Chapter . The Variadic Macro Data Library 1.3

Edward Diener

Table of Contents

Introduction

Welcome to the variadic macro data library version 1.3.

The variadic macro data library, or VMD for short, is a library of macros which provide important functionality for variadic macros as well as integrating variadic macros with the Boost preprocessor library (Boost PP). It integrates with Boost PP without changing the latter library in any way.

The functionality of the library may be summed up as:

1. Providing the means to extract any single token from the comma-separated data which makes up variadic macro data, as well as to calculate the number of tokens.

2. Convert variadic macro data to and from Boost PP data types.

3. Enhance the tuple functionality of Boost PP by providing a means of calculating the size of a tuple as well as by providing equivalent macros to Boost PP tuple macros which do not require the size of the tuple to be explicitly passed.

The library is a header only library and all macros in the library are in a single header, whose name is 'vmd.hpp'.

The library is dependent on Boost PP.

Variadic Macros

Variadic macros, as specified by C++0x, is a feature taken from the C99 specification. They are macros which take a final parameter denoted as '...' which represents one or more final arguments to the macro as a series of comma-separated tokens. In the macro ex-

pansion a special keyword of '__VA_ARGS__' represents the comma-separated tokens. This information when passed to a variadic macro I call 'variadic macro data', which gives its name to this library. The more general term 'variadic data' is used in this documentation to specify data passed to a macro which can contain any number of macro tokens as a single macro parameter, such as is found in Boost PP.

C99, and by implication C++0x, provides no built-in way of accessing a single token from the comma-separated list of variadic macro data. But this library does provide a means to do that among its other functionality.

Boost support

Boost until recently has had no generic support for variadic macros based on the ability of a given compiler/version to support C99's variadic macros. This support has now been added, as of Boost version 1.4.5, in the form of a macro which denotes that compiler support for variadic macros does not exist. This macro is BOOST_NO_VARIADIC_MACROS. The variadic macro data library will only work for a compiler/version combination when BOOST_NO_VARIADIC_MACROS is not defined.

Functionality

The design of Boost PP allows data, in the form of preprocessor tokens, to be grouped together into various data types, any one of which can be treated as a single preprocessor argument to a macro. A number of Boost PP macros accept data as a single argument. Each of these data types also has its own rich set of macros to manipulate the data. It is imperative when interoperating with Boost PP that data is able to be passed as a single argument, even though the data itself may consist of a number of preprocessing tokens.

In variadic macros the data to be passed as variadic macro data is a comma-separated list of arguments, each of which can be any preprocessing token.

Because the variadic macro data is more than a single token, in order to use variadic macro data with Boost PP, it is necessary to be able to convert the variadic macro data to a single argument Boost PP data type. One can do this either by converting the variadic macro data as a whole, by extracting any given token from the variadic macro data and use that as a single argument, or by combining individual tokens from the variadic macro data into Boost PP data types using the functionality of the Boost PP data type to do so. VMD provides means to interoperate variadic macro data with Boost PP in these ways.

Outside of Boost PP interoperability, VMD allows individual tokens to be extracted from the variadic macro data and used in macro expansion or passed to other macros.

Finally through the functionality of variadic macros, VMD provides parallel functionality to the Boost PP tuple interface macros with a set of macros which do not need the size of a tuple to be specified.

Naming conventions

All of the macros in the library begin with the prefix BOOST_VMD_, where VMD stands for 'Variadic Macro Data'.

Following the prefix, certain names in the macros refer to data types in this library or Boost PP. These names and their data types are:

1. DATA = variadic macro data as represented by '...' in a variadic macro signature and __VA_ARGS__ in variadic macro expansion.
2. TUPLE = Boost PP tuple data type.
3. ARRAY = Boost PP array data type.
4. LIST = Boost PP list data type.
5. SEQ = Boost PP sequence data type.

I have used names in order to mimic the naming of Boost PP as closely as possible. Therefore I have used the terms SIZE and ELEM in the macro names because these are the terms in Boost PP to denote the number of tokens of data and the general name for a token.

