# POCO Network Programming

## Writing network and internet applications with Poco::Net

*"Without a good library, most interesting tasks are hard to do in C++; but given a good library, almost any task can be made easy."*

Bjarne Stroustrup

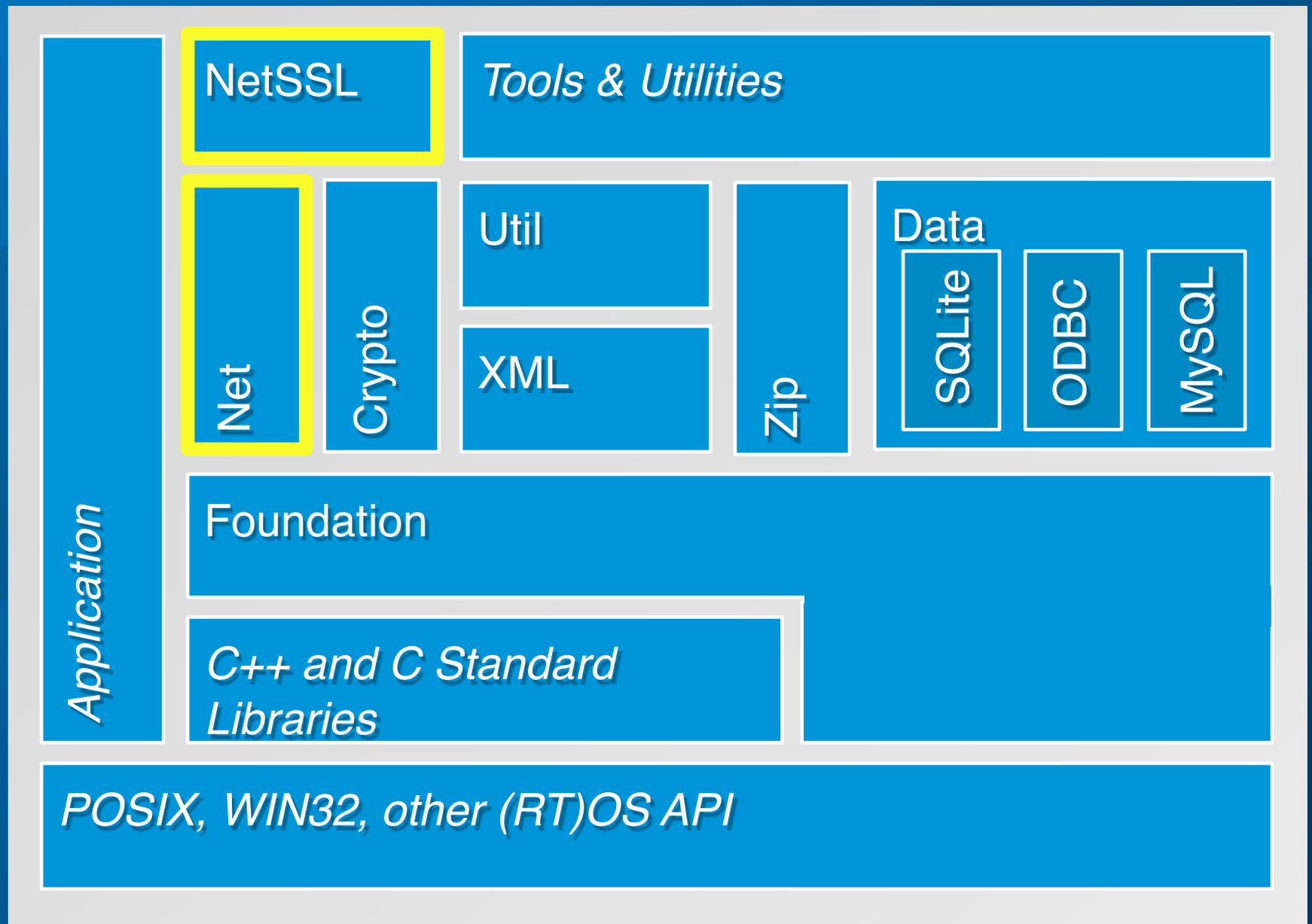
# Overview

- What is POCO ?
- Current and next releases review
- Network addressing and naming
- Sockets
- ICMP Package
- The TCP Server Framework
- The Reactor Framework
- High Level Protocols: HTTP, FTP and E-Mail

