL/TLS를 지원

Features (cont'd)

- 공유 라이브러리와 클래스 로더
- 스마트 포인터와 메모리 관리
- 소켓과 RAW 소켓
- 스트림: Base64와 바이너리 엔코드/디코드, 압축 (zlib), 개행 코드 변환, 메모리 읽기/쓰기
- 문자열 처리
- TCP 서버 프레임워크(멀티스레드)
- 텍스트 엔코딩과 변환
- Tuples
- URI 처리

Features (cont'd)

- UTF-8과 Unicode 지원
- UUID 처리와 생성
- XML 파서(SAX2, DOM)와 XML 생성
- Zip 파일 조작

```

void run()
{
    int lastCount = 0;
    for(int i=0; i<kNumLoops; ++i)
    {
        { // scope for read lock
            Poco::ScopedReadRWLock lock(m_Lock);

            lastCount = m_Counter;
            for(int k=0; k<100; ++k)
            {
                if(m_Counter != lastCount)
                {
                    m_Ok = false;
                }
                Poco::Thread::yield();
            }
        }

        { // scope for write lock
            Poco::ScopedWriteRWLock lock(m_Lock);

            for(int k=0; k<100; ++k)
            {
                --m_Counter;
                Poco::Thread::yield();
            }
            for(int k=0; k<100; ++k)
            {
                ++m_Counter;
                Poco::Thread::yield();
            }
            ++m_Counter;
            if(m_Counter <= lastCount)
            {
                m_Ok = false;
            }
        }
    }
}

```

```
msg.Message(Poco::format("      nodeId: %s"
                        , Poco::Environment::nodeId()));

// Poco::Environment::nodeName
msg.Message(Poco::format("      nodeName: %s"
                        , Poco::Environment::nodeName()));

// Poco::Environment::osArchitecture
msg.Message(Poco::format("    osArchitecture: %s"
                        , Poco::Environment::osArchitecture()));

// Poco::Environment::osName
msg.Message(Poco::format("      osName: %s"
                        , Poco::Environment::osName()));

// Poco::Environment::osVersion
msg.Message(Poco::format("      osVersion: %s"
                        , Poco::Environment::osVersion()));

// Poco::Environment::processorCount
msg.Message(Poco::format("    processorCount: %u"
                        , Poco::Environment::processorCount()));
```

```

friend Poco::BinaryWriter& operator << (Poco::BinaryWriter& lhs, MyClass& rhs)
{
    lhs    << rhs.m_count
           << rhs.m_bool
           << rhs.m_char
           << rhs.m_uChar
           << rhs.m_short
           << rhs.m_uShort
           << rhs.m_int32
           << rhs.m_uint32
           << rhs.m_int64
           << rhs.m_uint64
           << rhs.m_double
           << rhs.m_float
           << rhs.m_str;

    return lhs;
}

friend Poco::BinaryReader& operator >> (Poco::BinaryReader& lhs, MyClass& rhs)
{
    lhs    >> rhs.m_count
           >> rhs.m_bool
           >> rhs.m_char
           >> rhs.m_uChar
           >> rhs.m_short
           >> rhs.m_uShort
           >> rhs.m_int32
           >> rhs.m_uint32
           >> rhs.m_int64
           >> rhs.m_uint64
           >> rhs.m_double
           >> rhs.m_float
           >> rhs.m_str;

    return lhs;
}

```

```

Poco::BinaryWriter writer(ostr, Poco::BinaryWriter::NETWORK_BYTE_ORDER);
writer.writeBOM();
writer << myClass;
myClassCopy[i] = myClass;

```

```

Poco::BinaryReader reader(istr);
reader.readBOM();
reader >> myClass;

```

```

msg.Message("--- Static methods ---");
msg.Message(Poco::format("  home()      (Home directory): %s", Poco::Path::home()));
msg.Message(Poco::format("  expand(\\\"%s\\\")      : %s", std::string("~/"), Poco:
msg.Message(Poco::format("  current() (Current directory): %s", Poco::Path::current()));
msg.Message(Poco::format("  temp()   (Temporary directory): %s", Poco::Path::temp()));
msg.Message(Poco::format("  separator()      : %c", Poco::Path::separator()));
msg.Message(Poco::format("  pathSeparator()  : %c", Poco::Path::pathSeparator(
msg.Message(Poco::format("  null()          (Null device): %s", Poco::Path::null()));

```

```

--- Static methods ---
home()      (Home directory): C:\Documents and Settings\setsu\
expand("~/")      : ~/
current() (Current directory): c:\Documents and Settings\setsu\デスクトップ\Poco\Web\Path\
temp()   (Temporary directory): C:\DOCUME~1\setsu\LOCALS~1\Temp\
separator()      : \
pathSeparator()  : ;
null()          (Null device): NUL:
listRoots(roots)
  root[0]: A:\
  root[1]: C:\
  root[2]: D:\
  root[3]: Z:\

```



```
Poco::Logger& fileLogger = Poco::Logger::create( "FileLogger",
                                                new Poco::FileChannel("test.log"),
                                                Poco::Message::PRIO_INFORMATION );

// 로컬 시간 지정
fileLogger.getChannel()->setProperty( Poco::FileChannel::PROP_TIMES, "local" );

// 파일 이름 날짜시간 문자 추가
fileLogger.getChannel()->setProperty( Poco::FileChannel::PROP_ARCHIVE, "timestamp" );

// 압축 사용
fileLogger.getChannel()->setProperty( Poco::FileChannel::PROP_COMPRESS, "true" );

// 일 단위로 로테이션
fileLogger.getChannel()->setProperty( Poco::FileChannel::PROP_ROTATION, "daily" );

// 5일간 보관
fileLogger.getChannel()->setProperty( Poco::FileChannel::PROP_PURGEAGE, "5days" );

fileLogger.error( "에러 발생 !!!" );
```

POCO와 목표와 미션

- POCO는 애플리케이션을 빌드할 플랫폼에서 간단하게 사용할 수 있으며 강력하다.
- POCO를 사용하면 높은 이식성을 가진 애플리케이션을 만들 수 있다.
- POCO는 임베디드 소프트웨어에서 기업 애플리케이션까지 사용할 수 있는 모듈이다.
- POCO는 불변이며 포괄적이며 사용하기 쉬운 인터페이스를 제공한다.
- POCO는 C++로 효율적으로 만들었다.
- POCO에서는 복잡한 것보다 간단함을 좋아한다.
- POCO에서는 일관된 디자인, 코딩 스타일, 문서를 지향한다.
- POCO에서는 소스 코드 질, 가독성, 쉬운 이해, 일관성을 가진 스타일과 테스트를 중요시 한다.
- POCO는 C++ 팬이 되돌아 오는 것을 목표로 한다.

기본 이념

- 코드 질, 스타일, 일관성, 가독성을 중시한다. 모든 코드는 코딩 스타일 가이드를 따른다.
- 테스트를 중시한다.
- 모든 문제를 해결하기 보다 실용적이며 우아한 디자인을 좋아한다.
- 튼튼한 기반 위에 만든다. 필요한 곳에 기존의 경험 있는 C 라이브러리를 사용한다 (예를 들면 expat, zlib, PCRE, SQLite) .

