Chapter . The Variadic Macro Data Library 1.5

Edward Diener

Table of Contents

Introduction

Welcome to the variadic macro data library version 1.5.

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.
4. Provide additional support for Boost PP data types and other functionality through the use of variadic macros.

The library is a header only library and all macros in the library are included by a single header, whose name is 'vmd.hpp'. Individual headers may be used for different functionality in the library and will be denoted when that functionality is explained.

All the macros in the library begin with the sequence 'BOOST_VMD_' to distinguish them from other macros the end-user might use.

The library is dependent on Boost PP and, in default 'native' mode, Boost Config.

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 expansion 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 Boost PP library has had no support for variadic macros but a version has been recently put in the Boost trunk which does offer support for variadic macros. This version uses the macro BOOST_PP_VARIADICS to denote that compiler support for variadic macros exist. There are variadic macros in this version of Boost PP whose functionality is taken from the variadic macro support in the VMD.

The two modes of the library

Because of variadic macro support in the Boost PP library currently in the Boost trunk, the VMD library can operate in one of two modes. In its 'native' mode, which is currently the default mode, the library does not forward to the equivalent variadic macro support in the Boost PP library. In its 'pplib' mode the VMD library does forward to the equivalent variadic macro support in the Boost PP library. The modes can be controlled by the macro BOOST_VMD_PPLIB. When this macro is defined and set to 1, the library operates in the 'pplib' mode, else it operates in its 'native' mode. The end-user of VMD can also use the macro BOOST_VMD_PPLIB to test the mode the VMD library is in.

When the Boost PP library variadic macro support currently in the Boost trunk is added to a future official version of Boost, the default mode in a subsequent version of VMD will be changed to 'pplib' mode and the end-user will be encouraged to use the functionality of the Boost PP library instead of the functionality of VMD which duplicates it. At that time the functionality in VMD which duplicates the same functionality in Boost PP will be deprecated.

Depending on the mode being used VMD determines whether variadic macro support is enabled for a particular compiler. The end-user of VMD can also manually use the macro BOOST_VMD_VARIADICS to turn on or off compiler support for variadic macro data in the VMD library. When BOOST_VMD_VARIADICS is set to 0 variadic macro data is not supported in the VMD library, otherwise when BOOST_VMD_VARIADICS is set to non-zero it is supported in the VMD library. This same macro can be used to determine if VMD supports variadic macros for a particular compiler.

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.

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.

All of the previously mentioned variadic macro functionality has been added to a version of Boost PP which is currently in the Boost trunk. In 'pplib' mode, all of this functionality become invocations to the Boost PP trunk equivalent macros.

VMD also offers further macro functionality, using variadic macros, centered on the verification of Boost PP data types, on the detection and removal of beginning parentheses in a macro parameter, and on the determination of whether a parameter is empty or not.

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 five 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.
5. Macros which offer further functionality for common Boost PP situations through the use of variadic macros.
6. Macros which control internal variadic data functionality.

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.

The end-user of VMD can include the individual header file 'vmd_data.hpp' instead of the general header file 'vmd.hpp' for using these macros.

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.

The end-user of VMD can include the individual header file 'vmd_data.hpp' instead of the general header file 'vmd.hpp' for using these macros.

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.

The end-user of VMD can include the individual header file 'vmd_to_data.hpp' instead of the general header file 'vmd.hpp' for using these macros.

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.

The end-user of VMD can include the individual header file 'vmd_tuple.hpp' instead of the general header file 'vmd.hpp' for using these macros.

Further functionality

There are macros which add further functionality related to Boost PP through the use of variadic macros. These can be divided into three groups:

Testing for emptiness

Currently Boost PP has an undocumented macro for testing whether a parameter is empty of not, written without the use of variadic macros. The macro is called BOOST_PP_IS_EMPTY. The macro is by its nature flawed, since there is no generalized way of determining whether or not a parameter is empty using the C++ preprocessor. But the macro will work given some limited inputs or emptiness. Paul Mensonides, the developer of Boost PP and the BOOST_PP_IS_EMPTY macro there, also wrote a better macro, using variadic macros, for determining whether or not a parameter is empty or not. This macro is also not perfect, since there is no perfect solution, but will work correctly with almost all input. I have adapted his code for the VMD and developed my own code to work with the Visual C++ compiler since Paul's code would not because of deficiencies in the Visual C++ preprocessor.

