dhorn sockets

# Introduction

This section of the library attempts to “C++-ify” network sockets. There were multiple design goals when writing the interface, the more important of which are outlined below:

* For I/O, it was initially desired that all of the sockets API take advantage of operator<< and operator>> definitions with std::iostream. However, it quickly became clear that this would not be the best idea in all cases as std::basic\_iostream and std::basic\_streambuf are templated with a character type. This makes sending and receiving some forms of data (e.g. UTF-8) not ideal. Thus, whenever possible, we defer to using
* Functionality/functions were modeled after Berkley/Windows sockets, but with the goal to avoid naming conflicts in the event of “using” statements.
* Sacrifice outdated/rarely used features of Berkley/Windows sockets in favor of simplicity. For example, due to the fundamental differences between TCP sockets and UDP sockets, their complete interface is defined differently.

# Classes/Structures/Types/Enumerations/etc.

When dealing with sockets in any operating system, errors are bound to arise. Therefore, the following error type is defined depending on the sockets API used by the compiler. This definition is for Windows Sockets:

using socket\_error\_t = int;

Additionally, there are various constants that need to be brought cross-platform. Here are the following definitions for Windows Sockets:

enum address\_family

{

af\_inet = AF\_INET,

af\_inet6 = AF\_INET6

};

## dhorn::ip\_address Class

There is one class incorporating both ipv4 and ipv6 addresses – dhorn::ip\_address – and two typedefs for each type: ipv4\_address and ipv6\_address. The latter two are defined as:

using ipv4\_address = ip\_address<af\_inet>;

using ipv6\_address = ip\_address<af\_inet6>;

The interface for the dhorn::ip\_address class is given as:

template <address\_family \_Af>

class ip\_address

{

public:

/\*

\* Type/Value Definitions

\*/

using ip\_addr = ...;

static const address\_family family = \_Af;

/\*

\* Constructor(s)/Destructor

\*/

ip\_address(void);

ip\_address(\_In\_ const ip\_addr &ip);

template <typename \_CharT>

ip\_address(\_In\_ const \_CharT \*ip);

template <typename \_CharT>

ip\_address(\_In\_ const std::basic\_string<\_CharT> &ip);

/\*

\* Assignment Operators

\*/

ip\_address &operator=(\_In\_ const ip\_addr &ip);

template <typename \_CharT>

ip\_address &operator=(\_In\_ const \_CharT \*ip);

template <typename \_CharT>

ip\_address &operator=(\_In\_ const std::basic\_string<\_CharT> &ip);

/\*

\* Type Conversion Operators

\*/

operator const ip\_addr &(void) const;

template <typename \_CharT>

operator std::basic\_string<\_CharT>(void) const;

/\*

\* Public Functions

\*/

const ip\_addr &addr(void) const;

template <typename \_CharT = char>

std::basic\_string<\_CharT> str(void) const;

bool fail(void) const;

socket\_error\_t error(void) const;

};

The ip\_addr type is the sockets API representation of ip addresses and is allowed to depend on the address family used for the address. This type definition and availability is for use by the various socket classes, and should not be used by the user except for testing and debugging purposes when inspecting the internal values is needed. In all other cases, ip\_address objects should be interfaced with using strings. All string operations can be performed with either character or wide character strings. Despite being declared as template functions, these are the only two types of strings that are supported. If an error occurs during assignment/construction, the fail function will return true and the error function will return the implementation specific error code. In all other cases (when fail returns false), the value returned by error is undefined and is allowed to take on any value. That is, there is no required “no error” value, though it is suggested that one be provided if the implementation allows for it.

## dhorn::udp\_socket Class