History

- 2004년 여름 : Günter Obiltschnig가 개발을 시작
- 2005년 2월: 처음 릴리즈를 SourceForge에 했었다 (Release 0.91 Sleepycat 라이선스)
- 2005년 5월: Aleksandar Fabijanic가 개발에 참가하였다.
- 2006년 1월: Release 1.0
- 2006년 3월: Release 1.1
- 2006년 7월: Boost 라이선스로 변경. POCO 커뮤니티 웹사이트 개설
- 2006년 8월: Release 1.2
- 2007년 5월: Release 1.3
- 2010년 7월: Release 1.3.7(안정판), 20인 정도의 개발자가 있으며, 수 백 프로젝트에서 사용되고 있다.

지원 플랫폼

- Microsoft Windows
- Linux
- Mac OS X
- HP-UX, Solaris, AIX(패치 필요)
- Embedded Linux (uClibc, glibc)
- iOS
- Android
- Windows Embedded CE
- QNX

사용 예

- 빌딩 자동 시스템 미들웨어나 디바이스
- 공장 자동화나 산업 설비
- 교통 정리 시스템
- 의료용 애플리케이션
- 데이터 계측, 수집, 테스트 시스템
- 가정용 전화 제품, 홈 오토메이션
- 스마트 그리드
- 항공 제어 시스템
- VoIP
- 표 판매기나 입장 게이트 시스템
- 상용 애플리케이션

POCO를 사용하고 있는 기업

- 454 Life Sciences (Roche)
새로운 고속 게놈 분석기에서 POCO 사용 중
- ACTIA Automotive
자동차 진단 시스템에서 POCO와 OSP를 사용 중
- Appcelerator Titanium
웹베이스 데스크탑 애플리케이션용 플랫폼에서 POCO를 사용 중
- CACE Technologies
네트워크 모니터/해석 시작품에서 POCO를 사용 중
- CodeLathe Tonido
웹 애플리케이션 플랫폼 Tonido 등에서 POCO를 사용 중
- Comact Optimisation
QNX에서 동작하는 자제 설비나, e 러닝 플랫폼에서 POCO를 사용 중
- HORIBA
자동 테스트 시스템에서 POCO를 사용 중

POCO를 사용하고 있는 기업 (cont'd)

- Novonics Corporation
미국 국방성이나 국토안전보호청에서 사용하고 있는 분산 시뮬레이션 라이브러리에서 POCO를 사용 중
- Nucor Steel
빔 압연기의 자동 애플리케이션에서 POCO를 사용 중
- Schneider Electric Buildings Business (TAC)
빌딩 자동화용 플랫폼(Embedded Linux 디바이스와 Windows/Linux 서버)에서 POCO 사용 중
- StreamUnlimited
텔레비전 셋탑박스에서 POCO를 사용 중
- Starticket
티켓/입장 관리 시스템(Xscale 상에서 OpenEmbedded)에서 POCO를 사용 중

POCO의 임베디드 분야에서의 확장성

- POCO는 Embedded Linux, Windows Embedded CE, QNX 등에서 동작하는 임베디드 시스템에 적합하다.
- POCO를 사용한 애플리케이션 (POCO 웹 서버를 사용) 는 75MHz ARM9, 8MB RAM, 4MB Flash의 Linux 시스템(uClibc)에서 동작한다.
- POCO 웹 서버를 사용한 전형적인 애플리케이션에서는 2MB의 정적 링크를 가지고, RAM 2-3MB를 사용한다.
- 전형적인 미들렌지 임베디드 플랫폼(32-64MB RAM, 16-64MB Flash, 180MHz ARM9)는 복잡한 애플리케이션(OSP 사용이나 리모트 앱)에도 풍부한 리소스를 제공한다.

POCO와 임베디드 -코드 크기

- POCO(SSL/Crypto를 포함)에 필요한 Flash 저장소는 4MB 미만(jffs2나 squashfs로 압축한 경우).
- 아래의 애플리케이션 RAM 오버헤드는 8MB 이다.

```
guenter@cis-digiel:~/ws/poco-1.3$ ls -l lib/Linux/armv5tejl/*.so.*
-rwxr-xr-x 1 guenter guenter 103752 2009-03-03 19:12 lib/Linux/armv5tejl/libPocoCrypto.so.6
-rwxr-xr-x 1 guenter guenter 1582720 2009-03-03 18:43 lib/Linux/armv5tejl/libPocoFoundation.so.6
-rwxr-xr-x 1 guenter guenter 907192 2009-03-03 18:47 lib/Linux/armv5tejl/libPocoNet.so.6
-rwxr-xr-x 1 guenter guenter 293960 2009-03-03 19:11 lib/Linux/armv5tejl/libPocoNetSSL.so.6
-rwxr-xr-x 1 guenter guenter 281048 2009-03-03 18:45 lib/Linux/armv5tejl/libPocoUtil.so.6
-rwxr-xr-x 1 guenter guenter 577588 2009-03-03 18:44 lib/Linux/armv5tejl/libPocoXML.so.6
-rwxr-xr-x 1 guenter guenter 353312 2009-03-03 18:54 lib/Linux/armv5tejl/libPocoZip.so.6
```

POCO에 의한 이익과 특징

- 포괄적으로 안전하고 성숙한 C++ 프레임워크는 대부분의 일에 도움이 되고, 제품을 빠르게 시장에 출시한다.
- 직감적으로 간단하게 학습할 수 있다. 일관성이 있고 알기 쉬운 인터페이스, 많은 샘플 코드와 좋은 문서.
- 네이티브 C++ 성능 (VM 오버헤드 등이 없는), 작은 메모리
- 플랫폼에 의존하지 않는다: 한번 코드를 만들면 어디에서라도 컴파일 해서 동작 시킬 수 있다.
- 대부분의 경우 특정 하드웨어에 접근이 필요하지 않은 애플리케이션에서는 개발에 사용하고 있는 환경에서도 테스트와 디버깅 할 수 있다.
- 애플리케이션은 다른 플랫폼에 간단하게 이식할 수 있다.

POCO 버전

- 현재(2021.02) 최신 버전은 1.10 이다.
- 안정 버전 이외에 개발 버전이 있음.

Namespaces

- > [Poco](#)
- > [Poco::Crypto](#)
- > [Poco::Data](#)
- > [Poco::Data::Keywords](#)
- > [Poco::Data::MySQL](#)
- > [Poco::Data::ODBC](#)
- > [Poco::Data::SQLite](#)
- > [Poco::Dynamic](#)
- > [Poco::Dynamic::Impl](#)
- > [Poco::Impl](#)
- > [Poco::JSON](#)
- > [Poco::MongoDB](#)
- > [Poco::Net](#)
- > [Poco::Util](#)
- > [Poco::Util::Units](#)
- > [Poco::Util::Units::Constants](#)
- > [Poco::Util::Units::Internal](#)
- > [Poco::Util::Units::Values](#)
- > [Poco::XML](#)
- > [Poco::Zip](#)
- > [std](#)

Namespaces

- > [Poco](#)
- > [Poco::Crypto](#)
- > [Poco::Data](#)
- > [Poco::Data::MySQL](#)
- > [Poco::Data::ODBC](#)
- > [Poco::Data::SQLite](#)
- > [Poco::Net](#)
- > [Poco::Util](#)
- > [Poco::Util::Units](#)
- > [Poco::Util::Units::Internal](#)
- > [Poco::XML](#)
- > [Poco::Zip](#)