Functional groups

The macros in VMD can best be explained as falling into four groups. These are:

1. Macros which directly support variadic macro data usage.
2. Macros which convert variadic macro data to Boost PP data types.
3. Macros which convert Boost PP data types to variadic macro data.
4. Macros which offer an easy to use replacement for Boost PP tuple macros because they do not require the size of the tuple to be specified.

A further general explanation of each of these groups follow, while a specific explanation for each macro can be found in the reference section.

Variadic Usage

There are two macros which enhance variadic macro data usage. These macros add functionality to variadic macros so that the number of comma-separated tokens in the variadic macro data can be calculated, and that any token among the variadic macro data's comma-separated tokens can be returned. The two macros are:

1. `BOOST_VMD_DATA_SIZE`(...), which returns the number of comma-separated tokens.
2. `BOOST_VMD_DATA_ELEM`(n,...), which returns a particular token among the comma-separated sequence. Here 'n' stands for the number of the token, starting with 0, which is returned from the variadic macro data.

Convert to Boost PP data types

There are four macros which convert variadic macro data as a whole to each of the four Boost PP data types. These are:

1. `BOOST_VMD_DATA_TO_PP_TUPLE`(...), which converts to a Boost PP tuple.
2. `BOOST_VMD_DATA_TO_PP_ARRAY`(...), which converts to a Boost PP array.
3. `BOOST_VMD_DATA_TO_PP_LIST`(...), which converts to a Boost PP list.
4. `BOOST_VMD_DATA_TO_PP_SEQ`(...), which converts to a Boost PP sequence.

Convert from Boost PP data types

There are four macros which convert each of the four Boost PP data types to variadic macro data. These are:

1. `BOOST_VMD_PP_TUPLE_TO_DATA`(tuple), which converts from a Boost PP tuple.
2. `BOOST_VMD_PP_ARRAY_TO_DATA`(array), which converts from a Boost PP array.
3. `BOOST_VMD_PP_LIST_TO_DATA`(list), which converts from a Boost PP list.
4. `BOOST_VMD_PP_SEQ_TO_DATA`(seq), which converts from a Boost PP sequence.

In these macros the data is returned as a comma-separated list of tokens, which is the format of variadic macro data. The results of any of these macros can be passed to variadic macros as the final parameter.

Boost PP tuple enhancements

There are six macros which manipulate Boost PP tuple data. The first is an addition to Boost PP functionality when dealing with tuples while the final five are direct replacements for Boost PP tuple data manipulation macros and which do not require the size of the tuple. These are:

1. `BOOST_VMD_PP_TUPLE_SIZE`(tuple), which returns the size of the tuple.
2. `BOOST_VMD_PP_TUPLE_ELEM`(tuple), which is a replacement for BOOST_PP_TUPLE_ELEM without having to pass the size of the tuple as the first parameter.
3. `BOOST_VMD_PP_TUPLE_REM_CTOR`(tuple), which is a replacement for BOOST_PP_TUPLE_REM_CTOR without having to pass the size of the tuple as the first parameter.
4. `BOOST_VMD_PP_TUPLE_REVERSE`(tuple), which is a replacement for macroref BOOST_PP_TUPLE_REVERSE without having to pass the size of the tuple as the first parameter.
5. `BOOST_VMD_PP_TUPLE_TO_LIST`(tuple), which is a replacement for BOOST_PP_TUPLE_TO_LIST without having to pass the size of the tuple as the first parameter.
6. `BOOST_VMD_PP_TUPLE_TO_SEQ`(tuple), which is a replacement for BOOST_PP_TUPLE_TO_SEQ without having to pass the size of the tuple as the first parameter.