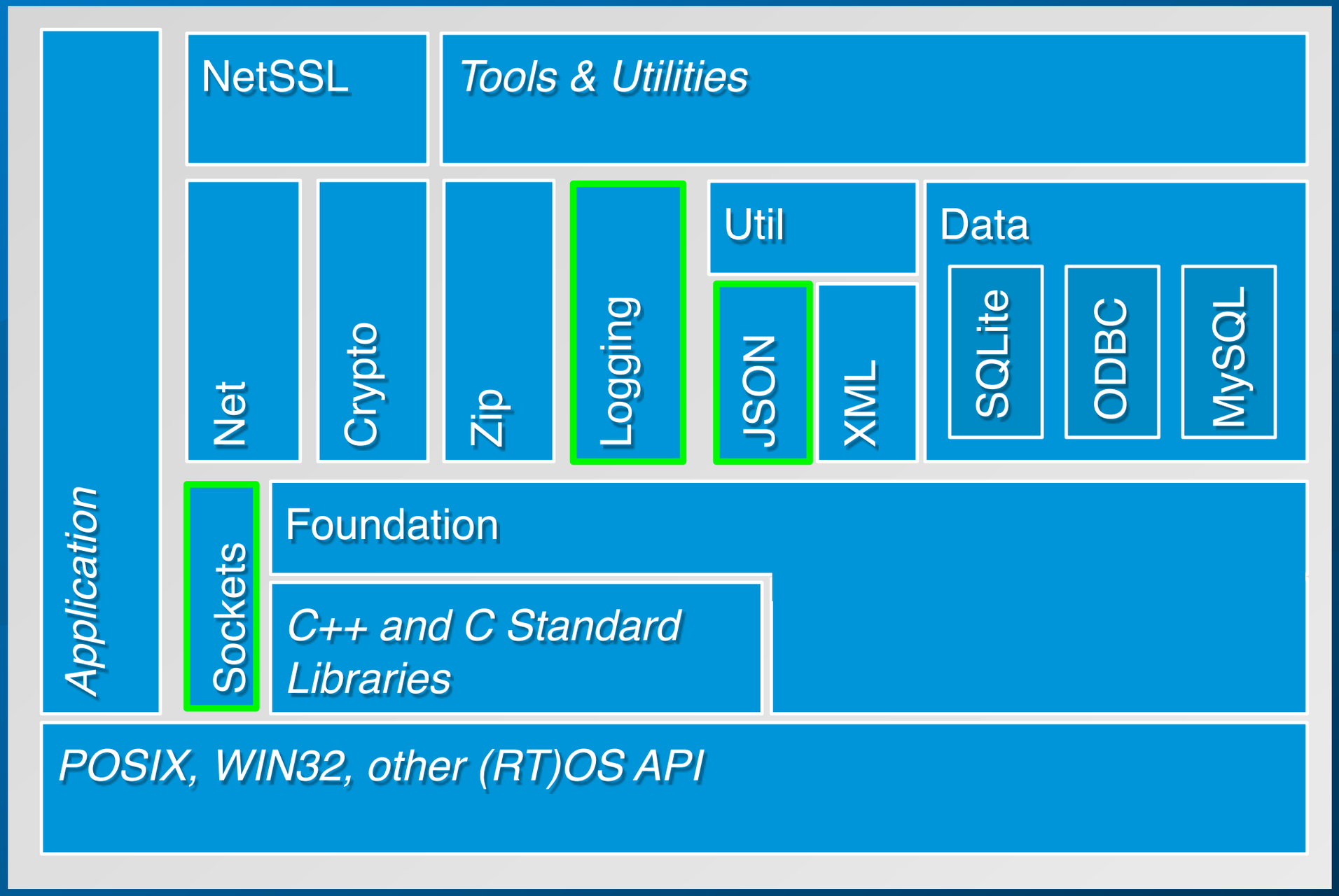
# The C++ POrtable COmponents Libraries

- A collection of C++ class libraries, similar in concept to the Java Class Library, the .NET Framework or Apple's Cocoa
- Focused on:
  - "internet-age" network-centric applications;
  - clean, well-designed and readable code
  - practical everyday use and developer productivity
  - simplicity (“as simple as possible ...”) and ease of use
  - consistent look-and-feel
- written in Standard C++ and based on the Standard Library/STL (“don’t reinvent the wheel” paradigm)
- highly portable and available on many different platforms
- Open Source, licensed under the Boost Software License

# Current Release (1.4.3)



# Next Release (1.5)



# Hello Poco::Net World !

```
#include "Poco/Net/DialogSocket.h"
#include "Poco/Net/SocketAddress.h"
#include <iostream>

using Poco::Net::DialogSocket;
using Poco::Net::SocketAddress;

int main(int, char**)
{
    EchoServer echoServer;
    DialogSocket ds;
    ds.connect(SocketAddress("localhost", echoServer.port()));

    ds.sendMessage("Hello, world!");
    std::string str;
    ds.receiveMessage(str);
    std::cout << str << std::endl; // Hello, world!
}
```

```
#include "EchoServer.h"
#include "Poco/Net/StreamSocket.h"
#include "Poco/Net/SocketAddress.h"
#include "Poco/Timespan.h"
#include <iostream>

using Poco::Net::Socket;
