Boost Library

* **What is Boost Library?**
* It’s a set of [libraries](https://en.wikipedia.org/wiki/Library_(computing)) for the [C++](https://en.wikipedia.org/wiki/C%2B%2B) programming language developed by the Boost Organization.
* It is developed based on c++ standard libraries.
* It is open source library.
* It will have advanced features of c++ libraries.
* All these are platform independent and portable
* Peer reviewed libraries
* **Advantages over standard libraries?**
* It provides advanced features of c++ libraries. (ex: thread)
* It provides more features than standard c++ libraries.In fact, many parts of Boost is considered to be included in the next C++ standard library.
* It is documented nicely.
* More efficient.

**When we choose boost library?**

The Boost libraries are a good choice to increase productivity in C++ projects when your requirements go beyond what is available in the standard library. Because the Boost libraries evolve **faster** than the standard library, you have earlier access to new developments, and you don’t need to wait until those developments have been added to a new version of the standard library.

Example: C++11 Support in GCC. GCC 4.8.1

**List of Libraries:**

[Any](https://www.boost.org/doc/libs/release/libs/any/)

[Array](https://www.boost.org/doc/libs/release/libs/array/)

Thread

Smart ptr

Serialization

Filesystem

Date Time (posix time)

Json

Etc…..

* Boost::Filesystem:

Boost::filesystem library is used to work with files and directories.

HPP files are C++ Header Files or header files that are written in the C++ programming language.  This file format used by MarsDigital C++ (previously Zortech C++), Borland C++ and other C++ compilers.

#include <iostream>

#include <boost/filesystem.hpp>

#include <ctime>

using namespace std;

using namespace boost::filesystem;

#if 0

int main(int argc, char\* argv[])

{

cout << "File size - " << boost::filesystem::file\_size("main.cpp") << '\n';

return 0;

}

#elif 0

int main()

{

// auto x =20;

// std::cout << x;

path p1 {"//home//pf//Navita//Test\_Boost"};

std::cout << p1.relative\_path();

}

#elif 0

int main()

{

path p{"//home//pf//Navita//"};

cout << p.root\_name() << endl;

cout << p.root\_directory() << endl;

cout << p.root\_path() << endl;

cout << p.relative\_path() << endl;

cout << p.parent\_path() << endl;

cout << p.filename() << endl;

path photo {"index.jpeg"};

cout << photo.root\_name() << endl;

cout << photo.root\_directory() << endl;

cout << photo.root\_path() << endl;

cout << photo.relative\_path() << endl;

cout << photo.parent\_path() << endl;

cout << photo.filename() << endl;

cout << photo.stem() << endl;

cout << photo.extension() << endl;

if (exists(p))

{

cout << "exists" << endl;

}

}

#elif 0

int main()

{

path p ("//home//pf//Navita//Links\_Help");

if (exists(p))

{

if (is\_regular\_file(p))

cout << p << " size is " << file\_size(p) << '\n';

else if (is\_directory(p))

cout << p << "is a directory\n";

}

else

cout << p << "does not exist\n";

return 0;

}

#elif 0

int main()

{

path p{"//home//pf//Navita//"};

try

{

file\_status s = status(p);

std::cout << is\_directory(s) << '\n';

}

catch (filesystem\_error &e)

{

std::cerr << e.what() << '\n';

}

}

#elif 0

int main ()

{

path p {"//home/pf//Navita//Links\_Help"};

if (is\_regular\_file (p))

{

time\_t t = last\_write\_time (p);

cout << ctime (&t);

}

}

#elif 0

int main ()

{

path p {"//home//pf//Navita//test1"};

// space\_info s = space (p);

// cout << s.capacity << " - " << s.free << " - " << s.available << endl;

create\_directory (p);

rename (p, "//home//pf//Navita//test2");

cout << remove ("//home//pf//Navita//test1");

}

#elif 0

#include <boost/filesystem.hpp>

#include <iostream>

using namespace boost::filesystem;

int main()

{

try

{

//std::cout << absolute("index.jpeg") << '\n';

std::cout << absolute("index.jpg", "Test/") << endl;

std::cout << current\_path () << endl;

}

catch (filesystem\_error &e)

{

std::cerr << e.what() << '\n';

}

}

#elif 1

//display .cpp files in a directory.

int main()

{

path p = current\_path ();

directory\_iterator iter (p);

/\* To retrieve the end of a directory,

\* the class must be instantiated with the default constructor.