The macro is called `BOOST_VMD_IS_EMPTY` and will return 1 if its input is empty or 0 if its input is not empty. For the standard version the macro is a variadic macro which make take any input. For the Visual C++ version the macro is not a variadic macro and therefore takes a single parameter, although it uses variadic macros in order to work properly.

The one situation where the macro may not work properly is if its input is a function-like macro name. In that case the macro may either work incorrectly or give a compiler error.

The end-user of VMD can include the individual header file 'vmd_is_empty.hpp' instead of the general header file 'vmd.hpp' for using this macro.

Testing and removing parens

A common need when using macros and the Boost PP is determining whether a parameter begins with a set of parenthesis, within which tokens may exist. This is the form of a Boost PP tuple. All Boost PP data types begin this way, and both PP arrays and lists are tuples of a certain form, while a PP seq is a series of one or more single element tuples.

The macro `BOOST_VMD_IS_BEGIN_PARENS` takes as its input a parameter and determines whether the parameter begins with a set of parenthesis. It returns 1 if its input begins with a set of parenthesis, else it returns 0. For the standard version the macro is a variadic macro which make take any input. For the Visual C++ version the macro is not a variadic macro and therefore takes a single parameter, although it uses variadic macros in order to work properly.

There may be other tokens after the set of parameters that begins the input, and the macro still returns 1.

Another common need when using macros and the Boost PP is to remove the set of parenthesis from the beginning of a parameter. This is what `BOOST_VMD_REMOVE_PARENS` does. if the parameter does not contain a set of beginning parenthesis, the input parameter is returned as is, else the input parameter is returned with the beginning parenthesis removed. No other parenthesis after the possible beginning parenthesis is removed.

The end-user of VMD can include the individual header file 'vmd_is_begin_parens.hpp' instead of the general header file 'vmd.hpp' for using the BOOST_VMD_IS_BEGIN_PARENS macro and the individual header file 'vmd_remove_parens.hpp' instead of the general header file 'vmd.hpp' for using the BOOST_VMD_REMOVE_PARENS macro.

Asserting Boost PP data types

There is no way to determine whether a macro parameter is a given Boost PP data type. But one can come close to this functionality using variadic macros and the test for emptiness in the VMD library which the BOOST_VMD_IS_EMPTY macro gives. Therefore I have developed four macros, for each of the four Boost PP data types, which will produce a compiler error if a parameter is not a given data type, else will output nothing. These macros are:

1. `BOOST_VMD_ASSERT_IS_ARRAY`(param), asserts that 'param' is a Boost PP array
2. `BOOST_VMD_ASSERT_IS_LIST`(param), asserts that 'param' is a Boost PP list
3. `BOOST_VMD_ASSERT_IS_SEQ`(param), asserts that 'param' is a Boost PP seq
4. `BOOST_VMD_ASSERT_IS_TUPLE`(param), asserts that 'param' is a Boost PP tuple

Each of these act like a C++ assert-like macro. They do nothing if the 'param' is of the known type, else produce a compiler error. Furthermore these macros, like 'assert-like macros, only check their 'param' in debug mode. This, however, can be overridden by the BOOST_VMD_ASSERT_DATA macro. If it is defined and set to 1, the macros will always check their 'param', otherwise if it is set to 0 the 'param' is never checked.

The assert macros can be used in code to test whether a 'param' is of the expected Boost PP data type. It is not possible to create equivalent macros which return 1 when the 'param' is of the correct type, else return 0, because of the assert macros' necessary reliance on BOOST_VMD_IS_EMPTY, which even in the best of conditions is slightly flawed. However the flaw in BOOST_VMD_IS_EMPTY, where a function-like macro name could produce a compiler error, is made use of in the assert macros so that the compiler error produced in that case will only occur when the assert macro verifies that its 'param' is not the Boost PP data type being checked. Still there is the very slight flaw in the assert macros where its reliance on BOOST_VMD_IS_EMPTY could run into input being checked where a function-like macro name will produce the incorrect result from the BOOST_VMD_IS_EMPTY macro. However this flaw is thought to be so slight, and an end-user's chance of inputting such data so minimal, that it was felt that such assert-like macros would be beneficial to macro programmers using Boost PP and VMD.