using Poco::Net::StreamSocket;
using Poco::Net::SocketAddress;

EchoServer::EchoServer(): _socket(SocketAddress()), _thread("EchoServer"), _stop(false)
{ _thread.start(*this);} }

EchoServer::~EchoServer()
{ _stop = true; _thread.join(); }

Poco::UInt16 EchoServer::port() const
{ return _socket.address().port(); }
```

```
void EchoServer::run()
{
     Poco::Timespan span(250000);
     while (!_stop)
     {
          if (_socket.poll(span, Socket::SELECT_READ))
          {
               StreamSocket ss = _socket.acceptConnection();
               char buffer[256];
               int n = ss.receiveBytes(buffer, sizeof(buffer));
               while (n > 0 && !_stop)
               {
                    ss.sendBytes(buffer, n);
                    n = ss.receiveBytes(buffer, sizeof(buffer));
               }
          }
     }
}
```

# IP Addresses

- The Poco::Net::IPAddress class stores an IPv4 or IPv6 host address
- 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
- To test for certain properties of an IP address, use: isWildcard(), isBroadcast(), isLoopback(), isMulticast(), etc.
- IPAddress supports full value semantics, including all relational operators

# 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, using:
  - IP address and a port number
  - a string containing both an IP address and a port number separated by a colon