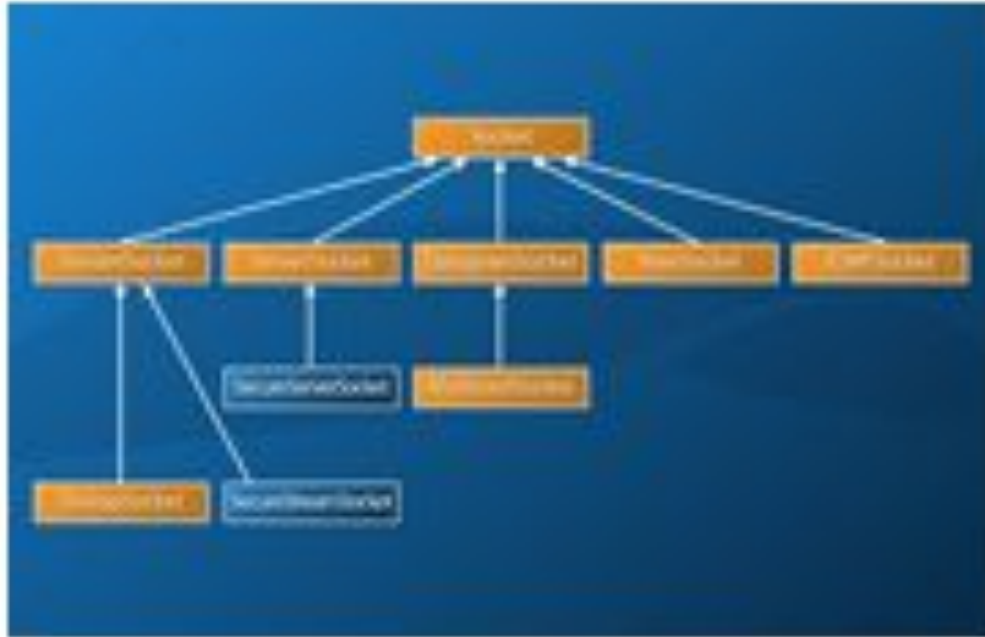
POCO.Net 소개

Boost.Asio vs POCO.NET

비동기 IO를 지원하지 않지만 Boost.Asio에서 비해서 사용하기 쉽고, 다양한 기능을 제공.

- FTP
- SMTP
- POP
- HttpClient, HttpServer
- TCPServer
- Reactor Framework

Linux에서는 Boost.Asio와 Poco.Net 둘 다 epoll 사용
Poco.Net은 Windows에서는 select 사용



Network Programming

Writing network and internet applications.

<http://pocoproject.org/slides/200-Network.pdf>

IP Addresses

- The `Poco::Net::IPAddress` class stores an IPv4 or IPv6 host address.
- An `IPAddress` can be parsed from a string, or formatted to a string. Both IPv4 style (d.d.d.d) and IPv6 style (x:x:x:x:x:x:x:x) notations are supported.
- You can test for certain properties of an IP address: `isWildcard()`, `isBroadcast()`, `isLoopback()`, `isMulticast()`, etc.
- `IPAddress` supports full value semantics, including all relational operators.
- See the reference documentation for details.

Socket Addresses

- A `Poco::Net::SocketAddress` combines an `IPAddress` with a **port number**, thus identifying the endpoint of an IP network connection.
- `SocketAddress` supports value semantics, but not comparison.
- A `SocketAddress` can be created from an `IPAddress` and a port number, a string containing an IP address and a port number, or a string containing both an IP address and a port number, separated by a colon.

```
DialogSocket ds;  
ds.connect(SocketAddress("localhost",  
9911));
```

```
explicit SocketAddress(  
    const std::string & hostAndPort  
);
```

```
SocketAddress(  
    const IPAddress & host,  
    Poco::UInt16 port  
);
```

```
SocketAddress(  
    const std::string & host,  
    Poco::UInt16 port  
);
```



```
192.168.1.10:80  
[::ffff:192.168.1.120]:2  
040  
www.appinf.com:8080
```

Name Resolution

- The `Poco::Net::DNS` class provides an interface to the Domain Name System, mapping domain names to IP addresses and vice versa.
- Address information for a host is returned in the form of a `Poco::Net::HostEntry` object.
- A `HostEntry` contains a host's primary name, a list of aliases, and a list of IP addresses.

```
const std::string hostname("www.google.com");
```

```
msg.Message(Poco::format("Poco::Net::DNS::hostByName(\"%s\")", hostname));  
ShowHostEntryItems(Poco::Net::DNS::hostByName(hostname), msg);
```

```
void ShowHostEntryItems(const Poco::Net::HostEntry& he, const ScopedLogMessage& msg)  
{  
    msg.Message(Poco::format("    name: %s", he.name()));  
    for (Poco::Net::HostEntry::AddressList::const_iterator itr = he.addresses().begin(); itr != he.addresses().end(); ++itr)  
    {  
        msg.Message(Poco::format("    address#%d: %s",  
            static_cast<int>(itr - he.addresses().begin()),  
            itr->toString()));  
    }  
    for (Poco::Net::HostEntry::AliasList::const_iterator itr = he.aliasnames().begin(); itr != he.aliasnames().end(); ++itr)  
    {  
        msg.Message(Poco::format("    alias#%d: %s",  
            static_cast<int>(itr - he.aliasnames().begin()),  
            *itr));  
    }  
}
```


1. Poco::Net::DNS::hostName() 으로 자신의 호스트 명을 얻는다

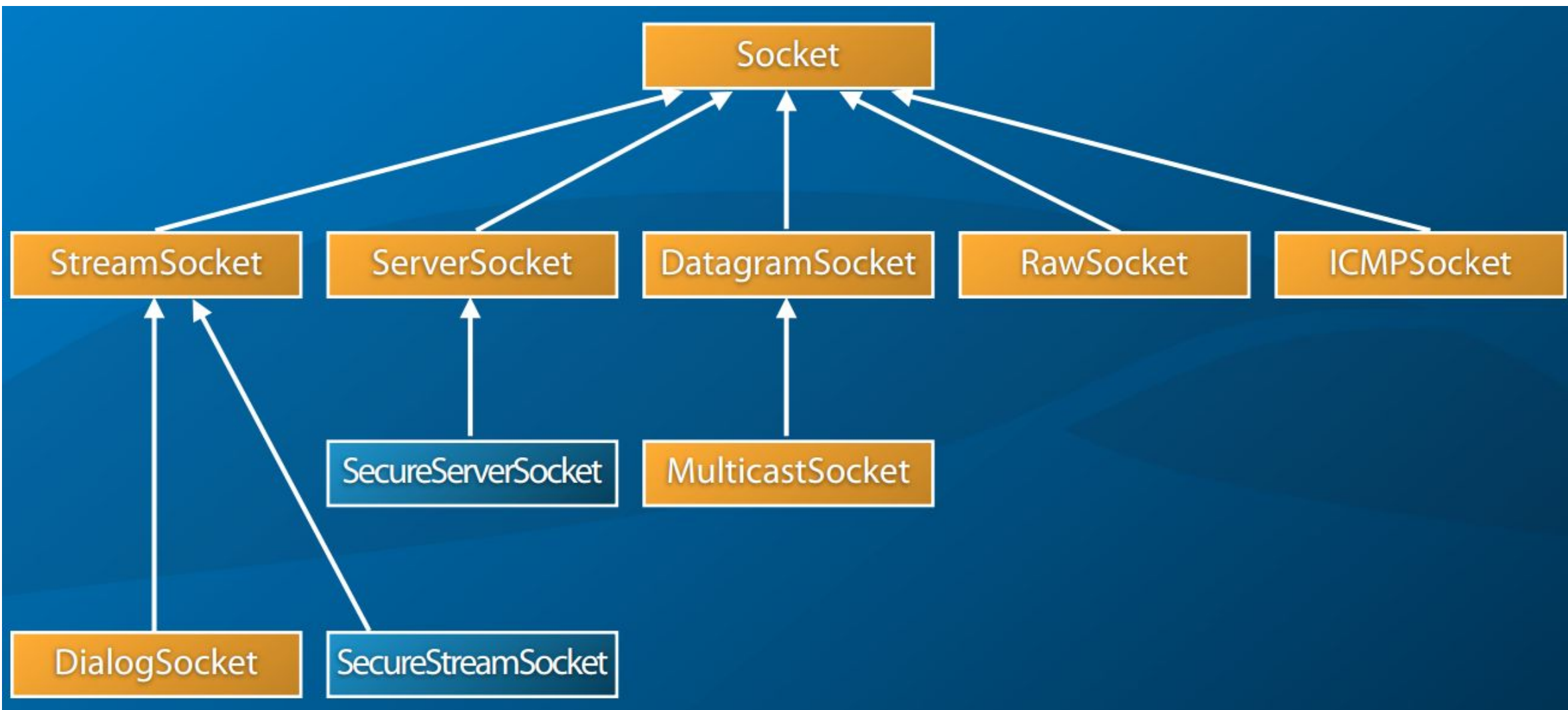
2. Poco::Net::DNS::flushCache() 로 캐시를 지운 후