Another point to be made is that Boost PP array and list are also tuples. So passing an array or list to BOOST_VMD_ASSERT_IS_TUPLE will assert that either is a valid tuple by not producing a compiler error.

A final point concerns using the assert macros with Visual C++. There is no way which I have been able to discover to produce a compiler error from within a macro using Visual C++ without producing invalid C++ output to trigger the error. Because of this the promise to produce no output in case of error from the assert macros must be amended for Visual C++. The compiler error when using Visual C++, and an assert macro fails, could be masked by using one of the assert macros in a situation where no output is generated by an enclosing macro.

Individual header files can be used for each of the assert macros instead of including the general header file 'vmd.hpp'. The individual header files are:

'vmd_assert_is_array.hpp' for the BOOST_VMD_ASSERT_IS_ARRAY macro 'vmd_assert_is_list.hpp' for the BOOST_VMD_ASSERT_IS_LIST macro 'vmd_assert_is_seq.hpp' for the BOOST_VMD_ASSERT_IS_SEQ macro 'vmd_assert_is_tuple.hpp' for the BOOST_VMD_ASSERT_IS_TUPLE macro.

Controlling internal usage

VMD has a few object-like macros which the end-user of the library can use to determine or change the way variadic macros are used in the library.

The macro BOOST_VMD_VARIADICS is used to denote whether variadic data support exists for the particular compiler the end-user is using with VMD. VMD automatically sets this macro to 0 if the compiler does not support variadic macros, else it sets this macro to non-zero if the compiler does support variadic macro data. An end-user of VMD can also use this macro in his own design to determine whether or not variadic macros are supported. Furthermore an end-user of VMD can set this macro to 0 or non-zero, before including a VMD header file, to force VMD to treat the particular compiler being used as not supporting or supporting variadic macros.

The macro BOOST_VMD_PPLIB controls the 'mode' VMD is operating in. When this macro is set to 0, which is currently the default, VMD is operating in its 'native' mode, whereas when this macro is set to non-zero VMD is operating in its 'pplib' mode. The end-user of VMD can set this macro to control the mode before including a VMD header file, or check the value if necessary after including a VMD header file.

The macro BOOST_VMD_ASSERT_DATA controls whether or not an assert macro will check its data. The default is that in compiler debug mode it will check the data while in compiler release mode it will not check its data. The end-user can change this by setting the macro to 0 to not check the data, or non-zero to check the data before including a VMD header file, or check the value if necessary after including a VMD header file.

Variadic Macro Data Reference

Header <boost/variadic_macro_data/vmd_assert_is_array_common.hpp>

```
BOOST_VMD_ASSERT_IS_ARRAY(array)
```

Macro BOOST_VMD_ASSERT_IS_ARRAY

BOOST_VMD_ASSERT_IS_ARRAY — Asserts that the parameter is a pplib array.

Synopsis

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

BOOST_VMD_ASSERT_IS_ARRAY(array)
```

Description

The macro checks that the parameter is a pplib array. If it is not a pplib array, it forces a compiler error.

The macro works through variadic macro support.

The macro normally checks for a pplib array only in debug mode. However an end-user can force the macro to check or not check by defining the macro BOOST_VMD_ASSERT_DATA to 1 or 0 respectively.

array = a possible pplib array.

returns = Normally the macro returns nothing.

If the parameter is a pplib array, nothing is output.

For VC++, because there is no sure way of forcing a compiler error from within a macro without producing output, if the parameter is not a pplib array the macro forces a compiler error by outputting invalid C++.

For all other compilers a compiler error is forced without producing output if the parameter is not a pplib array.

There is no completely fool-proof way to check if a parameter is empty without possible producing a compiler error if it is not. Because a macro checking if a parameter is a pplib array needs to perform such a check, the best that one can do is to create a compiler error if a parameter is not a pplib array rather than having a macro which returns 1 or 0, depending on whether a parameter is a pplib array.

Header <boost/variadic_macro_data/vmd_assert_is_list_common.hpp>

```
BOOST_VMD_ASSERT_IS_LIST(list)
```

Macro BOOST_VMD_ASSERT_IS_LIST

BOOST_VMD_ASSERT_IS_LIST — Asserts that the parameter is a pplib list.

Synopsis

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

BOOST_VMD_ASSERT_IS_LIST(list)
```

Description

The macro checks that the parameter is a pplib list. If it is not a pplib list, it forces a compiler error.