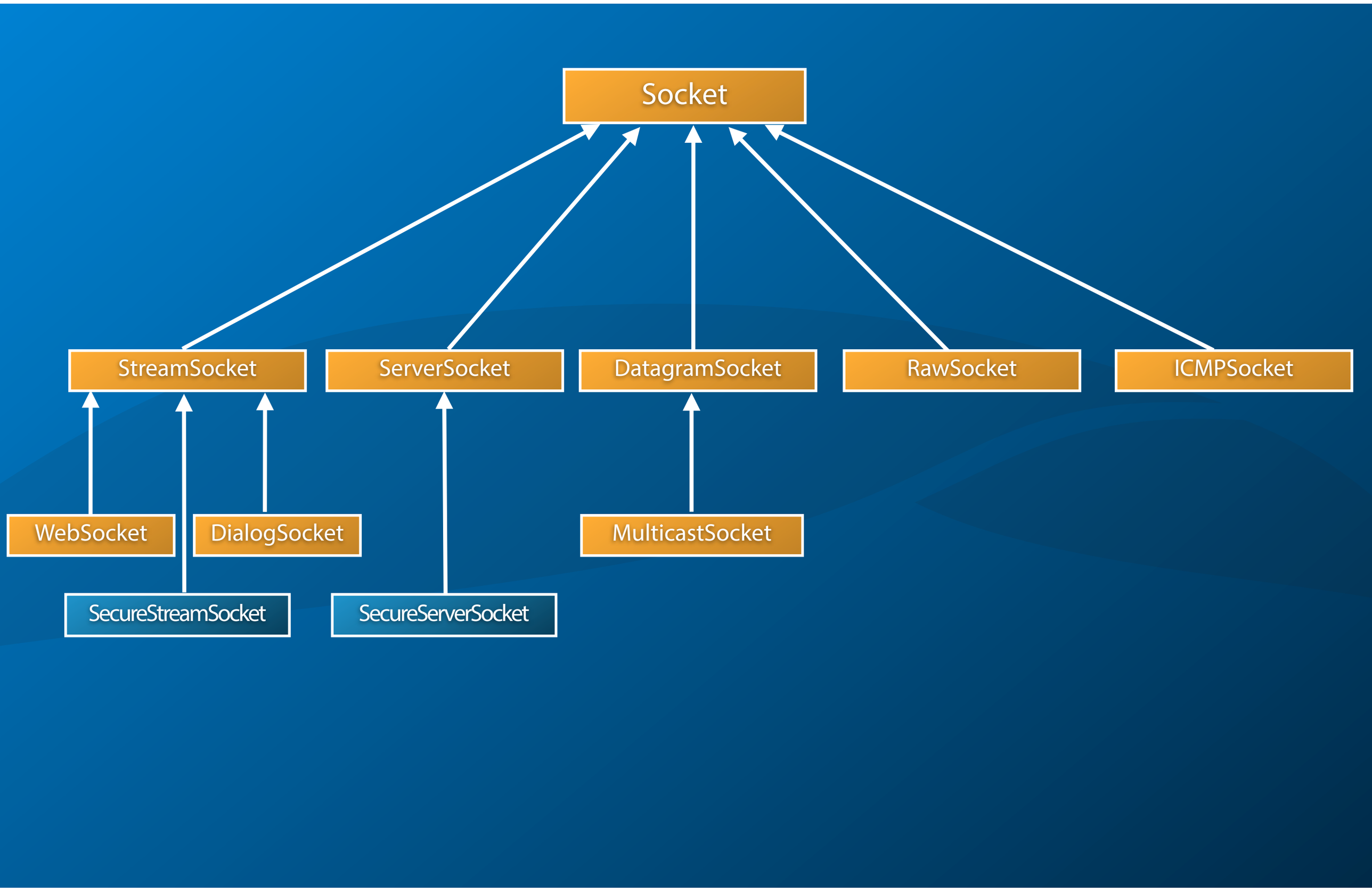
# Name Resolution

- 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

```
#include "Poco/Net/DNS.h"
#include <iostream>

using Poco::Net::DNS;
using Poco::Net::IPAddress;
using Poco::Net::HostEntry;

int main(int argc, char** argv)
{
    const HostEntry& entry = DNS::hostByName("www.appinf.com");
    std::cout << "Canonical Name: " << entry.name() << std::endl;

    const HostEntry::AliasList& aliases = entry.aliases();
    HostEntry::AliasList::const_iterator it = aliases.begin()
    for (; it != aliases.end(); ++it)
        std::cout << "Alias: " << *it << std::endl;

    const HostEntry::AddressList& addrs = entry.addresses();
    HostEntry::AddressList::const_iterator it = addrs.begin();
    for (; it != aliases.end(); ++it)
        std::cout << "Address: " << it->toString() << std::endl;

    return 0;
}
```

