Network Programming

Writing network and internet applications.

applied informatics

Overview

- Network programming basics
- Sockets
- The TCP Server Framework
- The Reactor Framework
- High Level Protocols: HTTP, FTP and E-Mail

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.

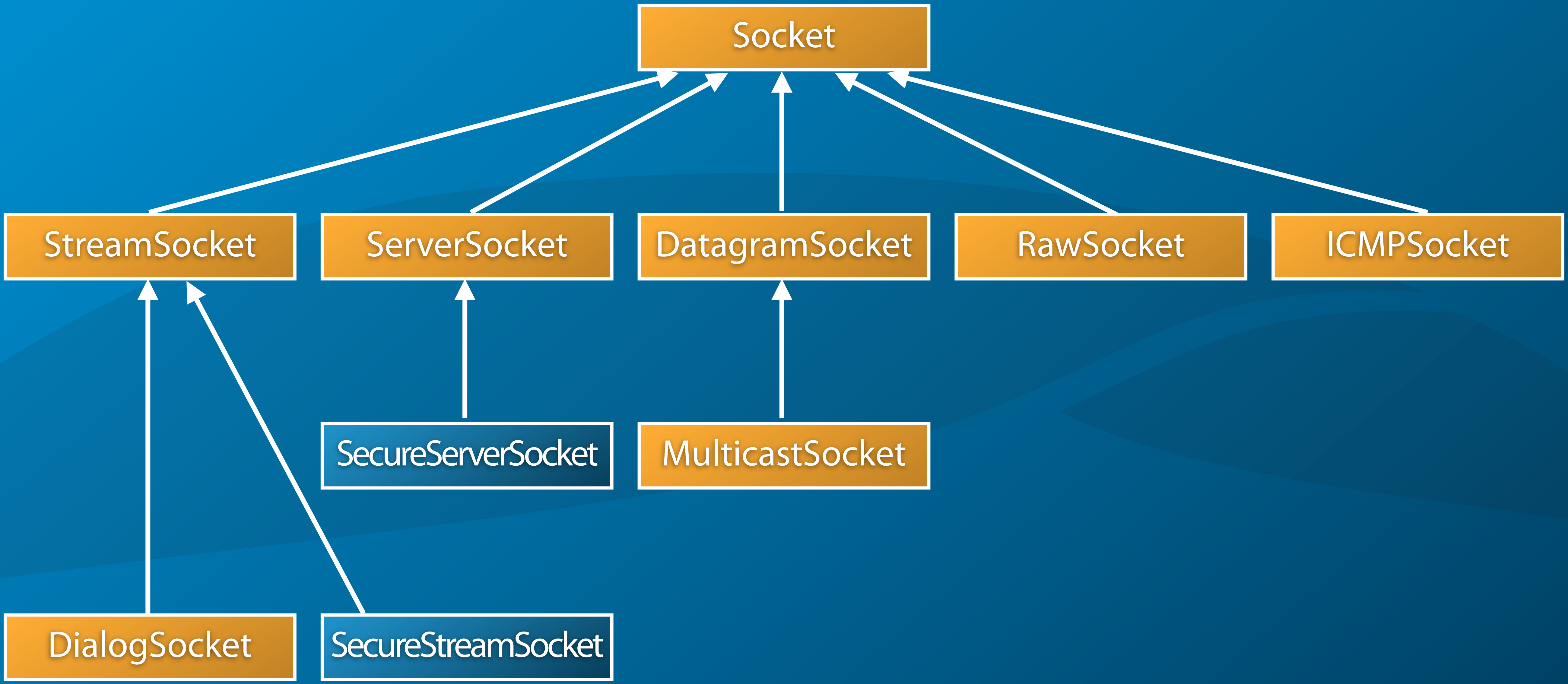
Name Resolution

- 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.

```
#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 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.


Socket
StreamSocket
ServerSocket
DatagramSocket
RawSocket
ICMPSocket
SecureServerSocket
MulticastSocket
DialogSocket
SecureStreamSocket

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

- 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

- 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(sa);

    std::string syslogMsg;
    Poco::Timestamp now;
    syslogMsg = 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;
}
```

```
// 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;
}
```

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.

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 occured).

> 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
  - maximum number of threads
  - uses thread pool
  - queue size for pending connections
- similar to TCPServer
  - expects a HTTPRequestHandlerFactory
  - which creates HTTPRequestHandler based on the URI

HTTPRequestHandlerFactory

- manages all known HTTPRequestHandlers
- sole purpose is to decide which request handler will answer a request
- can be used to check cookies, authentication info but this is mostly 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
  - authentification information
  - HTML form data

HTTPServerResponse

- created by the server but initialized by the request handler
  - set cookies
  - set content type of the answer
    `response.setContentType("text/html");`
  - either set the length of the content or use chunked transfer encoding

```
response.setContentLength(1024);
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, send the header

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

- if required, write data content to the returned stream

- `send()` must be invoked exactly 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) you must 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 you set GET/POST
  - authentication information
- Poco:Net::HTTPStreamFactory and Poco::StreamCopier if you only want to download something

```
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);
```

HTTPStreamFactory

- must register the factory at the Poco::URIStreamOpener

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

- allows to open all http URIs 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