Variadic Macro Data Reference

Header <boost/variadic_macro_data/vmd.hpp>

```
BOOST_VMD_DATA_SIZE(...)
BOOST_VMD_DATA_ELEM(n, ...)
BOOST_VMD_DATA_TO_PP_TUPLE(...)
BOOST_VMD_DATA_TO_PP_ARRAY(...)
BOOST_VMD_DATA_TO_PP_LIST(...)
BOOST_VMD_DATA_TO_PP_SEQ(...)
BOOST_VMD_PP_TUPLE_SIZE(tuple)
BOOST_VMD_PP_TUPLE_ELEM(n, tuple)
BOOST_VMD_PP_TUPLE_REM_CTOR(tuple)
BOOST_VMD_PP_TUPLE_REVERSE(tuple)
BOOST_VMD_PP_TUPLE_TO_LIST(tuple)
BOOST_VMD_PP_TUPLE_TO_SEQ(tuple)
BOOST_VMD_PP_TUPLE_TO_DATA(tuple)
BOOST_VMD_PP_ARRAY_TO_DATA(array)
BOOST_VMD_PP_LIST_TO_DATA(list)
BOOST_VMD_PP_SEQ_TO_DATA(seq)
```

Macro BOOST_VMD_DATA_SIZE

BOOST_VMD_DATA_SIZE — Expands to the number of comma-separated variadic macro data arguments.

Synopsis

```
// In header: <boost/variadic_macro_data/vmd.hpp>

BOOST_VMD_DATA_SIZE(...)
```

Description

... = variadic macro data.

returns = the number of comma-separated variadic macro data arguments being passed to it.

The value returned can be between 1 and 64.

Macro BOOST_VMD_DATA_ELEM

BOOST_VMD_DATA_ELEM — Expands to a particular variadic macro data argument.

Synopsis

```
// In header: <boost/variadic_macro_data/vmd.hpp>

BOOST_VMD_DATA_ELEM(n, ...)
```

Description

n = number of the variadic macro data argument. The number starts from 0 to the number of variadic macro data arguments - 1. The maximum number for n is 63.

... = variadic macro data.

returns = the particular macro data argument as specified by n. The argument returned can be any valid preprocessing token.

Macro BOOST_VMD_DATA_TO_PP_TUPLE

BOOST_VMD_DATA_TO_PP_TUPLE — Expand to a Boost PP tuple data type.

Synopsis

```
// In header: <boost/variadic_macro_data/vmd.hpp>

BOOST_VMD_DATA_TO_PP_TUPLE(...)
```

Description

... = variadic macro data.

returns = a Boost PP library tuple data type.

You can use the result of this macro whenever you need to pass a Boost PP library tuple as data to a Boost PP library macro.

Macro BOOST_VMD_DATA_TO_PP_ARRAY

BOOST_VMD_DATA_TO_PP_ARRAY — Expand to a Boost PP array data type.

Synopsis

```
// In header: <boost/variadic_macro_data/vmd.hpp>

BOOST_VMD_DATA_TO_PP_ARRAY(...)
```

Description

... = variadic macro data.

returns = a Boost PP library array data type.

You can use the result of this macro whenever you need to pass a Boost PP library array as data to a Boost PP library macro.

Macro BOOST_VMD_DATA_TO_PP_LIST

BOOST_VMD_DATA_TO_PP_LIST — Expand to a Boost PP list data type.

Synopsis

```
// In header: <boost/variadic_macro_data/vmd.hpp>

BOOST_VMD_DATA_TO_PP_LIST(...)
```

Description

... = variadic macro data.

returns = a Boost PP library list data type.

You can use the result of this macro whenever you need to pass a Boost PP library list as data to a Boost PP library macro.

Macro BOOST_VMD_DATA_TO_PP_SEQ

BOOST_VMD_DATA_TO_PP_SEQ — Expand to a Boost PP sequence data type.

Synopsis

```
// In header: <boost/variadic_macro_data/vmd.hpp>

BOOST_VMD_DATA_TO_PP_SEQ(...)
```

Description

... = variadic macro data.

returns = a Boost PP library sequence data type.

You can use the result of this macro whenever you need to pass a Boost PP library sequence as data to a Boost PP library macro.