- Poco::Net::DNS::hostByName()
- Poco::Net::DNS::hostByAddress()
- Poco::Net::DNS::resolve(hostname)
- Poco::Net::DNS::resolve(hostaddress)

로 얻은 Poco::Net::HostEntry 요소를 열거

Socket

- The socket classes in POCO are implemented using the Pimpl idiom.
- POCO sockets are a very thin layer on top of BSD sockets and thus incur a minimal performance overhead – basically an additional call to a (virtual) function.
- A Socket object supports full value semantics (including all comparison operators).
- A Socket object stores only a pointer to a corresponding SocketImpl object. SocketImpl objects are reference counted.



The Socket Class

- Poco::Net::Socket is the root class of the sockets inheritance tree.
- It supports methods that can be used with some or all kinds of sockets, like:
 - select() and poll()
 - setting and getting various socket options (timeouts, buffer sizes, reuse address flag, etc.)
 - getting the socket's address and the peer's address

The StreamSocket Class

클라이언트

- `Poco::Net::StreamSocket` is used for creating a TCP connection to a server.
- Use `sendBytes()` and `receiveBytes()` to send and receive data, or use the `Poco::Net::SocketStream` class, which provides an I/O streams interface to a `StreamSocket`.

```
#include "Poco/Net/SocketAddress.h"
#include "Poco/Net/StreamSocket.h"
#include "Poco/Net/SocketStream.h"
#include "Poco/StreamCopier.h"
#include <iostream>

int main(int argc, char** argv)
{
    Poco::Net::SocketAddress sa("www.appinf.com", 80);
    Poco::Net::StreamSocket socket(sa)
    Poco::Net::SocketStream str(socket);

    str << "GET / HTTP/1.1\r\n"
          "Host: www.appinf.com\r\n"
          "\r\n";
    str.flush();
    Poco::StreamCopier::copyStream(str, std::cout);
    return 0;
}
```

The ServerSocket Class

서버

- `Poco::Net::ServerSocket` is used to create a TCP server socket.
- It is pretty low level.
- For an actual server, consider using the `TCPServer` or the `Reactor` framework.

```
#include "Poco/Net/ServerSocket.h"
#include "Poco/Net/StreamSocket.h"
#include "Poco/Net/SocketStream.h"
#include "Poco/Net/SocketAddress.h"

int main(int argc, char** argv)
{
    Poco::Net::ServerSocket srv(8080); // does bind + listen
    for (;;)
    {
        Poco::Net::StreamSocket ss = srv.acceptConnection();
        Poco::Net::SocketStream str(ss);
        str << "HTTP/1.0 200 OK\r\n"
            "Content-Type: text/html\r\n"
            "\r\n"
            "<html><head><title>My 1st Web Server</title></head>"
            "<body><h1>Hello, world!</h1></body></html>"
            << std::flush;
    }
    return 0;
}
```


UDP Sockets

- `Poco::Net::DatagramSocket` is used to send and receive UDP packets.
- `Poco::Net::MulticastSocket` is a subclass of `Poco::Net::DatagramSocket` that allows you to send multicast datagrams.
- To receive multicast messages, you must join a multicast group, using `MulticastSocket::joinGroup()`.
- You can specify the network interface used for sending and receiving multicast messages.

```
// DatagramSocket send example
#include "Poco/Net/DatagramSocket.h"
#include "Poco/Net/SocketAddress.h"
#include "Poco/Timestamp.h"
#include "Poco/DateTimeFormatter.h"

int main(int argc, char** argv)
{
    Poco::Net::SocketAddress sa("localhost", 514);
    Poco::Net::DatagramSocket dgs;
    dgs.connect(sa);

    Poco::Timestamp now;
    std::string msg = Poco::DateTimeFormatter::format(now,
        "<14>%w %f %H:%M:%S Hello, world!");
    dgs.sendBytes(msg.data(), msg.size());
    return 0;
}
```

```
// DatagramSocket receive example
#include "Poco/Net/DatagramSocket.h"
#include "Poco/Net/SocketAddress.h"
#include <iostream>

int main(int argc, char** argv)
{
    Poco::Net::SocketAddress sa(Poco::Net::IPAddress(), 514);
    Poco::Net::DatagramSocket dgs(sa);

    char buffer[1024];
    for (;;)
    {
        Poco::Net::SocketAddress sender;
        int n = dgs.receiveFrom(buffer, sizeof(buffer)-1, sender);
        buffer[n] = '\0';
        std::cout << sender.toString() << ": " << buffer << std::endl;
    }
    return 0;
}
```

The TCPServer Framework

- Poco::Net::TCPServer **implements a multithreaded** TCP server.
- The server uses a ServerSocket to accept incoming connections. You must put the ServerSocket into listening mode before passing it to the TCPServer.
- The server maintains a **queue for incoming connections**.
- **A variable number of worker threads fetches connections from the queue to process them**. The number of worker threads is adjusted automatically, depending on the number of connections waiting in the queue.