The macro works through variadic macro support.

The macro normally checks for a pplib list only in debug mode. However an end-user can force the macro to check or not check by defining the macro BOOST_VMD_ASSERT_DATA to 1 or 0 respectively.

list = a possible pplib list.

returns = Normally the macro returns nothing.

If the parameter is a pplib list, nothing is output.

For VC++, because there is no sure way of forcing a compiler error from within a macro without producing output, if the parameter is not a pplib list the macro forces a compiler error by outputting invalid C++.

For all other compilers a compiler error is forced without producing output if the parameter is not a pplib list.

There is no completely fool-proof way to check if a parameter is empty without possible producing a compiler error if it is not. Because a macro checking if a parameter is a pplib list needs to perform such a check, the best that one can do is to create a compiler error if a parameter is not a pplib list rather than having a macro which returns 1 or 0, depending on whether a parameter is a pplib list.

Header <boost/variadic_macro_data/vmd_assert_is_seq_common.hpp>

```


BOOST_VMD_ASSERT_IS_SEQ(seq)
```

Macro BOOST_VMD_ASSERT_IS_SEQ

BOOST_VMD_ASSERT_IS_SEQ — Asserts that the parameter is a pplib seq.

Synopsis

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

BOOST_VMD_ASSERT_IS_SEQ(seq)
```

Description

The macro checks that the parameter is a pplib seq. If it is not a pplib seq, it forces a compiler error.

The macro works through variadic macro support.

The macro normally checks for a pplib seq only in debug mode. However an end-user can force the macro to check or not check by defining the macro BOOST_VMD_ASSERT_DATA to 1 or 0 respectively.

seq = a possible pplib seq.

returns = Normally the macro returns nothing.

If the parameter is a pplib seq, nothing is output.

For VC++, because there is no sure way of forcing a compiler error from within a macro without producing output, if the parameter is not a pplib seq the macro forces a compiler error by outputting invalid C++.

For all other compilers a compiler error is forced without producing output if the parameter is not a pplib seq.

There is no completely fool-proof way to check if a parameter is empty without possible producing a compiler error if it is not. Because a macro checking if a parameter is a pplib seq needs to perform such a check, the best that one can do is to create a compiler error if a parameter is not a pplib seq rather than having a macro which returns 1 or 0, depending on whether a parameter is a pplib seq.

Header <boost/variadic_macro_data/vmd_assert_is_tuple_common.hpp>

```


BOOST_VMD_ASSERT_IS_TUPLE(tuple)
```

Macro BOOST_VMD_ASSERT_IS_TUPLE

BOOST_VMD_ASSERT_IS_TUPLE — Asserts that the parameter is a pplib tuple.

Synopsis

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

BOOST_VMD_ASSERT_IS_TUPLE(tuple)
```

Description

The macro checks that the parameter is a pplib tuple. If it is not a pplib tuple, it forces a compiler error.

The macro works through variadic macro support.

The macro normally checks for a pplib tuple only in debug mode. However an end-user can force the macro to check or not check by defining the macro BOOST_VMD_ASSERT_DATA to 1 or 0 respectively.

tuple = a possible pplib tuple.

returns = Normally the macro returns nothing.

If the parameter is a pplib tuple, nothing is output.

For VC++, because there is no sure way of forcing a compiler error from within a macro without producing output, if the parameter is not a pplib tuple the macro forces a compiler error by outputting invalid C++.

For all other compilers a compiler error is forced without producing output if the parameter is not a pplib tuple.

There is no completely fool-proof way to check if a parameter is empty without possible producing a compiler error if it is not. Because a macro checking if a parameter is a pplib tuple needs to perform such a check, the best that one can do is to create a compiler error if a parameter is not a pplib tuple rather than having a macro which returns 1 or 0, depending on whether a parameter is a pplib tuple.