\* \*/

for (;iter != directory\_iterator{};iter++)

{

path temp (\*iter);

if (temp.extension() == ".cpp")

{ cout <<\*iter<< endl; }

//cout << \*iter << endl;

}

}

#endif

==================================================================================

* Boost:Serialization:

**Serialization** is a process of converting an Object into stream of bytes so that it can be transferred over a network or stored in a persistent storage.

Deserialization is exact opposite - Fetch a stream of bytes from network or persistence storage and convert it back to the Object **with the same state**.

The main concept of Boost.Serialization is the *archive*. An archive is a sequence of bytes that represent serialized C++ objects.

22 (length of the signature)

serialization::archive (signature)

9 (archive version, 10 on boost 1.53)

#if 0

#include <iostream>

#include <fstream>

using namespace std;

int main ()

{

ofstream file;

file.open("/home/pf/Navita/Boost/serialization/Test.txt", ios::out | ios::trunc);

char info[100];

cin.getline (info, 100);

file << info;// << endl;

cin >> info;

cin.ignore();

cin.getline (info, 100);

file << info ;//<< endl;

cin.getline (info, 100);

file << info;// << endl;

file.close();

ifstream ifile;

ifile.open ("/home/pf/Navita/Boost/serialization/Test.txt", ios::in);

ifile >> info; // read till new line '\n'

cout << info;

ifile.close();

return 0;

}

#elif 0

#include <boost/archive/text\_oarchive.hpp>

#include <boost/archive/text\_iarchive.hpp>

#include <iostream>

#include <fstream>

#include <sstream>

using namespace boost::archive;

using namespace std;

stringstream ss;

//filestream, stringstream

void add\_to\_archive()

{

ofstream file{"/home/pf/Navita/Boost/serialization/archive.txt"};

// Constructors of archives expect an input or output stream as a parameter. The stream is used to serialize or restore data.

text\_oarchive oa{file};

// text\_oarchive oa{ss};

string str = "fdsfsdfdfsdfffsdffffffffffffffffffffffffffffffffff";

// oa << str; // adding to archive

//or

oa & str;

}

void load\_from\_archive()

{

ifstream file{"/home/pf/Navita/Boost/serialization/archive.txt"};

text\_iarchive ia{file};

// text\_iarchive ia{ss};

string str;// = nullptr;

//ia >> str;

//or

ia & str;

cout << str << '\n';

}

int main()

{

add\_to\_archive();

load\_from\_archive();

}

#elif 1

#include <iostream>

#include <boost/archive/binary\_iarchive.hpp>

#include <boost/archive/binary\_oarchive.hpp>

#include <sstream>

#include <string>

using namespace std;

using namespace boost::archive;

stringstream ss;

class Test

{

private:

int x;

char y;

string name;

friend class boost::serialization::access;

/\* For each class to be saved via serialization and to load class members

\* there must exist a function serialize.

\* \*/

template <typename Archive>

void serialize(Archive &ar, const unsigned int version)

{

ar & x;

ar & y;

ar & name;

}

public:

Test (int a, char b, string c) :

x(a), y(b), name (c)

{ }

void display ()

{

cout << x << " - " << y << " - " << name << endl;

}

};

void add\_to\_archive()

{

binary\_oarchive oa{ss};

Test obj (1, 'Z', "ARI");

oa & obj;

}

void load\_from\_archive()

{

binary\_iarchive ia{ss};

Test obj (0,0,"");

ia & obj;

obj.display();

}

int main()

{

add\_to\_archive();

load\_from\_archive();

}

#endif

serialize() is automatically called any time an object is serialized or restored. It should never be called explicitly and, thus, should be declared as private. If it is declared as private, the class boost::serialization::access must be declared as a friend to allow Boost.Serialization to access the member function.

==================================================================================

Reference links:

<https://www.boost.org/doc/libs/>

<https://theboostcpplibraries.com/>