Macro BOOST_VMD_PP_TUPLE_SIZE

BOOST_VMD_PP_TUPLE_SIZE — Expands to the number of elements in a tuple.

Synopsis

```
// In header: <boost/variadic_macro_data/vmd.hpp>

BOOST_VMD_PP_TUPLE_SIZE(tuple)
```

Description

tuple = a Boost PP library tuple data type.

returns = the number of elements in the tuple, commonly referred to as the tuple size.

In the Boost PP library there is no way to calculate the size of a tuple, so that the size must be known in order to be used by Boost PP library tuple macros. With variadic macros the size of a tuple can be calculated from the tuple itself.

Macro BOOST_VMD_PP_TUPLE_ELEM

BOOST_VMD_PP_TUPLE_ELEM — Expands to a particular tuple element.

Synopsis

```
// In header: <boost/variadic_macro_data/vmd.hpp>

BOOST_VMD_PP_TUPLE_ELEM(n, tuple)
```

Description

n = number of the tuple element. The number starts from 0 to the size of the tuple - 1.

tuple = a Boost PP library tuple data type.

returns = the particular tuple element as specified by n.

In the Boost PP library there is no way to calculate the size of a tuple, so that the size must be known in order to be used by Boost PP library tuple macros. With variadic macros the size of a tuple can be calculated from the tuple itself.

Therefore this macro is a replacement for the BOOST_PP_TUPLE_ELEM macro without the necessity of having to pass a size.

Macro BOOST_VMD_PP_TUPLE_REM_CTOR

BOOST_VMD_PP_TUPLE_REM_CTOR — Expands to a series of tokens which are equivalent to removing the parentheses from a tuple.

Synopsis

```
// In header: <boost/variadic_macro_data/vmd.hpp>

BOOST_VMD_PP_TUPLE_REM_CTOR(tuple)
```

Description

tuple = a Boost PP library tuple data type.

returns = a series of comma-separated tokens equivalent to removing the parentheses from a tuple.

This result is actually equivalent to the form of variadic macro data and can be used as an alternative to BOOST_VMD_PP_TUPLE_TO_DATA to convert the tuple to variadic macro data.

In the Boost PP library there is no way to calculate the size of a tuple, so that the size must be known in order to be used by Boost PP library tuple macros. With variadic macros the size of a tuple can be calculated from the tuple itself.

Therefore this macro is a replacement for the BOOST_PP_TUPLE_REM_CTOR macro without the necessity of having to pass a size.

Macro BOOST_VMD_PP_TUPLE_REVERSE

BOOST_VMD_PP_TUPLE_REVERSE — Expands to a tuple whose elements are in reversed order.

Synopsis

```
// In header: <boost/variadic_macro_data/vmd.hpp>

BOOST_VMD_PP_TUPLE_REVERSE(tuple)
```

Description

tuple = a Boost PP library tuple data type.

returns = a tuple whose elements are in reversed order from the original tuple.

In the Boost PP library there is no way to calculate the size of a tuple, so that the size must be known in order to be used by Boost PP library tuple macros. With variadic macros the size of a tuple can be calculated from the tuple itself.

Therefore this macro is a replacement for the BOOST_PP_TUPLE_REVERSE macro without the necessity of having to pass a size.

Macro BOOST_VMD_PP_TUPLE_TO_LIST

BOOST_VMD_PP_TUPLE_TO_LIST — Expands to a list whose elements are the same as a tuple.

Synopsis

```
// In header: <boost/variadic_macro_data/vmd.hpp>

BOOST_VMD_PP_TUPLE_TO_LIST(tuple)
```

Description

tuple = a Boost PP library tuple data type.

returns = a list whose elements are the same as the tuple that is inputted.

In the Boost PP library there is no way to calculate the size of a tuple, so that the size must be known in order to be used by Boost PP library tuple macros. With variadic macros the size of a tuple can be calculated from the tuple itself.

Therefore this macro is a replacement for the BOOST_PP_TUPLE_TO_LIST macro without the necessity of having to pass a size.