Header <boost/variadic_macro_data/vmd_data_native.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(...)
```

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_data_native.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_data_native.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_data_native.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_data_native.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_data_native.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_data_native.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.

Header <boost/variadic_macro_data/vmd_is_begin_parens_common.hpp>

```


BOOST_VMD_IS_BEGIN_PARENS(...)
```

Macro BOOST_VMD_IS_BEGIN_PARENS

BOOST_VMD_IS_BEGIN_PARENS — Tests whether a parameter begins with a set of parentheses.

Synopsis

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

BOOST_VMD_IS_BEGIN_PARENS(...)
```

Description

The macro checks to see if the parameter begins with a set of parentheses surrounding any tokens.

.... = variadic param(s)

returns = 1 if the param begins with a set of parentheses, 0 if it does not.

The macro works through variadic macro support.

The code for the non-VC++ version of this is taken from a posting by Paul Mensonides.

This macro is not a test for a parameter which is only single set of parentheses surrounding any tokens (a Boost pplib tuple) since the parameter may have other tokens following the beginning set of parentheses and it will still return 1.

There is no completely safe way to test whether the param is a tuple. At best one can use BOOST_VMD_ASSERT_IS_TUPLE to cause a compiler error if the parameter is not a tuple.

Header <boost/variadic_macro_data/vmd_is_empty_common.hpp>

```
BOOST_VMD_IS_EMPTY(...)
```

Macro BOOST_VMD_IS_EMPTY

BOOST_VMD_IS_EMPTY — Tests whether a parameter is empty or not.

Synopsis

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

BOOST_VMD_IS_EMPTY(...)
```

Description

The macro checks to see if the parameter is empty or not. It returns 1 if the parameter is empty, else returns 0.

The macro works through variadic macro support.

The macro is not perfect, and can not be so. The problem area is if the parameter to be checked is a function-like macro name, in which case either a compiler error can result or a false result can occur.

This macro is a replacement, using variadic macro support, for the undocumented macro BOOST_PP_IS_EMPTY in the Boost pplib. The code is taken from a posting by Paul Mensonides of a variadic version for BOOST_PP_IS_EMPTY, and changed in order to also support VC++.

.... = variadic param(s)

returns = 1 if the param is empty, 0 if it is not

Header <boost/variadic_macro_data/vmd_remove_parens_common.hpp>

```
BOOST_VMD_REMOVE_PARENS(param)
```

Macro BOOST_VMD_REMOVE_PARENS

BOOST_VMD_REMOVE_PARENS — Removes the set of parens from the start of a parameter if it has any.

Synopsis

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

BOOST_VMD_REMOVE_PARENS(param)
```

Description

param = a macro parameter.

returns = the parameter with the beginning set of parens removed. If the parameter has no beginning set of parameters, the parameter is returned as is. If there are further sets of parens after the beginning set of parameters, they are not removed.

Header <boost/variadic_macro_data/vmd_to_data_native.hpp>

```


BOOST_VMD_PP_ARRAY_TO_DATA(array)
BOOST_VMD_PP_LIST_TO_DATA(list)
BOOST_VMD_PP_SEQ_TO_DATA(seq)
BOOST_VMD_PP_TUPLE_TO_DATA(tuple)
```

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_to_data_native.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_to_data_native.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_to_data_native.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.

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_to_data_native.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.

Header <boost/variadic_macro_data/vmd_tuple_native.hpp>

```


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)
```

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_tuple_native.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_tuple_native.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_tuple_native.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_tuple_native.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_tuple_native.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_tuple_native.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.

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 !BOOST_VMD_VARIADICS
     #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 for 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 BOOST_VMD_VARIADICS
     #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 BOOST_VMD_VARIADICS
     #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.

I have further designed the library to offer a few macro programming features which are impossible to do effectively without the use of variadic macros. But I have kept these features at a minimum because of the difficulty of using variadic macros with compilers, most notably Visual C++, whose implementation of variadic macros is substandard and therefore very difficult to get to work correctly when variadic macros must be used.

Compilers

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.

In 'pplib' mode the compilers supported are those which are deemed to offer C99 variadic macro support for Boost PP.

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 1.5

- Added macros for verifying Boost PP data types.
- Added macros for detecting and removing beginning parens.
- Added a macro for testing for the emptiness of a parameter.
- Added support for individual header files.
- Added support for 'native' and 'pplib' modes.

Version 1.4

- Removed internal dependency on BOOST_PP_CAT and BOOST_PP_ADD when using VC++.

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