# Sockets

- The socket classes in POCO are implemented using the Pimpl idiom
- POCO sockets are a thin layer on top of BSD sockets with a minimal performance overhead – 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


Socket
StreamSocket
ServerSocket
DatagramSocket
RawSocket
ICMPSocket
WebSocket
DialogSocket
MulticastSocket
SecureStreamSocket
SecureServerSocket

# 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, for example:
  - select() and poll() (epoll(), where available)
  - 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

- Poco::Net::StreamSocket is used for creating a TCP connection to a server
- Use
  - StreamSocket::send/receiveBytes() to send and receive data, or
  - SocketStream class - a standard C++ 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

- Poco::Net::ServerSocket is used to create a TCP server socket
- Too low level for end-user
- For an actual server, consider 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 sending multicast datagrams
- To receive multicast messages, join a multicast group using MulticastSocket::joinGroup()
- 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(sa);

    std::string syslogMsg;
    Poco::Timestamp now;
    syslogMsg = Poco::DateTimeFormatter::format(now,
                    "<14>%w %f %H:%M:%S Hello, world!");

    dgs.sendBytes(syslogMsg.data(), syslogMsg.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;
}
```

```
// MulticastSocket receive example

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


int main(int argc, char* argv[])
{
    Poco::Net::SocketAddress  address("239.255.255.250", 1900);
    Poco::Net::MulticastSocket socket(
        Poco::Net::SocketAddress(
            Poco::Net::IPAddress(), address.port()
        )
    );

    // to receive any data you must join
    socket.joinGroup(address.host());

    Poco::Net::SocketAddress sender;
    char buffer[512];

    int n = socket.receiveFrom(buffer, sizeof(buffer), sender);
    socket.sendTo(buffer, n, sender);

    return 0;
}
```

# Web Sockets

- RFC 6455
- Standardized way for web server to send unsolicited data to the browser
- Standardization of the “comet” hack
- Used when server notifications to web client(s) are needed

# ICMP Package

- Internet Control Message Protocol (a.k.a. PING) support.
- Direct and event-based

```
// Direct IPv4 ping

#include "Poco/Net/ICMPClient.h"

using Poco::Net::ICMPClient

int main(int, char**)
{
    if (ICMPClient::pingIPv4("localhost")) return 0;
    else return -1;
}
```

```
// Event-based IPv4 ping
#include "Poco/Net/ICMPClient.h"
#include "Poco/Net/IPAddress.h"
#include "Poco/Net/ICMPEventArgs.h"
#include “Poco/Delegate.h”
#include <iostream>

using Poco::Net::ICMPClient;
using Poco::Net::IPAddress;
using Poco::Net::ICMPEventArgs;
using Poco::Delegate;

struct Ping
{
	void onReply(ICMPEventArgs& args)
	{
		std::cout << "Reply from " << args.hostAddress()
			<< " bytes=" << args.dataSize()
			<< " time=" << args.replyTime() << "ms"
			<< " TTL=" << args.ttl() << std::endl;
	}
}

int main(int argc, char** argv)
{
	Ping p;
	ICMPClient icmpClient(IPAddress::IPv4);

	icmpClient.pingReply += Delegate<Ping, ICMPEventArgs>(&p, &Ping::onReply); //register
	icmpClient.ping("localhost”);
	icmpClient.pingReply -= Delegate<Ping, ICMPEventArgs>(&p, &Ping::onReply); //unregister