The TCPServer Framework (cont'd)

- The number of connections in the queue can be limited to prevent the server from being flooded with requests. **Incoming connections that no longer fit into the queue are closed immediately.**
- TCPServer creates its own thread that accepts connections and places them in the queue.
- TCPServer uses **TCPServerConnection** objects to handle a connection. **You must create your own subclass of TCPServerConnection**, as well as a factory for it. The factory object is passed to the constructor of TCPServer

The TCPServer Framework (cont'd)

- Your subclass of **TCPServerConnection** must **override the run()** method. In the run() method, you handle the connection.
- **When run() returns, the TCPServerConnection object will be deleted, and the connection closed.**
- A new TCPServerConnection will be created for every accepted connection.

```

class MyConnection : public TCPServerConnection
{
public:
    MyConnection(const StreamSocket &socket) : TCPServerConnection(socket)    {    }
    virtual ~MyConnection()    {    }

    virtual void run()
    {
        static const char message[] = "This is TCP server sample\n";
        socket().sendBytes(message, sizeof(message) - 1);
    }
};

```

- run() 함수를 재 정의하여 클라이언트와 통신에 대해서 기술한다.
- run() 함수를 빠져나오면 TCPServerConnection 인스턴스는 자동적으로 파괴 되고, 접속이 자동적으로 끊어진다.
- run() 함수 내에서는 socket() 메소드를 통해서 소켓에 접근하고, 데이터 송수신을 할 수 있다.

```
class MyConnectionFactory : public TCPServerConnectionFactory
{
public:
    MyConnectionFactory()    {    }
    virtual ~MyConnectionFactory()    {    }

    virtual TCPServerConnection* createConnection(const StreamSocket &socket)
    {
        return new MyConnection(socket);
    }
};
```

createConnection(const StreamSocket&) 함수를 재정의 한다.
이 함수는 Factory Method 패턴으로 TCPServerConnection
인스턴스를 돌려주는 일을 한다.


```

static const Poco::UInt16 SERVER_PORT = 8888;

ServerSocket sock(SERVER_PORT);    sock.listen();
TCPServer server(new MyConnectionFactory(), sock);

printf("Simple TCP Server Application.\n");
printf("command:\n");
printf("  q | quit : Quit application.\n");
printf("\n");    server.start();

bool running = true;
while(running)
{
    char buff[256];
    if(fgets(buff, sizeof(buff), stdin) == NULL)
    {
        running = false;
    }
    else
    {
        buff[strlen(buff) - 1] = '\0';

        if(strcmp(buff, "quit") == 0 || strcmp(buff, "q") == 0)
        {
            printf(">> QUIT command accepted.\n");
            running = false;
        }
        else
        {
            printf(">> UNKNOWN command.\n");
        }
    }
}

server.stop();
return 0;

```

TCPServer 클래스를 인스턴스화 하고,
대기 상태에 들어간다.

주의점으로 TCPServer::start()를
호출하기 전에 **ServerSocket을 리스닝
상태로 하는 것**(listen 함수를 호출)을
잊지 않도록 한다.

```

class EchoServiceHandler
{
public:
    EchoServiceHandler(StreamSocket& socket, SocketReactor& reactor)
        : _socket(socket)
        , _reactor(reactor)
        , _pBuffer(new char[BUFFER_SIZE])
        , _onWritableAdded(false)
        , _onTimeoutAdded(false)
    {
        Application& app = Application::instance();
        app.logger().information("Connection from " + socket.peerAddress().toString());

        _reactor.addEventHandler(_socket, NObserver<EchoServiceHandler, ReadableNotification>(*this, &EchoServiceHandler::onReadable));
        _reactor.addEventHandler(_socket, NObserver<EchoServiceHandler, ShutdownNotification>(*this, &EchoServiceHandler::onShutdown));
        _reactor.addEventHandler(_socket, NObserver<EchoServiceHandler, ErrorNotification>(*this, &EchoServiceHandler::onError));
        _reactor.addEventHandler(_socket, NObserver<EchoServiceHandler, IdleNotification>(*this, &EchoServiceHandler::onIdle));
    }
}

```

Code: TCP Echo Server/Client

“에코 서버/클라이언트”

대부분의 네트워크 프로그래밍 책, 강좌에서 다루는 예제.

네트워크 프로그래밍 핵심을 쉽게 파악할 수 있다.

```

#include "Poco/Net/StreamSocket.h"
#include "Poco/Net/SocketAddress.h"

const Poco::UInt16 PORT = 32452;

int main()
{
    std::cout << "서버에 연결 시도..." << std::endl;
    Poco::Net::StreamSocket ss;

    try
    {
        ss.connect(Poco::Net::SocketAddress("localhost", PORT));

        for (int i = 0; i < 7; ++i)
        {
            char szMessage[128] = { 0, };
            sprintf_s(szMessage, 128 - 1, "%d - Send Message", i);
            int nMsgLen = (int)strnlen_s(szMessage, 128 - 1);

            ss.sendBytes(szMessage, nMsgLen);

            std::cout << "서버에 보낸 메시지: " << szMessage << std::endl;

            char buffer[256] = { 0, };
            auto len = ss.receiveBytes(buffer, sizeof(buffer));

            if (len <= 0)
            {
                std::cout << "서버와 연결이 끊어졌습니다" << std::endl;
                break;
            }

            std::cout << "서버로부터 받은 메시지: " << buffer << std::endl;
        }

        ss.close();
    }
}

```

```

#include "Poco/Net/ServerSocket.h"
#include "Poco/Net/StreamSocket.h"
#include "Poco/Net/SocketAddress.h"

const Poco::UInt16 PORT = 32452;

int main()
{
    std::cout << "서버 초기화 시작" << std::endl;

    Poco::Net::SocketAddress server_add(PORT);
    Poco::Net::ServerSocket server_sock(server_add);

    std::cout << "서버 초기화 완료. 클라이언트 접속 대기 중..." << std::endl;

    while (true)
    {
        Poco::Net::StreamSocket ss = server_sock.acceptConnection();

        try
        {
            while (true)
            {
                char buffer[256] = { 0, };
                int n = ss.receiveBytes(buffer, sizeof(buffer));
                std::cout << "클라이언트에서 받은 메시지: " << buffer << std::endl;

                if (n <= 0) {
                    break;
                }

                char szSendMessage[256] = { 0, };
                sprintf_s(szSendMessage, 256 - 1, "Re:%s", buffer);
                int nMsgLen = (int)strnlen_s(szSendMessage, 256 - 1);

                ss.sendBytes(szSendMessage, nMsgLen);
            }

            std::cout << "클라이언트와 연결이 끊어졌습니다" << std::endl;
        }
    }
}

```

구성(C): 활성(Debug) 플랫폼(P): x64 구성 관리자(O)...