Macro BOOST_VMD_PP_TUPLE_TO_SEQ

BOOST_VMD_PP_TUPLE_TO_SEQ — Expands to a sequence whose elements are the same as a tuple.

Synopsis

```
// In header: <boost/variadic_macro_data/vmd.hpp>

BOOST_VMD_PP_TUPLE_TO_SEQ(tuple)
```

Description

tuple = a Boost PP library tuple data type.

returns = a sequence whose elements are the same as the tuple that is inputted.

In the Boost PP library there is no way to calculate the size of a tuple, so that the size must be known in order to be used by Boost PP library tuple macros. With variadic macros the size of a tuple can be calculated from the tuple itself.

Therefore this macro is a replacement for the BOOST_PP_TUPLE_TO_SEQ macro without the necessity of having to pass a size.

Macro BOOST_VMD_PP_TUPLE_TO_DATA

BOOST_VMD_PP_TUPLE_TO_DATA — Expands to variadic macro data whose arguments are the same as a tuple's elements.

Synopsis

```
// In header: <boost/variadic_macro_data/vmd.hpp>

BOOST_VMD_PP_TUPLE_TO_DATA(tuple)
```

Description

tuple = a Boost PP library tuple data type.

returns = variadic macro data whose arguments are the same as the elements of a tuple that is inputted.

The variadic macro data that is returned is in the form of of comma separated arguments. The variadic macro data can be passed to any macro which takes variadic macro data in the form of a final variadic macro data '...' macro parameter.

Macro BOOST_VMD_PP_ARRAY_TO_DATA

BOOST_VMD_PP_ARRAY_TO_DATA — Expands to variadic macro data whose arguments are the same as an array's elements.

Synopsis

```
// In header: <boost/variadic_macro_data/vmd.hpp>

BOOST_VMD_PP_ARRAY_TO_DATA(array)
```

Description

array = a Boost PP library array data type.

returns = variadic macro data whose arguments are the same as the elements of an array that is inputted.

The variadic macro data that is returned is in the form of of comma separated arguments. The variadic macro data can be passed to any macro which takes variadic macro data in the form of a final variadic macro data '...' macro parameter.

Macro BOOST_VMD_PP_LIST_TO_DATA

BOOST_VMD_PP_LIST_TO_DATA — Expands to variadic macro data whose arguments are the same as a list's elements.

Synopsis

```
// In header: <boost/variadic_macro_data/vmd.hpp>

BOOST_VMD_PP_LIST_TO_DATA(list)
```

Description

list = a Boost PP library list data type.

returns = variadic macro data whose arguments are the same as the elements of a list that is inputted.

The variadic macro data that is returned is in the form of of comma separated arguments. The variadic macro data can be passed to any macro which takes variadic macro data in the form of a final variadic macro data '...' macro parameter.

Macro BOOST_VMD_PP_SEQ_TO_DATA

BOOST_VMD_PP_SEQ_TO_DATA — Expands to variadic macro data whose arguments are the same as a sequence's elements.

Synopsis

```
// In header: <boost/variadic_macro_data/vmd.hpp>

BOOST_VMD_PP_SEQ_TO_DATA(seq)
```

Description

seq = a Boost PP library sequence data type.

returns = variadic macro data whose arguments are the same as the elements of a sequence that is inputted.

The variadic macro data that is returned is in the form of of comma separated arguments. The variadic macro data can be passed to any macro which takes variadic macro data in the form of a final variadic macro data '...' macro parameter.

VMD and Boost PP

Boost PP already has the ability to pass variadic data as a single macro argument through any of its data types. It may then be reasonably asked why there is any need to use variadic macros to pass preprocessor data instead.

There are two considerations for using variadic macros:

1. The syntax for using variadic macros is the more natural syntax for passing macro arguments. Providing a comma-separated list of data mimics the way macro arguments are usually passed.

2. The length of the variadic data does not have to be passed. In Boost PP the length does not have to be passed for the sequences and lists, but it has to be specified as part of an array, and must be separately passed, or known in advance, for tuples. Functionality in this library, however, alleviates this last requirement for tuples.