	return 0;
}
```

# The TCPServer Framework

- Poco::Net::TCPServer implements a multithreaded TCP server
- Server uses a ServerSocket to accept incoming connections; ServerSocket must be in listening mode before passed to the TCPServer
- Server maintains a queue for incoming connections
- A variable number of worker threads fetches connections from the queue to process them. 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
  - uses TCPServerConnection objects to handle a connection.
  - user must create a subclass of TCPServerConnection and a factory for it
  - the factory object is passed to the constructor of TCPServer

# The TCPServer Framework (cont'd)

- A subclass of TCPServerConnection; must override the run() method, which handles 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

# The Reactor Framework

- The Reactor framework is based on the Reactor design pattern, described by Douglas C. Schmidt in PLOP
- Non-blocking sockets and select() (poll(), epoll() where available) combined with a NotificationCenter
- The Poco::Net::SocketReactor observes the state of an arbitrary number of sockets, and dispatches a notification whenever a socket state 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

# The HTTPServer Framework

- POCO contains a ready-to-use HTTP Server framework
  - multithreaded
  - HTTP 1.0/1.1
  - authentication support
  - cookie support
  - HTTPS by using the NetSSL library

# HTTPServer

- Configurable multi-threading (uses thread pool)
  - maximum number of threads
  - queue size for pending connections

- Similar to TCPServer
  - expects a HTTPRequestHandlerFactory, which creates HTTPRequestHandler based on the URI

# HTTPRequestHandlerFactory

- Manages all known HTTPRequestHandlers
- Decides which request handler will answer a request
- Can be used to check cookies, authentication info (usually done by the request handlers)

```
Poco::UInt16 port = 9999;
HTTPServerParams* pParams = new HTTPServerParams;
pParams->setMaxQueued(100);
pParams->setMaxThreads(16);

ServerSocket svs(port); // set-up a server socket

HTTPServer srv(new MyRequestHandlerFactory(), svs, pParams);

// start the HTTPServer
srv.start();

waitForTerminationRequest();

// Stop the HTTPServer
srv.stop();
```

```
#include "Poco/Net/HTTPRequestHandlerFactory.h"
#include "Poco/Net/HTTPServerRequest.h"
#include "RootHandler.h"
#include "DataHandler.h"

class MyRequestHandlerFactory: public
Poco::Net::HTTPRequestHandlerFactory
{
public:
    MyRequestHandlerFactory()
    {
    }

    Poco::Net::HTTPRequestHandler* createRequestHandler(
        const Poco::Net::HTTPServerRequest& request)
    {
        if (request.getURI() == "/")
            return new RootHandler();
        else
            return new DataHandler();
    }
};
```

# HTTPServerRequest

- Created by the server
- Passed as parameter to the HTTPRequestHandler/-Factory
- Contains:
  - URI
  - cookies
  - authentication information
  - HTML form data

# HTTPServerResponse

- Created by server, initialized by the request handler
- Sets:
  - cookies
  - response content type
    `response.setContentType("text/html");`
  - either
    - length of the content `response.setContentLength(1024);`
    - chunked transfer encoding
      `response.setChunkedTransferEncoding(true);`

# HTTPServerResponse (cont)

- Set response type

```
response.setStatus[AndReason](
        HTTPResponse::HTTP_OK); // default
response.setStatus[AndReason](
        HTTPResponse::HTTP_UNAUTHORIZED)
```

- After response is fully configured, header can be sent

```
std::ostream& out = response.send();
```

- If required, write data content to the returned stream
- Invoke send() only once !

```
class RootHandler: public Poco::Net::HTTPRequestHandler
{
public:
    void handleRequest(Poco::Net::HTTPServerRequest& request,
                       Poco::Net::HTTPServerResponse& response)
    {
        Application& app = Application::instance();
        app.logger().information("Request from " +
            request.clientAddress().toString());

        response.setChunkedTransferEncoding(true);
        response.setContentType("text/html");

        std::ostream& ostr = response.send();

        ostr << "<html><head><title>HTTP Server powered by POCO C++
                 Libraries</title></head>";
        ostr << "<body>";
        [...]
        ostr << "</body></html>";
    }
};
```

# Handling Cookies

- Support for handling cookies is provided by the Poco::Net::HTTPCookie class, as well as by the Poco::Net::HTTPRequest and Poco::Net::HTTPResponse classes