구성 속성	공용 프로젝트 콘텐츠																				
<ul style="list-style-type: none"> 일반 고급 디버깅 VC++ 디렉터리 <ul style="list-style-type: none"> C/C++ <ul style="list-style-type: none"> 일반 최적화 전처리기 코드 생성 언어 미리 컴파일된 헤더 출력 파일 찾아보기 정보 고급 	<ul style="list-style-type: none"> 공용 포함 디렉터리 <table border="1"> <tr> <td>모든 헤더 파일이 공용임</td> <td>아니요</td> </tr> <tr> <td>공용 C++ 모듈 디렉터리</td> <td>예</td> </tr> <tr> <td>모든 모듈이 공용임</td> <td>예</td> </tr> </table> 일반 <table border="1"> <tr> <td>실행 가능 디렉터리</td> <td>\$(VC_ExecutablePath_x64);\$(CommonExecutablePath)</td> </tr> <tr> <td>포함 디렉터리</td> <td>F:\WC++\thirdparty\poco\Foundation\WinInclude;F:\WC++\thirdparty\poco\Foundation\WinInclude</td> </tr> <tr> <td>참조 디렉터리</td> <td>\$(VC_ReferencesPath_x64);</td> </tr> <tr> <td>라이브러리 디렉터리</td> <td>F:\WC++\thirdparty\poco\WinLib64;\$(LibraryPath)</td> </tr> <tr> <td>라이브러리 WinRT 디렉터리</td> <td>\$(WindowsSDK_MetadataPath);</td> </tr> <tr> <td>소스 디렉터리</td> <td>\$(VC_SourcePath);</td> </tr> <tr> <td>제외 디렉터리</td> <td>\$(CommonExcludePath);\$(VC_ExecutablePath_x64);\$(VC_Lib</td> </tr> </table> 	모든 헤더 파일이 공용임	아니요	공용 C++ 모듈 디렉터리	예	모든 모듈이 공용임	예	실행 가능 디렉터리	\$(VC_ExecutablePath_x64);\$(CommonExecutablePath)	포함 디렉터리	F:\WC++\thirdparty\poco\Foundation\WinInclude;F:\WC++\thirdparty\poco\Foundation\WinInclude	참조 디렉터리	\$(VC_ReferencesPath_x64);	라이브러리 디렉터리	F:\WC++\thirdparty\poco\WinLib64;\$(LibraryPath)	라이브러리 WinRT 디렉터리	\$(WindowsSDK_MetadataPath);	소스 디렉터리	\$(VC_SourcePath);	제외 디렉터리	\$(CommonExcludePath);\$(VC_ExecutablePath_x64);\$(VC_Lib
모든 헤더 파일이 공용임	아니요																				
공용 C++ 모듈 디렉터리	예																				
모든 모듈이 공용임	예																				
실행 가능 디렉터리	\$(VC_ExecutablePath_x64);\$(CommonExecutablePath)																				
포함 디렉터리	F:\WC++\thirdparty\poco\Foundation\WinInclude;F:\WC++\thirdparty\poco\Foundation\WinInclude																				
참조 디렉터리	\$(VC_ReferencesPath_x64);																				
라이브러리 디렉터리	F:\WC++\thirdparty\poco\WinLib64;\$(LibraryPath)																				
라이브러리 WinRT 디렉터리	\$(WindowsSDK_MetadataPath);																				
소스 디렉터리	\$(VC_SourcePath);																				
제외 디렉터리	\$(CommonExcludePath);\$(VC_ExecutablePath_x64);\$(VC_Lib																				

구성(C): 활성(Debug) 플랫폼(P): x64 구성 관리자(O)...

구성 속성	추가 종속성
<ul style="list-style-type: none"> 일반 고급 디버깅 VC++ 디렉터리 C/C++ 링커 <ul style="list-style-type: none"> 일반 입력 매니페스트 파일 디버깅 시스템 최적화 포함 IDL 	<ul style="list-style-type: none"> 모든 기본 라이브러리 무시 특정 기본 라이브러리 무시 모듈 정의 파일 어셈블리에 모듈 추가 관리되는 리소스 파일 포함 강제 기호 참조 지연 로드된 DLL 어셈블리와 리소스 링크

React 프레임워크

reactor pattern

Reactor 패턴은 하나 이상의 클라이언트로 부터의 요청(입력)을 동시처리하기 위해서 사용하는 패턴이다. 서버는 각 입력에 대해서 받을 이벤트를 동적으로 등록/해제하는 식으로 처리해야할 입력과 이벤트를 관리할 수 있다.

1. 새로운 연결이 들어온다.
2. 이 연결과 연결에 대해서 처리할 이벤트를 일괄 등록한다.
 - 이벤트는 "입력 데이터", "출력 데이터", "종료"등이 있을 수 있다.
3. 주기적으로 이벤트를 확인한다.
4. 이벤트가 발생하면 **dispatcher** 에 전달해서 처리한다.

2번이 **multiplexing**과정이다. 즉 여러 개의 입력을 처리할 수 있도록 다중화 한다. 3번 과정이 **demultiplexing** 과정으로 이벤트가 발생한 "하나의" 입력을 찾아낸다. 마지막으로 dispatcher이 데이터를 처리한다.

기본적으로 demultiplexing 에서 multiplexing의 과정이 단일 스레드에서의 처리에 적합한 관계로 단일 프로세스 프로그램에서 두 개 이상의 입력을 처리하기 위해서 주로 사용한다.

Reactor pattern은 유닉스에서는 **select**, **poll**을 이용해서 매우 오래전 부터 **입출력다중화**라는 이름으로 사용해왔다. 아래는 select의 전형적인 입력 처리 구조다.

- FD_SET을 이용, multiplexing 과정을 거쳐서 입력을 등록하고
- select를 이용, demultiplexing 과정을 거쳐서 이벤트가 발생한 입력을 검사
- 입력의 데이터를 처리

https://www.joinc.co.kr/w/Site/SoftWare_engineering/pattern/reactor

- Reactor 프레임워크는 논블록 소켓으로 select(**Linux에서는 epoll**)과 NotificationCenter를 조합한 것.
- Reactor 프레임워크의 핵심 클래스는 Poco::Net::SocketReactor.
- 이 클래스는 임의 수의 소켓 상태를 감시하여 소켓 상태가 변화하면 통지해준다(읽기 가능, 쓰기 가능, 오류 발생 등).
- 이런 소켓 상태 변화 통지에 대한 처리를 하기 위해서는 SocketReactor 클래스에 각각의 통지와 관련된 콜백 함수(핸들러)를 등록해야 한다

The Reactor Framework

- The Reactor framework is based on the Reactor design pattern, described by Douglas C. Schmidt in PLOP.
- Basically, it's non-blocking sockets and `select()` combined with a `NotificationCenter`.
- The `Poco::Net::SocketReactor` observes the state of an arbitrary number of sockets, and dispatches a notification if a state of a socket changes (it becomes readable, writable, or an error occurred).
- Classes register a socket and a callback function with the `SocketReactor`.

The Reactor Framework (cont'd)

- The SocketReactor is used together with a SocketAcceptor.
- The SocketAcceptor waits for incoming connections.
- **When a new connection request arrives, the SocketAcceptor accepts the connection and creates a new ServiceHandler object that handles the connection.**
- When the ServiceHandler is done with the connection, it must delete itself.

필요한 헤더 파일

Reactor 프레임워크를 사용하려면 다음 헤더 파일이 필요하다.

```
#include <Poco/Net/SocketReactor.h>
```

```
#include <Poco/Net/SocketAcceptor.h>
```

서버 생성과 시작

```
const Poco::UInt16 PORT = 11021;
```

```
Poco::Net::SocketReactor reactor;
```

```
Poco::Net::ServerSocket serverSocket(PORT);
```

```
Poco::Net::SocketAcceptor<Session> acceptor(serverSocket, reactor);
```

```
reactor.run();
```

리액터 방식으로 클라이언트 접속 처리
시작

ServiceHandler 에 해당.
Reactor 프레임워크의 대부분 로직
(네트워크 처리와 관련된)은 이
ServiceHandler에 정의한다

run() 멤버 함수를 호출하면 대기 상태가 된다.
서버를 종료하기 위해 대기 상태에서 빠져 나오려면 SocketReactor의
stop() 멤버 함수를 호출한다.

ServiceHandler 정의

Reactor 프레임워크의 ServiceHandler는 앞에 설명했듯이 템플릿 클래스다.

그래서 우리가 임의의 클래스를 정의하면 된다.