On the other hand there are considerations for using Boost PP data types for passing variadic data to macros:

1. Boost PP data types can be passed multiple times in any macro whereas variadic macros can only pass its variadic macro data a single time as the final set of arguments to a macro.

2. Boost PP data types, which are single macro arguments, fit in well with Boost PP functionality.

3. Boost PP data types have a rich set of functionality for manipulating the data in the data type.

The more natural syntax of variadic macro data still provides enough importance, from the end-user's point of view, for using this library's facilities. A macro writer can design macros for the end-user which take variadic data using variadic macros while internally using Boost PP data types to manipulate that data and pass that data to other Boost PP macros. This library provides functionality to do just that with its macros which convert from variadic macro data to Boost PP data types.

An example design

We will design a macro for end-users which takes variadic data as its argument. Let's call this macro, just as an example, ENDUSER_MACRO.

Without variadic macro support, but in keeping with Boost PP, the best way of designing this macro is probably to use a Boost PP sequence. Our design of the macro might look like this:

```
#define ENDUSER_MACRO(ppSequence) ENDUSER_DETAIL_MACRO(ppSequence)
```

The reason for calling another macro which does the actual work of expansion will be explained below when discussing how to design this macro using VMD.

```
#define ENDUSER_DETAIL_MACRO(ppSequence) \
/* expansion which manipulates the data
  using the Boost PP facilities for a sequence.
  In Boost PP one can find out the size of the
  sequence, extract any token from the sequence,
  and much more...
*/
```

The end-user would pass data to this macro in this way:

```
ENDUSER_MACRO((a)(b)(c)(d)(e)) // etc. with each "token" in the sequence surrounded by ()
```

That is certainly acceptable, and without variadic macros, it is certainly excellent to have the Boost PP functionality that allows us to design macros taking variadic data and manipulate that data using the functionality of Boost PP.

With variadic macro support and VMD, but wishing to use our variadic data in exactly the same way as above, we could design our macro like this:

```
#define ENDUSER_MACRO(...) ENDUSER_DETAIL_MACRO(BOOST_VMD_DATA_TO_PP_SEQ(__VA_ARGS__))
```

Here we again call the macro which does the actual work. This is the reason why I designed my version of the macro without variadic macro support in the way that I did.

The end-user would pass data to this macro in this way:

```
ENDUSER_MACRO(a,b,c,d,e) // etc. with each token being comma-separated from each other
```

I think this last way of passing variadic data is more natural to an end-user than using a Boost PP sequence directly, but of course it depends on having compiler variadic macro support.

One decision to be made is whether to support, for any given variadic data macro functionality, a single macro or two macros with slightly different names.

1. Single macro design:

   In our example, if we wish to support a single macro, for compilers that both support or do not support variadic macros, the code would be:

   ```
   #if defined(BOOST_NO_VARIADIC_MACROS)
     #define ENDUSER_MACRO(ppSequence) ENDUSER_DETAIL_MACRO(ppSequence)
   #else
     #define ENDUSER_MACRO(...) ENDUSER_DETAIL_MACRO(BOOST_VMD_DATA_TO_PP_SEQ(__VA_ARGS__))
   #endif
   ```

   We would now have a single macro which would be used slightly differently by the end-user depending on whether the compiler being used supported variadic macros or not. This might not be best if the end-user's code needed to work for different compilers, some of which support variadic macros and some of which do not. In that latter case, a dual macro design (see below) might be better.