```
// set cookie in response
Poco::Net::HTTPCookie cookie(“name”, “Peter”);
cookie.setPath(“/”);
cookie.setMaxAge(3600); // 1 hour
response.addCookie(cookie);


// extract cookie from request
Poco::Net::NameValueCollection cookies;
request.getCookies(cookies);
Poco::Net::NameValueCollection::ConstIterator it = cookies.find(“name”);
if (it != cookies.end())
    std::string userName = it->second;
```

# Handling Credentials

```
void handleRequest(Poco::Net::HTTPServerRequest& request,
                   Poco::Net::HTTPServerResponse& response)
{
     if(!request.hasCredentials())
     {
          response.requireAuthentication("My Realm");
          response.setContentLength(0);
          response.send();
          return;
     }
     else
     {
          Poco::Net::HTTPBasicCredentials cred(request);
          const std::string& user = cred.getUsername();
          const std::string& pwd = cred.getPassword();
          [...]
     }
}
```

# HTMLForm

- Helper class to handle HTML form data
  Poco::Net::HTMLForm form(request);
  form[“entry1”] == “somedata”;
- To handle file uploads (POST with attachments), combine it with a Poco::Net::PartHandler:

```
MyPartHandler myHandler;
Poco::Net::HTMLForm form(request, request.stream(), myHandler);
```

```
#include “Poco/Net/PartHandler.h”
#include “Poco/Net/MessageHeader.h”


class MyPartHandler: public Poco::Net::PartHandler
{
public:
    void handlePart(const Poco::Net::MessageHeader& header,
                    std::istream& stream)
{
    _disp = header["Content-Disposition"];
    _type = header["Content-Type"];
     // read from stream and do something with it
}

private:
    std::string _disp;
    std::string _type;
};
```

# HTTP Client

- Poco::Net::HTTPClientSession
  - allows to send GET/POST requests
  - authentication information
- Poco:Net::HTTPStreamFactory and Poco::StreamCopier to download content

```
using namespace Poco::Net;

HTTPClientSession s("www.somehost.com");
//s.setProxy("localhost", srv.port());
HTTPRequest request(HTTPRequest::HTTP_GET, "/large");
HTMLForm form;
form.add(“entry1”, “value1”);
form.prepareSubmit(request);
s.sendRequest(request);

HTTPResponse response;
std::istream& rs = s.receiveResponse(response);
StreamCopier::copyStream(rs, std::cout);
```

# [HTTP | FTP] StreamFactory

- Must register the factory at the Poco::URIStreamOpener

```
Poco::Net::HTTPStreamFactory::registerFactory();
Poco::Net::FTPStreamFactory::registerFactory();
```

- Allows to open any http, ftp or file URI with a one-liner:

```
std::auto_ptr<std::istream>
pStr(URIStreamOpener::defaultOpener().open(uri));

StreamCopier::copyStream(*pStr.get(), std::cout);
```

- See Poco/Net/samples/download for a complete example

# Networking Clients

## FTPClientSession

```
FTPClientSession session("ftp.server.com");

session.login("user", "password");
std::istream& istr = session.beginDownload("file.txt");
std::ostringstream dataStr;
StreamCopier::copyStream(istr, dataStr);
session.endDownload();
```

## SMTPClientSession

```
SMTPClientSession session("smtp.server.com");
session.login("localhost");

MailMessage message;
message.setSender("john.doe@no.where.com");
message.addRecipient(MailRecipient(MailRecipient::PRIMARY_RECIPIENT,
     "jane.doe@no.where.com", "Jane Doe"));
message.setSubject("Test Message");
message.setContent("Hello John\r\n");
session.sendMessage(message);
```

# Networking Clients (cont)

- POP3ClientSession

```
POP3ClientSession session("localhost", server.port());
session.login("user", "secret");

MailMessage message;
session.retrieveMessage(1, message);
session.close();
```

http://pocoproject.org

# applied informatics



www.appinf.com | info@appinf.com
T +43 4253 32596 | F +43 4253 32096