클래스의 생성자에서는
클라이언트의 접속과 관련된 처리와 소켓 이벤트에 호출될 함수를
정의하고,

클래스의 소멸자에서는
생성자에서 등록한 소켓 이벤트 핸들러를 해제하는 기능을 꼭
구현해야 한다.

```
class Session {  
public:  
  
    Session(Poco::Net::StreamSocket& socket,  
            Poco::Net::SocketReactor& reactor) :  
        m_Socket(socket),  
        m_Reactor(reactor)  
    {}  
  
    ~Session() {}  
  
private:
```

꼭 필요한 멤버

```
    Poco::Net::StreamSocket m_Socket;  
    Poco::Net::SocketReactor& m_Reactor;  
};
```

```

class Session {
public:
    Session(Poco::Net::StreamSocket& socket, Poco::Net::SocketReactor& reactor) :
        m_Socket(socket),
        m_Reactor(reactor)
    {
        m_PeerAddress = socket.peerAddress().toString();
        std::cout << "connection from " << m_PeerAddress << " ..." << std::endl;

        m_Reactor.addEventHandler(m_Socket,
                                   Poco::Observer<Session,
                                   Poco::Net::ReadableNotification>(
                                       *this, &Session::onReadable)
                                   );
    }

    ~Session()
    {
        std::cout << m_PeerAddress << " disconnected ..." << std::endl;
        m_Reactor.removeEventHandler(m_Socket,
                                      Poco::Observer<Session,
                                      Poco::Net::ReadableNotification>(
                                          *this, &Session::onReadable)
                                      );
    }

    void onReadable(Poco::Net::ReadableNotification* pNotification)
    {
        pNotification->release();

        try
        {
            char buffer[256] = { 0, };
            int n = m_Socket.receiveBytes(buffer, sizeof(buffer));
        }
    }
}

```

이 코드를 호출해야
해당 이벤트를
처리했음을 Reactor
프레임워크에
알려준다

Session 클래스의 생성자에서는
소켓 상태가 변경 되었을 때
(이벤트가 발생할 때) 호출할
핸들러를 반드시
`m_Reactor.addEventHandler`를
사용하여 등록해야 한다.

생성자에서 등록한 핸들러를
`m_Reactor.removeEventHandler`를
사용하여 해제해야 한다.

데이터 읽기 가능 핸들러의 이름은 어떤
것으로 하든지 괜찮지만, 읽기 가능
핸들러는 함수의 파라미터 타입이
`onReadable` 함수와 같아야 한다.
이벤트 마다 파라미터 타입은 다르다

```

void onReadable(Poco::Net::ReadableNotification* pNotification)
{
    pNotification->release();

    try
    {
        char buffer[256] = { 0, };
        int n = m_Socket.receiveBytes(buffer, sizeof(buffer));
        if (n > 0)
        {
            char szSendMessage[256] = { 0, };
            sprintf_s(szSendMessage, 256 - 1, "Re:%s", buffer);
            int nMsgLen = (int)strlen_s(szSendMessage, 256 - 1);

            m_Socket.sendBytes(szSendMessage, nMsgLen);

            std::cout << "클라이언트에서 받은 메시지: " << buffer << std::endl;
        }
        else
        {
            m_Socket.shutdown();
            delete this; // 메모리 해제하지 않으면 소멸자가 호출되지 않는다.
        }
    }
    catch (Poco::Exception& exc)
    {
        std::cout << "EchoServer: " << exc.displayText() << std::endl;

        m_Socket.shutdown();
        delete this;
    }
}

```

클라이언트가 서버에 접속하면 Reactor 프레임워크는 ServiceHandler 객체를 자동으로 동적 할당 한다. 그러나 ServiceHandler 객체의 메모리 해제는 우리가 직접 해야 한다. 메모리 해제를 해야 하는 상황은 클라이언트와 서버가 접속이 끊어졌을 때다.

소켓 이벤트

- **class ReadableNotification**
데이터 읽기가 가능할 때 발생하는 이벤트
- **class ErrorNotification**
소켓에 오류가 생겼을 때 발생하는 이벤트
- **class IdleNotification**
Poco::Net::select 호출 시 react 할 소켓이 없는 경우 발생하는 이벤트
- **class ShutdownNotification**
SocketReactor가 셧다운 할 때 발생하는 이벤트
- **class TimeoutNotification**
Poco::Net::select 호출 후 지정 시간까지 이벤트가 없을 때 발생하는 이벤트
- **class WritableNotification**
데이터 쓰기가 가능할 때 발생하는 이벤트.

앞의 이벤트 중 Timeout 이벤트는 SocketReactor 객체를 생성할 때 타임아웃 할 시간을 지정할 수 있다.

```
Poco::Timespan timeout(10, 0); // 10 초  
Poco::Net::SocketReactor reactor(timeout);
```

또는 SocketReactor 객체 생성 후에 SocketReactor 클래스의 멤버 함수를 사용하여 지정할 수도 있다.

```
void setTimeout(const Poco::Timespan & timeout);
```

코드를 잠깐 봅시다...

멀티스레드 사용하기

Reactor 프레임워크는 멀티스레드를 사용하지 않는다는 단점이 있다.

요즘 PC의 CPU 코어는 최소 2개 이상이므로 네트워크 처리에 1개의 스레드만 사용한다는 것은 효율적이지 못한 면이 있다.
다행히 Reactor 프레임워크는 이 문제를 해결하였다.

Reactor 프레임워크를 초기화할 때 사용한 `Poco::Net::SocketAcceptor` 클래스 대신 **`Poco::Net::ParallelSocketAcceptor`** 클래스를 사용하는 것만으로 끝이다. 더 이상 할 것은 없다.

```
Poco::Net::ServerSocket serverSocket(PORT);
```

```
Poco::Net::SocketReactor reactor;
```

```
Poco::Net::ParallelSocketAcceptor<Session, Poco::Net::SocketReactor> acceptor(serverSocket, reactor);
```

ParallelSocketAcceptor 클래스의 이름을 보면 클라이언트 접속 처리에 멀티스레드를 사용할 것 같지만, **접속 처리에는 멀티스레드를 사용하지 않고 접속 처리 후부터 멀티스레드를 사용한다.**

이를 꼭 주의해야 한다.

디폴트 멀티스레드 수는 PC의 코어 수만큼 사용한다.
사용할 멀티스레드 수를 변경하고 싶다면
ParallelSocketAcceptor 객체 생성 시 사용할 멀티스레드 수를
지정하면 된다.

ParallelSocketAcceptor ⓘ

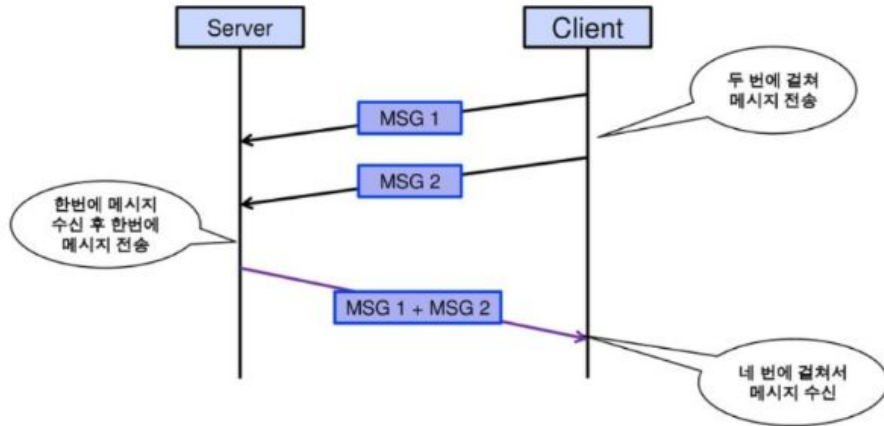
```
ParallelSocketAcceptor(  
    ServerSocket & socket,  
    SocketReactor & reactor,  
    unsigned threads = Poco::Environment::processorCount ()  
);
```