   Another solution supporting a single macro is to just ignore variadic macros, and then our solution would be:

   ```
   #define ENDUSER_MACRO(ppSequence) ENDUSER_DETAIL_MACRO(ppSequence)
   ```

We could also ignore any compilers which do not support variadic macros, and then our solution would be:

```
#if !defined(BOOST_NO_VARIADIC_MACROS)
  #define ENDUSER_MACRO(...) ENDUSER_DETAIL_MACRO(BOOST_VMD_DATA_TO_PP_SEQ(__VA_ARGS__))
#endif
```

2. Dual macro design:

   Perhaps best is to provide two macros with slightly different names. Our solution would then be:

```
  #define ENDUSER_MACRO(ppSequence) ENDUSER_DETAIL_MACRO(ppSequence)
#if !defined(BOOST_NO_VARIADIC_MACROS)
  #define ENDUSER_MACRO_VM(...) ENDUSER_DETAIL_MACRO(BOOST_VMD_DATA_TO_PP_SEQ(__VA_ARGS__))
#endif
```

   Here I have attached an '_VM' to the name of the macro which supports variadic macros.

In an ideal world, once the new C++ standard is ratified, all compilers will support variadic macros, and then we can design a single macro which takes variadic macro data.

Of course using variadic macro data works smoothly if there is only a single set of variadic data which a macro needs. If there is more than one set of variadic data, then we can consider using a combination of Boost PP variadic data functionality and variadic macro data. This is because variadic macro data can only be specified once for a variadic macro and must be the last parameter in the variadic macro.

Design

The initial impetus for creating this library was entirely practical. I had been working on another library of macro functionality, which used Boost PP functionality, and I realized that if I could use variadic macros with my other library, the end-user usability for that library would be easier. Therefore the main design goal of this library is to interoperate variadic macro data with Boost PP in the easiest and clearest way possible.

I also wanted to make the library as orthogonal and as easy to use as possible. Because of this, there is functionality in this library that is really not necessary for someone knowledgable about Boost PP, but it is included in the library even though it is just a convenient shorthand for functionality in Boost PP combined with new functionality in this library. This includes a number of the conversions back and forth between variadic macro data and Boost PP data types as well as nearly all of the specific Boost PP tuple functionality. As long as there is no run-time programming overhead, and a minimum of compile-time overhead, I value ease of use to be much more important than redundancy, so I added the functionality to this library.

Compilers

A previously mentioned in the section 'Boost Support' the compilers supported by this library must not have the Boost Config macro BOOST_NO_VARIADIC_MACROS defined. I have tested this library using gcc/MingW and VC++ on Windows. The compilers tested are gcc 3.3.3, 3.4.2, 3.4.5, 4.3.0, 4.4.0, 4.5.0-1, 4.5.2-1 and VC++ 8.0, 9.0, 10.0. Other compilers which currently should work with this library are gcc on Linux, Digital Mars C++, and Borland C++ Builder 6/Codegear C++.

As of Boost 1.4.5 the macro BOOST_NO_VARIADIC_MACROS is supported. This library could be used with previous Boost installations but there would be no guarantee that the compilers for which one wanted to use this library actually support variadic macros.

Limitations

The number of comma-separated macro parameters supported by the macros BOOST_VMD_DATA_SIZE(...) and BOOST_VMD_DATA_ELEM(n,...) is 64.

History

Version 1.3

- Moved version information and history into the documentation.
- Separate files for build.txt in the doc sub-directory and readme.txt in the top-level directory.
- Breaking changes
  - The name of the main header file is shortened to 'vmd.hpp'.
  - The library follows the Boost conventions.
    - Changed the filenames to lower case and underscores.
    - The macros now start with BOOST_VMD_ rather than just VMD_ as previously.

Version 1.2

- Added a readme.txt file.
- Updated all jamfiles so that the library may be tested and docs generated from its own local directory.

Version 1.1

- Added better documentation for using variadic data with Boost PP and VMD.

Version 1.0

Initial version of the libary.

Acknowledgements

First and foremost I would like to thank Paul Mensonides for providing advice, explanation and code for working with variadic macros and macros in general. Secondly I would like to thank Steve Watanabe for his help, code, and explanations. Finally I have to acknowledge that this library is an amalgam of already known techniques for dealing with variadic macros themselves, among which are techniques published online by Laurent Deniau. I have added design and some cleverness in creating the library but I could not have done it without the previous knowledge of others.

Index