채팅 서버 만들기

패킷, TCP의 스트림 처리

```
class Packet
{
    public UInt16 TotalSize;
    public UInt16 Id;
    public SByte Type;
}
```

TCP



```
while
{
    소켓으로 데이터를 받기 요청을 한다

    if(연결이 끊어졌다면)
        break;

    받은 데이터를 버퍼에 저장한다(데이터가 남아 있다면 뒤에)

    while
    {
        if(버퍼에 있는 데이터 크기가 헤드 보다 작다면)
            break;

        패킷의 크기를 얻는다

        if(버퍼에 있는 데이터 크기가 패킷의 크기보다 작으면)
            break;

        하나의 패킷 완성!
    }

    버퍼에 패킷으로 만들지 못한 데이터가 있다면 버퍼의 앞으로 옮긴다
}
```

리모트에서 받은 데이터를
쉽게 패킷으로 만드는 방법:

I/O 호출을 두 번
발생시키지만 받은 데이터를
패킷을 만드는 로직이
간단하다

```
// read message (via stream) with the <size,content> message structure
protected static bool ReadMessageBlocking(NetworkStream stream, int MaxMessageSize, out byte[] content)
{
    content = null;

    // create header buffer if not created yet
    if (header == null)
        header = new byte[4];

    // read exactly 4 bytes for header (blocking)
    if (!stream.ReadExactly(header, 4))
        return false;

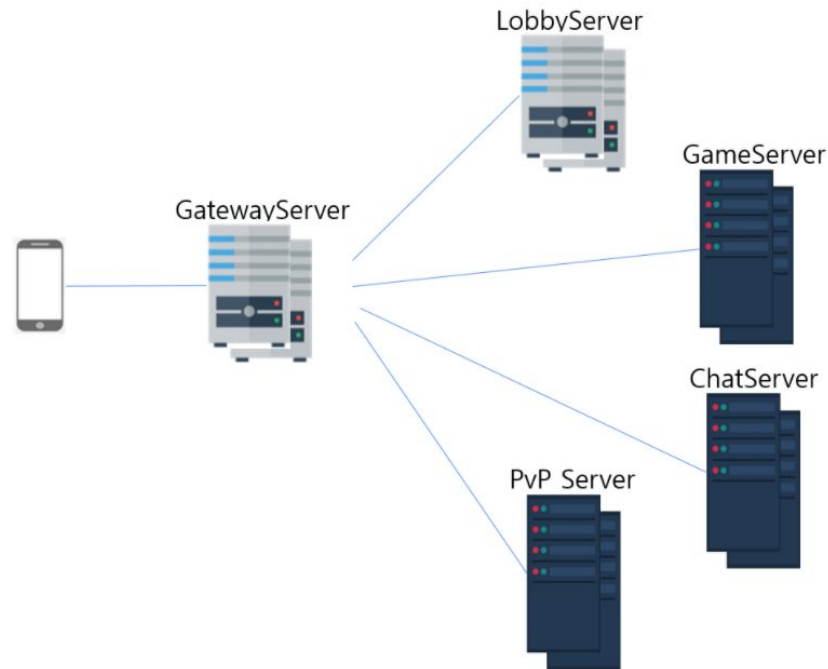
    // convert to int
    int size = Utils.BytesToIntBigEndian(header);

    // protect against allocation attacks. an attacker might send
    // multiple fake '2GB header' packets in a row, causing the server
    // to allocate multiple 2GB byte arrays and run out of memory.
    if (size <= MaxMessageSize)
    {
        // read exactly 'size' bytes for content (blocking)
        content = new byte[size];
        return stream.ReadExactly(content, size);
    }
    Logger.LogWarning("ReadMessageBlocking: possible allocation attack with a header of: " + size + " bytes.");
    return false;
}
```

<https://github.com/vis2k/Telepathy/blob/master/Telepathy/Common.>

고성능을 위해서

- 게이트웨이 서버나 mq 사용. windows에서 IOCP 사용하지 못하는 단점 해결
- 메모리 할당. TCMalloc 같은 메모리 풀 사용
- 성능 좋고 안전한 스레드 세이프한 컨테이너 사용



master 1 branch 0 tags

Go to file

Add file



jacking75 No commit message

9cf5a1f 4 days ago

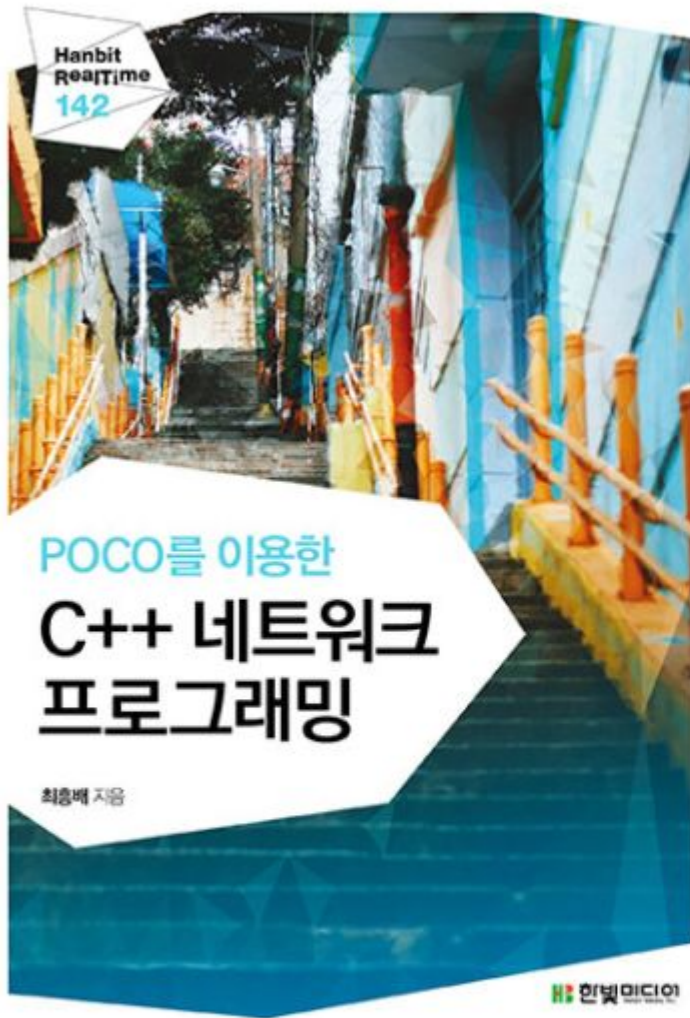



POCO_Net_Samples	No commit message	
Semina_2017-09-16	No commit message	
Semina_2021-03-17	No commit message	
exampleCodes	No commit message	
LICENSE	Initial commit	1.06 KB
README.md	Update README.md	170 Bytes

README.md


POCO 이용하 네트워크 프로그래밍 예제


https://github.com/jacking75/Poco_network_sample




 미리보기

POCO를 이용한 C++ 네트워크 프로그래밍


 리얼타임 eBook

 집필서

 판매중



- 저자 : 최흥배
- 출간 : 2017-04-28
- 페이지 : 123 쪽
- ISBN : 9788968488498

 TAG POCO , C++ , 네트워크프로그래밍 , 소켓 , 웹소켓 , Boost , 라이브러리

